Université Paris Cité

École doctorale Science Informatique de Paris Centre (ED386)
Institut de Recherche en Informatique Fondamentale (UMR 8243)

Efficient Secure Computation from Correlated Pseudorandomness

Par DUNG BUI

Thèse de doctorat de SPÉCIALITÉ INFORMATIQUE

Dirigée par
GEOFFROY COUTEAU,
et SOPHIE LAPLANTE
Présentée et soutenue publiquement le 18/03/2025

Devant un jury composé de :

Carsten Baum	Associate Professor	DTU Compute, Denmark	Rapporteur
Benny Pinkas	Professor	Bar-Ilan University, Israel	Rapporteur
Elette Boyle	Associate Professor,	Reichman University, Israel	Examinatrice
	Senior Scientist	NTT Research, USA	
Duong-Hieu Phan	Professeur	Telecom Paris, France	Examinateur
Fabien Laguillaumie	Professeur	Université de Montpellier, France	Examinateur
Peter Scholl	Associate Professor	Aarhus University, Denmark	Examinateur
Geoffroy Couteau	Chargé de Recherche	CNRS, IRIF, Université Paris Cité	Co-directeur de thèse
Sophie Laplante	Professeure	IRIF, Université Paris Cité	Directrice de thèse

Abstract

In this thesis, we put forward advancements in Multi-Party Computation (MPC), enabling parties to jointly compute arbitrary functions while preserving the privacy of their inputs. Our focus is on MPC with correlated randomness, where efficiency is improved through correlations generated using preprocessing. We investigate two key paradigms for generating such randomness: Pseudorandom Correlation Generators (PCGs) and Pseudorandom Correlation Functions (PCFs). First, we propose new PCG-based constructions for Private Set Intersection (PSI), achieving either tighter security or improved performance. We then explore the applications of PCGs and PCFs in Zero-Knowledge Proofs (ZKPs), addressing both interactive and non-interactive settings. For interactive ZKPs, we design a privately verifiable protocol for circuit satisfiability with sublinear communication and efficient computation. In the non-interactive setting, we introduce a new designated-verifier ZKP (DV-NIZK) leveraging PCF with a public-key structure (PK-PCF). Finally, we introduce a novel MPC framework for binary circuits, utilizing a PCG optimized for small fields.

Keywords: Multi-Party Computation, Pseudorandom Correlation Generator, Pseudorandom Correlation Function, Zero-Knowledge Proof.

Résumé

Dans cette thèse, nous explorons les avancées en calcul multipartite sécurisé (MPC), permettant à plusieurs parties de calculer conjointement des fonctions arbitraires tout en préservant la confidentialité de leurs entrées. Notre étude se concentre sur le MPC avec aléa corrélé, où l'efficacité est améliorée grâce à des corrélations générées en phase de prétraitement. Nous examinons deux paradigmes clés pour générer ces corrélations: les Générateurs de Corrélations Pseudoaléatoires (PCGs) et les Fonctions de Corrélations Pseudoaléatoires (PCFs). Tout d'abord, nous proposons de nouvelles constructions de PCG pour l'Intersection Privée d'Ensembles (PSI), offrant une sécurité renforcée ou de meilleures performances. Nous explorons ensuite les applications des PCGs et PCFs aux preuves à divulgation nulle de connaissance (ZKPs), en couvrant à la fois les cadres interactifs et non interactifs. Pour les ZKPs interactives, nous concevons un protocole vérifiable de manière privée pour la satisfaisabilité de circuits, avec une communication sous-linéaire et un calcul efficace. Dans le cadre non interactif, nous introduisons une nouvelle preuve à divulgation nulle de connaissance pour vérificateur désigné (DV-NIZK) exploitant les PCF avec une structure à clé publique (PK-PCF). Enfin, nous introduisons un nouveau cadre MPC pour les circuits binaires, basé sur un PCG optimisé pour les petits corps finis.

Mots Clés : Calcul Multipartite Sécurisé, Correlation Pseudo-aléatoire, Generateurs de Corrélations Pseudo-aléatoires, Fonction Pseudo-aléatoire Correlées, Preuve à Divulgation Nulle de Connaissance.

Introduction en Français

La *cryptographie moderne* est devenue un pilier essentiel à l'ère numérique, sécurisant les communications, protégeant les informations sensibles et garantissant la confidentialité dans un monde interconnecté. Son champ d'application dépasse le simple chiffrement, englobant une vaste gamme d'objectifs : la confidentialité, l'intégrité, l'authentification, la non-répudiation et l'anonymat. Ces objectifs sont atteints grâce à des outils cryptographiques sophistiqués, tels que les algorithmes de chiffrement, les fonctions de hachage, les signatures numériques et les preuves à divulgation nulle de connaissance (ZKPs). Parmi ces outils, le calcul multipartite sécurisé (MPC) se distingue comme une technique révolutionnaire pour une collaboration respectueuse de la vie privée, permettant à plusieurs parties d'effectuer des calculs sur des données sensibles sans les exposer aux autres parties.

Le MPC est particulièrement efficace dans de nombreuses applications du monde réel. Dans le domaine de la santé, il facilite l'analyse collaborative des données des patients tout en préservant leur confidentialité. Par exemple, il permet à différents hôpitaux d'agréger leurs données afin de poursuivre des études sur des maladies, sans pour autant compromettre les dossiers des patients. De même, dans le secteur bancaire, les institutions utilisent le MPC pour détecter des fraudes sur différents comptes tout en protégeant les informations des clients. Une autre application se trouve dans les recommandations de contenu, où les plateformes de streaming pourraient utiliser le MPC pour suggérer des films ou des chansons sans accéder directement aux préférences des utilisateurs. Au-delà de ces exemples, le MPC renforce la gestion des clés cryptographiques en distribuant les clés de manière sécurisée et permet d'établir le vote électronique en protégeant la confidentialité des électeurs.

Le Calcul Sécurisé dans le Modèle de Corrélation Aléatoire

Le calcul multipartite sécurisé (MPC), sous-domaine de la cryptographie visant à préserver la confidentialité des informations privées dans le cadre d'une fonctionnalité publique f, permet à N parties disposant d'entrées privées $(x_1, \ldots, x_N)$ de calculer en toute sécurité $f(x_1, \ldots, x_N)$, tout en dissimulant les autres informations concernant leurs entrées privées aux coalitions de parties corrompues.

Le MPC a été introduit dans les travaux fondateurs de Yao [Yao86] et Goldreich, Micali et Wigderson (GMW) [GMW87], ouvrant la voie à un vaste ensemble de recherches qui ont établi les bases du calcul multipartite sécurisé.

L'efficacité des protocoles MPC peut être optimisée grâce à un *prétraitement indépendant des entrées.* Concrètement, si les parties peuvent générer à l'avance de nombreuses instances de corrélations aléatoires, puis les utiliser dans une phase en ligne, le coût de l'exécution du protocole MPC est alors *considérablement réduit* en termes de communication et de calcul. C'est justement le cas du protocol GMW [GMW87], basé sur le calcul sécurisé via le partage secret. Dans un protocole MPC basé sur le partage secret, les parties détiennent des parts des entrées et calculent de manière itérative des parts de la fonction, en utilisant sa répresentation en circuit, porte par porte.

Comme les portes additives peuvent être calculées localement par les parties détenant les parts des entrées, seules les portes multiplicatives nécessitent une interaction entre les parties pour être évaluées. Ainsi, le principal goulot d'étranglement des protocoles MPC provient de la communication nécessaire pour évaluer les portes multiplicatives dans un circuit.

Cependant, un avantage clé du MPC basé sur le partage secret, identifié pour la première fois dans les travaux de Beaver [Bea92], réside dans le fait que les multiplications sécurisées peuvent être *prétraitées* dans une phase de pré-calcul *indépendante des entrées*. En particulier, les parties peuvent générer de manière sécurisée des “triplets de Beaver”, des parts additives de $(a, b, a \cdot b) \in \mathbb{F}^3$ où $\mathbb{F}$ est un corps fini. Ensuite, pour chaque porte multiplicative à calculer dans la phase en ligne, les parties peuvent exécuter un protocole rapide de multiplication à sécurité théorie de l'information, utilisant un triplet de Beaver et impliquant la communication de seulement deux éléments de $\mathbb{F}$ par partie.

Ce modèle de calcul sécurisé avec prétraitement, en raison de l'efficacité de la phase en ligne, constitue la base des protocoles MPC modernes. Cependant, ce paradigme de prétraitement ne fait que déplacer le goulot d'étranglement de l'inefficacité du MPC vers la phase hors ligne, qui consiste à générer de nombreuses corrélations aléatoires, telles que les triplets de Beaver. De plus, bien que la communication hors ligne puisse être peu coûteuse, le stockage d'importantes quantités de corrélations aléatoires pour chaque interaction future potentielle peut néanmoins représenter un défi.

Ainsi, une question clé pour un MPC efficace est de savoir si nous pouvons générer des corrélations aléatoires avec une communication sous-linéaire par rapport au nombre de corrélations, tout en minimisant les besoins en stockage.

Génération de corrélations pseudoaléatoires. Récemment, un nouveau paradigme a émergé, permettant la génération *silencieuse* de longues chaînes de *pseudoaléa* corrélées [BCG+18; BCG+19b; BCG+19a], éliminant ainsi presque entièrement la communication pendant la phase de prétraitement. Cela est rendu possible grâce à des primitives cryptographiques, telles que les *générateurs de corrélations pseudoaléatoires* (PCG) [BCG+19b] et les *fonctions corrélées pseudoaléatoires* (PCF) [BCG+20a].

Un PCG permet de compresser de longues corrélations en des graines corrélées courtes, qui peuvent ensuite être localement étendues en instances pseudoaléatoires de la corrélation cible. Plus formellement, un PCG est un couple d'algorithmes $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$, où $\mathsf{PCG.Gen}$ génère deux clés courtes $(\mathsf{k}_0, \mathsf{k}_1)$, et $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ produit une longue chaîne y_σ telle que (y_0, y_1) forme des échantillons pseudorandomisés de la corrélation cible. Les PCG permettent un calcul sécurisé silencieux de la manière suivante : en utilisant un petit protocole distribué pour générer les clés $(\mathsf{k}_0, \mathsf{k}_1)$, deux parties peuvent ensuite les étendre localement en longues chaînes pseudoaléatoires corrélées sans nécessiter de communication supplémentaire.

Les PCG souffrent toutefois d'une limitation importante : une fois les clés distribuées, les parties sont contraintes de générer *en une seule fois* une quantité a priori *fixée* de corrélations. Les PCF résolvent ce problème : une PCF est un couple d'algorithmes $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$, où $\mathsf{PCF.Gen}$ génère deux clés courtes $(\mathsf{k}_0, \mathsf{k}_1)$, et $\mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x)$ produit y_σ^x, où pour chaque nouvelle entrée x, (y_0^x, y_1^x) apparaît comme un nouvel échantillon de la corrélation cible. Ainsi, après avoir généré les clés $(\mathsf{k}_0, \mathsf{k}_1)$ de manière distribuée une fois pour toutes, deux parties peuvent générer à la volée une quantité quelconque de corrélations pour leurs calculs sécurisés futurs.

Corrélations utiles pour le MPC. Bien que générer des corrélations lors de la phase de prétraitement puisse améliorer l'efficacité de la phase en ligne du MPC, il est crucial d'examiner les types concrets de corrélations à deux parties nécessaires pour les protocoles MPC.

Par exemple, la corrélation de *Oblivious Transfer* (OT) aléatoire, dans laquelle une partie reçoit une paire de bits aléatoires (s_0, s_1) (ou plus généralement une paire de chaîne de bit) et l'autre partie reçoit la paire (b, s_b) pour un bit aléatoire b. La corrélation OT peut servir de base pour des protocoles MPC généraux sans majorité honnête [GMW87; Yao86]. D'autres types de corrélations utiles pour le MPC basé sur des circuits arithmétiques incluent les *Oblivious Linear Evaluation* (OLE) [ADI+17] et les triplets de multiplication (ou "triplets de Beaver") [Bea92; DPS+12]. Les travaux récents sur les PCG et les PCF ont permis des avancées significatives dans la génération d'OT, d'OLE ou de triples de Beaver :

- Pour la corrélation OLE sur un corps fini $\mathbb{F}$, c'est-à-dire une corrélation à deux parties (r_0, r_1) où $r_0 = (\Delta, u) \leftarrow\$ \mathbb{F}^2$ et $r_1 = (v, w) \in \mathbb{F}^2$, $v \leftarrow\$ \mathbb{F}$, et $w := \Delta u + v$.
- Une autre corrélation utile est l'OLE vectoriel (VOLE), une extension de l'OLE qui peut remplacer plusieurs OLE dans certaines applications [ADI+17]. Dans un VOLE de longueur n, une partie détient $(\Delta, \mathbf{u})$, et une autre détient $(\mathbf{v}, \mathbf{w})$, où $\Delta \leftarrow\$ \mathbb{F}$, $(\mathbf{u}, \mathbf{v}) \leftarrow\$ \mathbb{F}^n$, et $\mathbf{w} := \Delta\mathbf{u} + \mathbf{v} \in \mathbb{F}^n$. De plus, l'OT aléatoire peut être réalisé par une corrélation VOLE si $\mathbb{F} = \mathbb{F}_2$, ce qui fait que des VOLE basés sur des PCG/PCF efficaces conduisent à des OT basés sur des PCG.

Principales Contributions sur Le Calcul Sécurisé

La recherche sur les Générateurs de Corrélations Pseudoaléatoires (PCGs) et les Fonctions de Corrélations Pseudoaléatoires (PCFs) a considérablement amélioré l'utilisation pratique du Calcul Multipartite Sécurisé (MPC). En capitalisant sur les avancées de ces outils, nos contributions se déclinent en trois axes majeurs. Ces contributions sont présentées dans quatre publications [BC23; BCC+24; BCM+24; BBC+24] et sont présentés en détail dans cette thèse, tandis que d'autres contributions réalisées durant mon doctorat sont présentées dans l'ordre chronologique de rédaction dans la Section 1.2.4.

Intersection Privée d'Ensembles (PSI)

Le *Private Set Intersection* (PSI) est un protocole cryptographique essentiel permettant à plusieurs parties de calculer l'intersection de leurs ensembles de données sans divulguer d'informations supplémentaires [PSZ14; PSS+15; KKR+16; RR17; KRT+19; PSW+18; PRT+19; PRT+20; CM20; RS21; GPR+21; RT21a]. Le PSI trouve de nombreuses applications, notamment dans la détection d'attributs partagés entre ensembles de données ou le partage sécurisé de données entre organisations. Les récents progrès des techniques basées sur les générateurs de corrélations pseudoaléatoires (PCG) ont rendu le PSI nettement plus efficace. En exploitant les extensions silencieuses de OT basée sur PCG, les chercheurs ont réduit les coûts de communication à seulement 247 bits par élément de base de données, rendant le PSI évolutif et adapté aux grands ensembles de données [RS21]. Ces optimisations ont établi le PSI comme une composante clé des calculs préservant la confidentialité.

Nous démontrons l'impact significatif des PCGs, tels que le vector-OLE et l'OLE, sur l'amélioration de l'efficacité de l'Intersection de Sets Privés (PSI) dans Chapter 3. Ces résultats ont été publiés à la conférence PKC 2023 [BC23] (en collaboration avec Geoffroy Couteau).

Preuve à divulgation nulle de connaissance (ZKP)

Une preuve à divulgation nulle de connaissance (ZKP) est un protocole cryptographique qui permet à une partie, appelée le prouveur, possédant une information confidentielle (appelée le témoin), de démontrer à une autre partie, le vérifieur, qu'une déclaration publique donnée est vraie, sans révéler d'autres informations sur le témoin en dehors de la véracité de la déclaration. Ce concept est fondamental en cryptographie pour protéger les données et vérifier des déclarations sans exposer d'informations sensibles. Une ZKP satisfait trois propriétés principales :

- Consistance (*completeness*): Si la déclaration est vraie, le prouveur peut convaincre le vérifieur de ce fait.

- Robustesse (*soundness*): Si la déclaration est fausse, un prouveur malhonnête ne pourra pas convaincre le vérifieur.

- Aucun apport d'information (*zero knowledge*): Le vérifieur n'apprend rien au-delà de la véracité de la déclaration.

Dans ce travail, nous nous concentrons sur les ZKP dans un cadre à *vérifieur désigné*, c'est-à-dire que seul un vérifieur disposant d'une clé désignée peut vérifier la déclaration. Nous distinguons deux types de ZKP avec vérifieur désigné :

- *Preuves interactives à divulgation nulle de connaissance (Interactive ZKPs).* Le prouveur et le vérifieur s'engagent dans un protocole interactif, où le vérifieur pose des défis auxquels le prouveur répond. Cette interaction nécessite souvent plusieurs tours.

- *Preuves non interactives à divulgation nulle de connaissance (DV-NIZKs).* Dans une preuve NIZK, le prouveur peut générer la preuve sans interagir avec le vérifieur. Ce type de preuve est particulièrement utile dans les systèmes distribués et décentralisés où l'interaction n'est pas réalisable. Une DV-NIZK *réutilisable* est une version avancée des preuves DV-NIZK qui permet à un vérifieur désigné de vérifier plusieurs preuves du même prouveur sans exiger une nouvelle configuration pour chaque preuve.

Preuves ZKP pour la satisfaisabilité de circuits basées sur des PCG sous-linéaires. En s'appuyant sur la génération efficace et à la volée de VOLE basées sur des PCG, des travaux récents [BMR+21; WYK+21; YSW+21] ont permis des avancées significatives dans les preuves à divulgation nulle de connaissance pour la satisfaisabilité de circuits. Ces ZKP utilisent des schémas d'engagement interactifs pour s'engager indépendamment sur chaque valeur des fils dans le circuit, suivis de vérifications de cohérence avec un surcoût minimal. Bien que la complexité de communication des ZKP générales basées sur VOLE évolue linéairement avec la taille du circuit, elles maintiennent un débit élevé grâce aux opérations sous-jacentes légères. Ces approches mettent en avant l'efficacité des PCGs dans le cadre des applications pratiques de ZKP.

Nous proposons des constructions de Preuves à Divulgation Nulle de Connaissance avec Vérifieur Désigné, y compris des ZKP basés sur des PCGs efficaces pour les Circuits Généraux avec communication sous-linéaire dans Chapter 4 (Section 4.1). Ces résultats sont publiés au Journal of Cryptology 2024 [BCC+24] (co-auteurs Haotian Chu, Geoffroy Couteau, Xiao Wang, Chenkai Weng, Kang Yang et Yu Yu).

Designated-Verifier NIZK à partir de PCFs à Clé Publique. La génération silencieuse de corrélations aléatoires à partir de PCGs ou PCFs nécessite que deux parties participent à un protocole interactif pour générer de manière sécurisée les clés PCG/PCF. Les PCFs à *clés publiques* réduisent cette phase interactive au strict minimum en la remplaçant par une configuration à clés publiques. Plus précisément, après avoir publié leurs clés publiques en ligne, n'importe quelle paire de parties sur un réseau peut commencer à générer des corrélations aléatoires sans *aucune interaction* supplémentaire au-delà de l'infrastructure de clés publiques (PKI). Il est bien établi que les corrélations d'OT sont utiles dans la conception de nombreux primitives ou constructions cryptographiques, y compris les DV-NIZKs [LQR+19; CDI+19]. Ainsi, en supposant l'existence d'une PCF à clés publiques efficace pour l'OT avec une interaction minimale, une question naturelle se pose : peut-on concevoir un schéma DV-NIZK efficace ?

Nous montrons que les PCFs à clé publique peuvent servir de bloc de construction pour concevoir des preuves à divulgation nulle de connaissance non-interactives à vérifieur désigné réutilisables (DV-NIZK) en boîte noire dans Chapter 4 (Section 4.2). Ces résultats sont publiés à la conférence EUROCRYPT 2024 [BCM+24] (co-auteurs Geoffroy Couteau, Pierre Meyer, Alain Passelègue et Mahshid Riahinia).

MPC pour circuit booléen dans le modèle de prétraitement

L'évaluation efficace des circuits booléens est cruciale pour de nombreuses applications cryptographiques. Les circuits booléens représentent des fonctions en format binaire, et leur évaluation sécurisée nécessite des ressources computationnelles importantes. Contrairement au calcul sécurisé des circuits arithmétiques sur de grands corps, la méthode la plus rapide pour exécuter les protocoles MPC à N parties pour les circuits booléens reste la méthode “naïve” de génération de nombreux OT pour chaque pair de partie, avec un coût de $\Omega(m \cdot N^2)$ bits pour m triplets de Beaver. Dans le cadre *à deux parties*, les triplets de Beaver à deux parties peuvent être générés de manière très efficace grâce à une ligne de travaux récents [BCG+18; BCG+19b; BCG+19a] sur l'extension silencieuse d'OT. La situation, en revanche, est beaucoup moins satisfaisante pour le cadre de la computation sécurisée de circuits booléens avec un nombre plus élevé de parties.

Dans le Chapter 5, nous proposons un paradigme, $\mathbb{F}_4$OLEAGE, dans lequel nous introduisons des modèles de prétraitement efficaces pour les circuits booléens, en générant des millions de triplets Beaver par seconde. En exploitant des générateurs programmables de corrélations pseudoaléatoires (PCGs), ces méthodes étendent les avantages de l'OT silencieux aux environnements multipartites, les rendant particulièrement adaptés à des calculs complexes impliquant de nombreux participants.

Ces résultats sont publiés dans les actes de la conférence ASIACRYPT 2024 (co-auteurs Maxime Bombar, Geoffroy Couteau, Alain Couvreur, Clément Ducros et Sacha Servan-Schreiber).

Acknowledgement

As I reach the end of my PhD journey, I find myself reflecting on the many people who have supported, guided, and encouraged me along the way. Writing this thesis has been both one of the most challenging and rewarding experiences of my life, and I truly could not have done it alone. I am deeply grateful to everyone who has been part of this journey—whether through their mentorship, collaboration, or simply their unwavering support.

First and foremost, I want to express my deepest gratitude to Geoffroy for being an incredible advisor. It has truly been an honor to work with you over the past four years. From the very beginning, you have been patient, supportive, and always willing to guide me, helping me grow from a second-year master's intern to where I am today. I've learned so much from you—not just about conducting research properly and developing new ideas, but also about managing multiple projects at once. Your ability to juggle so many responsibilities while staying thoughtful and insightful has always amazed me. The opportunities you provided to work on diverse projects have been invaluable, allowing me to grow both academically and personally. Your patience, guidance, and constant support have meant so much to me. No matter how many questions I had or how stuck I felt, you were always there to listen, offer advice, and help me move forward. I deeply appreciate your mentorship and the trust you placed in me. You always encouraged me to explore topics I was passionate about and to collaborate with people I wanted to work with. That independence, combined with your constant support, helped me gain confidence in my own ideas and research. For all of this and more, thank you. I couldn't have asked for a better mentor, and I'm truly grateful for everything I've learned from you.

From time to time, I like to reflect on and appreciate the people who have given me opportunities and taught me valuable lessons. First and foremost, I want to sincerely thank Hieu for giving me the chance to do a summer internship during my first year of my master's. That opportunity was invaluable, as it was my first step toward learning how to do research. I am also deeply grateful for all the support and guidance throughout my PhD journey. Your encouragement has meant a lot to me, and I truly appreciate everything you have done—not only for me but also for always supporting the Vietnamese community and inspiring us to pursue cryptography. Next, I would like to thank Nigel for allowing me to join the COSIC group in the summer of 2023 when I was just a first-year PhD student. This opportunity allowed me to work with amazing people, helping me learn how to conduct research independently and understand how proactive a PhD student should be. For the summer of 2024, I am incredibly grateful to Masa for welcoming me to NTT and giving me the chance to work together. Thank you for taking the time to have weekly discussions with me during the summer break, for answering all my random questions, and for always being open to exchanging ideas. I deeply admire your sharp thinking, your energy, and your ability to refine and elevate even the simplest ideas. Thanks also for letting me be part of projects that have helped me grow and learn so much. Finally, I would like to express my gratitude to Benny and Carsten for agreeing to be my reviewers and for taking the time to evaluate my manuscript. I also want to thank Elette, Fabien, Hieu, Peter, and Sophie for being part of my jury and for reading my manuscript. Your time and effort are truly appreciated.

My academic journey would not have been complete without the support of my co-authors and collaborators. I would like to express my sincere gratitude to Kelong for being an incredible companion during the summer of 2023 when I was interning at COSIC. Thank you for spending time working with me and for setting an example of how proactive and hardworking a PhD student should be. I learned so much from you, and that summer marked a turning point in my PhD journey. I am also deeply grateful to all my co-authors and everyone I've had the opportunity to collaborate with on various projects. Your insights, support, and contributions have enriched my research experience in countless ways. I also want to extend my thanks to everyone I've had discussions with and met at conferences and seminars. Every conversation has been valuable, and I've gained so much from these exchanges.

I've always considered myself lucky to be surrounded by amazing friends. I want to thank the Crypto team at IRIF, including all the postdocs, PhDs, and both long-term and short-term visitors. Thank you for the weekly beers, the discussions, and all the fun stories we've shared. These moments have provided me with invaluable insights and endless encouragement. A special thanks to Clément and Eliana for being there with me throughout these four years, from the moment we started our PhDs together. I'd also like to thank the IRIF team as well as my officemates, especially Eliana and Maël, for the lunches and the coffees we shared after lunch every day, the seminars we attended together, the discussions we've had, and all the advice—both personal and academic—that I received from everyone. A special thanks to Hang for being my only Vietnamese friend at IRIF. Back in the summer of 2024, I'm grateful to the NTT team, the postdocs and interns, for spending time with me every week over a few months—hanging out and having fun as we explored Japan. Those moments were so much fun and a great way to bond. Finally, I want to thank all my Vietnamese friends I've met throughout my master's, within the Vietnamese crypto community, and at conferences as well as in daily life. Your support and the shared experiences have been a constant source of motivation as I've worked to complete this journey.

Last but not least, I want to express my deep gratitude to my family for always supporting me and allowing me to pursue what I'm passionate about. Thank you for trusting me, for letting me leave home at the age of 15 to follow my dreams, and for giving me the opportunity to come to France to continue my journey.

Contents

Chapter 1

Introduction

Secure Communication refers to the practice of exchanging information between two or more parties in such a way that no unauthorized third party can intercept, understand, or tamper with the communication. The goal is to protect the privacy and integrity of the information being shared. Secure communication is crucial because it underpins a wide range of real-life applications.

- *Online Banking.* When you access your bank account online, secure communication ensures that your login details, transactions, and personal data are kept private.
- *Messaging Apps.* Apps like WhatsApp or Signal use secure communication to ensure that only you and the person you are communicating with can read your messages, preventing others (like adversaries) from spying on your conversations.
- *Secure Websites.* When you visit websites that require sensitive information (like online shopping sites or social media), the use of "https://" in the URL indicates that the communication between your browser and the website is encrypted, protecting your data.

In summary, *secure communication* is all about ensuring that the information you share stays private, accurate, and safe from tampering or eavesdropping, whether you are sending a message, conducting business transactions, or sharing personal data. *Secure communication* is first realized using *encryption. Secure communication* is realized through *encryption.* Encryption involves using a special code (the encryption algorithm) to scramble the contents of a message. Only the person with the correct "key" can unscramble (decrypt) it, ensuring that even if the message is intercepted, it remains unreadable to unauthorized parties.

Modern Cryptography is the study and application of mathematical techniques to ensure the security and integrity of information in the digital age. It is not just about hiding information (*encryption*) but also includes many different security goals and cryptographic tools. Modern cryptography is foundational for securing communication, protecting data, enabling digital trust, and ensuring privacy in an increasingly interconnected world.

The objectives of modern cryptography can be categorized into five key areas:

- Confidentiality: Ensuring that information is accessible only to authorized parties (e.g., symmetric-key and public-key encryption).
- Integrity: Guaranteeing that data remains private during transmission or storage (e.g., hash functions, message authentication codes).

- Authentication: Verifying the identities of parties in a communication (e.g., digital signatures, public-key certificates).
- Non-repudiation: Ensuring that parties cannot deny their involvement in a communication or transaction (e.g., commitment schemes, digital signatures).
- Privacy and Anonymity: Protecting sensitive information and the identities of participants (e.g., zero-knowledge proofs, ring signatures, threshold-ring signatures).

Among the various branches of modern cryptography, cryptographic protocols play a pivotal role. These protocols combine cryptographic primitives—such as encryption, hash functions, and digital signatures—to achieve specific security objectives, including confidentiality, integrity, and authentication, in a structured and reliable manner.

1.1 Secure Multi-Party Computation (MPC)

Multi-party computation (MPC) is a cryptographic technique that enables multiple organizations or parties to work together on calculations on sensitive data, without actually sharing any private information. This is incredibly valuable in privacy-preserving applications because it allows for collaborative analysis while protecting individual data. Here are some specific applications, along with real-world examples:

Healthcare Data Analysis. In healthcare, hospitals often need to analyze data collectively to find patterns or treatments without exposing individual patient records. Using MPC, hospitals can securely compute statistics or train models on patient data without compromising privacy. For instance, the *Enigma platform* [Nat+20] by the Initiative for Cryptocurrencies and Contracts (IC3) allows medical institutions to aggregate data across hospitals securely to study disease trends, ensuring patient information remains confidential.

Detecting Fraud in Banking. Safely sharing suspicious activities of banks, like *JP Morgan*, that are using MPC to collaboratively detect fraud without exposing their customer data [Kul24]. Through MPC, banks can securely compare suspicious accounts without revealing full account information, making it easier to identify common fraud across institutions. Only the necessary information is shared, enabling enhanced fraud detection while keeping customer data private.

Making Personalized Recommendations Without Compromising Privacy. For personalized recommendations (like movie or song suggestions), streaming services could combine their data using MPC without directly accessing each user's preferences. *Google* has explored this with MPC-based private information retrieval [Goo23], allowing platforms to recommend content based on broader trends without directly accessing individual user data. This way, services can make relevant suggestions without needing full access to private data.

Secure Key Management with Distributed Keys. *IBM* leverages MPC in threshold cryptography for secure key management [IBM21]. Sensitive keys are split and stored across multiple servers, allowing cryptographic operations (like signing data) without ever bringing the full key together in one place. This approach reduces the risk of key exposure, making it a useful example of MPC's use in enhancing security.

Anonymous Voting in Elections. MPC can also be applied in secure electronic voting. *Denmark's exploration* of MPC in elections demonstrated how votes could be counted while protecting individual privacy [BCD+09]. Through MPC, voters can submit their choices anonymously, and the overall tally is calculated without revealing who voted for what, thus maintaining both transparency and confidentiality.

These examples show how MPC allows multiple parties to collaborate securely, obtaining useful results from private data without exposing that data. By enabling secure, private collaboration in healthcare, finance, content recommendations, key management, and even voting, MPC is helping organizations to improve user-privacy while still achieving valuable outcomes.

In the next two sections, we survey the basics of multi-party computation (MPC) in cryptography and explore the motivation for executing MPC within the correlated randomness model, as well as the limitations associated with this approach. We then present our contributions, which include demonstrating the concrete efficiency gains in MPC when utilizing correlated randomness and addressing some of the primary limitations inherent in this approach.

1.1.1 Secure Computation in the Correlated Randomness Model

Secure multiparty computation (MPC), which is a subfield of cryptography whose goal is to protect private information, for a public functionality f allows N parties with private inputs $(x_1, \ldots, x_N)$ to securely compute $f(x_1, \ldots, x_N)$, while concealing all other information about their private inputs to coalitions of corrupted parties. MPC protocol was introduced in the seminal work of Yao [Yao86] and later in the work of Goldreich, Micali, and Wigderson (GMW) [GMW87], and has since led to a rich body of work developing the foundations of MPC.

The efficiency of multiparty computation protocols can be optimized via *input-independent preprocessing*. Specifically, once parties can generate many instances of random correlations in advance, later using them in the online phase then the cost of executing MPC protocol is significantly *reduced* in both communication and computation. For example, the GMW framework [GMW87], which is known as secret-sharing-based secure computation. The parties hold shares of the inputs and iteratively compute the circuit representing the function, gate-by-gate. Because addition gates can be computed locally by the parties holding the input shares, only multiplication gates require interaction between the parties to evaluate. As such, the major bottleneck of MPC protocols is due to the communication required to evaluate the multiplication gates in a circuit. However, a core advantage of secret-sharing-based MPC, first identified in the work of Beaver [Bea92], is that secure multiplications can be *preprocessed* in an *input-independent* precomputation phase. In particular, the parties can securely generate additive shares of many "Beaver triples" $(a, b, a \cdot b) \in \mathbb{F}^3$. Then, for each multiplication gate that needs to be computed in the online phase, the parties can run a fast information-theoretically secure multiplication protocol that consumes one Beaver triple and involves communicating just two elements of $\mathbb{F}$ per party. This model of secure computation with preprocessing, due to the efficiency of the online phase, forms the basis for modern MPC protocols. However, this preprocessing paradigm only serves to push the inefficiency bottleneck of MPC to the offline phase that consists of generating many correlated randomnesses such as Beaver triples. Furthermore, although offline communication may be inexpensive, storing substantial amounts of correlated randomness for each potential future interaction can still be costly.

Therefore, a key question for efficient MPC is whether we can generate correlated randomness with sublinear communication compared to the number of correlations and minimal storage requirements.

Generating Pseudorandom Correlations. Recently, a new paradigm has emerged that enables the *silent* generation of long correlated *pseudo*random strings [BCG+18; BCG+19b; BCG+19a], removing essentially all of the communication in the preprocessing phase. Concretely, this is made possible by the new cryptographic primitives, such as *pseudorandom correlation generators* (PCGs) [BCG+19b] and *pseudorandom correlated functions* (PCFs) [BCG+20a].

A PCG compresses long correlations into short, correlated seeds that can later be locally expanded into pseudorandom instances of the target correlation. Slightly more formally, a PCG is a pair of algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$, where $\mathsf{PCG.Gen}$ produces two short keys $(\mathsf{k}_0, \mathsf{k}_1)$, and $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ produces a long string y_σ such that (y_0, y_1) form pseudorandom samples from the target correlation. PCGs enable silent secure computation as follows: using a small distributed protocol to securely generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$, two parties can afterward locally expand them into long correlated pseudorandom strings without any further communication.

PCGs suffer from a major limitation: after distributing the keys, the parties are bound to generate *all at once* a priori *fixed amount* of correlated randomness. PCFs overcome this issue: a PCF is a pair of algorithms $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ where $\mathsf{PCF.Gen}$ produces two short keys $(\mathsf{k}_0, \mathsf{k}_1)$, and $\mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x)$ outputs y_σ^x where for each new input x, (y_0^x, y_1^x) appears like a fresh sample from the target correlation. Hence, after distributively generating the keys $(\mathsf{k}_0, \mathsf{k}_1)$ once and for all, two parties can generate any amount of target correlations on the fly in all their future secure computations.

Useful Correlations for MPC. While generating correlations in preprocessing can boost the efficiency of the online phase in MPC, it is important to discuss the concrete types of two-party correlations crucial for MPC protocols. For example, random Oblivious Transfer (OT) correlation, in which one party is given a pair of random bits (more generally, strings) (s_0, s_1) and the other party is given the pair (b, s_b) for a random bit b. The OT correlation can serve as a basis for general MPC protocols with no honest majority [GMW87; Yao86]. Other kinds of two-party correlations that are useful for arithmetic circuit based-MPC include oblivious linear-function evaluation (OLE) correlations [ADI+17], and multiplication triples (also known as "Beaver triples") [Bea92; DPS+12]. The line of work on PCGs and PCFs has been fairly successful in generating OT, OLE, or Beaver triples:

- For OLE correlation over a finite field $\mathbb{F}$, i.e., a two-party correlation (r_0, r_1) where $r_0 = (\Delta, u) \leftarrow\$ \mathbb{F}^2$ and $r_1 = (v, w) \in \mathbb{F}^2$, $v \leftarrow\$ \mathbb{F}$ and $w := \Delta u + v$. The PCG-based OLE [BCG+22] has concrete efficiency features. The parties can store two 1.25MB seeds, which they expand as needed to generate over a million OLEs (32MB total, 26 times the seed size) in Z_q, where q is a 128-bit secure product of two 62-bit primes. On a single core of a modern laptop, this process takes under 10 seconds, achieving a throughput of more than 100 thousand OLEs per second. To generate authenticated triples from PCG-OLE, the cost of expansion doubles, yielding 50 thousand triples per second and increasing the seed size to 2.6MB.

- Another useful correlation is vector OLE (VOLE), an extension of OLE that can replace OLE with multiple vector OLEs in certain applications [ADI+17]. In a VOLE of length n, one party holds $(\Delta, \mathbf{u})$ and another holds $(\mathbf{v}, \mathbf{w})$ such that $\Delta \leftarrow\$ \mathbb{F}$, $(\mathbf{u}, \mathbf{v}) \leftarrow\$ \mathbb{F}^n$ and $\mathbf{w} := \Delta\mathbf{u} + \mathbf{v} \in \mathbb{F}^n$. Moreover, random OT can be realized by a VOLE correlation if $\mathbb{F} = \mathbb{F}_2$, therefore efficient PCG/PCF-based VOLE leads to that of PCG-based OT. Modern PCG protocols for OT correlation (often called *silent OT extension*) can stretch up to 10 million OTs per second on one core of a standard laptop [CRR21; BCG+22; RRT23] from keys in the 10$\sim$20 KB range, and the fastest PCFs for OT [BCG+22] can generate up to 100 thousand OTs per second on one core of a standard laptop.

1.1.2 Practical Secure Computation

The line of work on Pseudorandom Correlation Generators (PCGs) and Pseudorandom Correlation Functions (PCFs) has significantly advanced the practical application of Multi-Party Computation (MPC). In this section, we discuss two key aspects:

- The practical applications of PCGs and PCFs in enhancing the efficiency of concrete functionalities in MPC, such as Private Set Intersection (PSI) and Zero-Knowledge Proofs (ZK).
- The current limitations within the state-of-the-art PCGs and PCFs, along with the open questions that our work aims to address.

Private Set Intersection (PSI)

PSI is a cryptographic primitive allowing parties to jointly compute the set of all common elements between their datasets, without leaking any value outside the intersection. It is a special case of secure multi-party computation. PSI enjoys a wide array of real-life applications; it is perhaps the most actively researched concrete functionality in secure computation and has been the target of a tremendous number of works, see [PSZ14; PSS+15; KKR+16; RR17; KRT+19; PSW+18; PRT+19; PRT+20; CM20; RS21; GPR+21; RT21a] and references therein for a sample. As a consequence of this intense research effort, recent PSI protocols now achieve impressive efficiency features, communicating only a few hundred bits per database item, and processing millions of items in seconds.

Improving PSI with pseudorandom correlation generators. One of important applications of PCGs is that PCGs allow to construct *silent OT extension* protocols [BCG+19a], which can realize (pseudorandom) OT extension with minimal (logarithmic) communication. Since the top-performing PSI protocols rely on efficient OT extension, using PCG-based techniques to improve their efficiency is a natural idea. And indeed, this was done recently for OKVS-based PSI in [RS21], leading to the most efficient PSI protocol known to date (OKVS stands for oblivious key-value store [GPR+21]; the use of OKVS is the leading paradigm for the design of PSI protocols). To give a single datapoint, computing the intersection between two databases of size $n = 2^{20}$ with the protocol of [RS21] communicates as little as $426n$ bits in total. In addition, some of the tools used in [RS21] have been significantly improved since: replacing their OKVS (which is the PaXoS OKVS of [PRT+20]) by the more recent 3H-GCT OKVS of [GPR+21], and replacing their PCG (which is the one from [WYK+21]) by the recent PCG of [CRR21], the cost goes down to an impressive $247n$ bits of total communication. In comparison, even the *insecure* approach of exchanging the hashes of all items in the databases already requires $160n$ bits of communication. OKVS-based PSI protocols are now firmly established as the leading paradigm in the field, and the use of PCGs to reduce their communication overhead even more seems to further widen the gap with the other paradigms.

Therefore, taking the advantage of PCGs, we aim to explore whether it is possible to design two-party PSI protocols that achieve either improved performance or enhanced security.

Zero-Knowledge Proofs (ZKPs)

A ZKP is a cryptographic protocol that allows one party who possesses a witness, known as the prover, to demonstrate to another party, called the verifier, that a particular public statement is true without revealing any additional information about witness beyond the truth of the statement itself. This concept is critical in cryptography for securing data and verifying statements without exposing sensitive information. A ZKP has three properties:

- *Completeness*: if the statement is true then the prover can convince the verifier of this fact.
- *Soundness*: if the statement is false, a cheating prover will fail to convince the verifier.
- *Zero-Knowledge*: the verifier learns nothing beyond the fact that the statement is true.

In this work, we focus on ZKPs in the *designated-verifier* setting. We classify two types of designated-verifier ZKPs: Interactive Zero-Knowledge Proofs and Non-Interactive Zero-Knowledge Proofs (DV-NIZKs).

Interactive Zero-Knowledge Proof: The prover and verifier engage in an interactive protocol, where the verifier poses challenges, and the prover responds. This interaction often involves several rounds. Specifically, in this work, we work on interactive ZKPs for circuit satisfiability where the verification of a statement is represented as a public circuit $C : \{0,1\}^n \to \{0,1\}$. The commit-and-prove zero-knowledge (CP-ZK) paradigm is among the most flexible and modular design mechanisms for constructing ZKP. In commit-and-prove zero-knowledge protocols, the prover begins by committing to a value related to the secret without revealing it. Later, the prover can "open" this commitment to reveal specific aspects of the committed information or demonstrate properties of the secret, proving its validity while still keeping the sensitive information concealed.

Non-Interactive Zero-Knowledge Proofs (NIZKs): The proof can be generated by the prover without interacting with the verifier. This kind of proof is particularly useful in distributed and decentralized systems, where interaction isn't feasible. The main difference between DV-NIZKs and standard NIZKs lies in the verification process. Standard NIZKs are publicly verifiable, meaning anyone with the proof can verify it. In contrast, DV-NIZKs restrict verification to a single designated verifier, preventing the proof from being reused or validated by anyone else. A *reusable* DV-NIZK is a more advanced type of designated-verifier NIZK that allows a designated verifier to verify multiple proofs from the same prover without requiring a new setup for each individual proof.

Sublinear PCG-based Zero-Knowledge Proof for Circuit Satisfiability. By leveraging efficient on-the-fly generation of PCG-based VOLE, recent VOLE-based zero-knowledge proofs for circuit satisfiability [BMR+21; WYK+21; YSW+21] have achieved notable efficiency. These ZKPs use interactive commitment schemes that independently commit to each wire value in the circuit, followed by consistency checks with minimal overhead. Although the communication complexity of general VOLE-based ZKPs scales linearly with the circuit size, they maintain high throughput due to the underlying lightweight operations.

From SIMD-ZK to general ZK. Define $(B, \mathcal{C})$-SIMD circuit (Single Instruction Multiple Data) which contains B identical components of the circuit $\mathcal{C}$. A SIMD-ZK proves that for input witnesses $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_B)$, $\mathcal{C}(\boldsymbol{w}_i) = 0$ for $i \in [B]$. By exploiting the fact that operations are identical across B components, SIMD-ZK schemes typically utilize vector commitments and batch proofs to achieve communication sublinear in $B \cdot |\mathcal{C}|$. In more detail, denote by $[\![\boldsymbol{w}]\!]$ a commitment to a vector $\boldsymbol{w}$. Define a witness matrix $\boldsymbol{W} = (\boldsymbol{w}_1 \| \ldots \| \boldsymbol{w}_B)$. Instead of viewing the ith column as the witness to the ith evaluation of $\mathcal{C}$, a prover commits to each row vector and lets the verifier obtain $([\![\boldsymbol{w}^1]\!], \ldots, [\![\boldsymbol{w}^{|\mathcal{C}|}]\!])$. In this way, for any gate $(\alpha, \beta, \gamma, \diamond)$ in $\mathcal{C}$ and $\diamond \in \{\mathsf{Add}, \mathsf{Mult}\}$, the prover only needs to prove that $\boldsymbol{w}^\gamma = \boldsymbol{w}^\alpha \diamond \boldsymbol{w}^\beta$. ZKP schemes achieve $\mathcal{O}(|\mathcal{C}|)$ proof size if both the vector commitment and batch proof of additions and multiplications incur constant size. Following the line of work of VOLE-based ZKPs, AntMan [WYY+22] designs a ZKP for $(B, \mathcal{C})$-SIMD circuits with a complexity of $\mathcal{O}(B + C)$ and they then conduct the transformation from SIMD-ZK to general ZK by using a wire consistency check on top of SIMD-ZK. In particular, AntMan first arranges all gates in batches, commits to their input and output wire values, and then utilizes a SIMD-ZK to prove that all batches of gates are computed correctly. Then an extra protocol is invoked to prove the consistency of each wire value that is repeatedly packed in multiple commitments, e.g., for batched wire values $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{F}^B$ and wire indices $i, j \in [B]$, it aims to check whether they satisfy $\boldsymbol{w}_1[i] = \boldsymbol{w}_2[j]$. AntMan requires $\mathcal{O}(B^3)$ complexity for checking all combinations of $(i, j) \in [B] \times [B]$, which leads to a total communication complexity of $\mathcal{O}(B^3 + C/B)$. This translates to a $\mathcal{O}(C^{3/4})$ cost when setting $B = C^{1/4}$.

An interesting question is whether we can design a generic a compiler that translates any commit-and-prove SIMD-ZK (CP-SIMD-ZK) into a general CP-ZK with exactly *sublinear communication complexity.*

Designated-Verifier NIZK from Public-Key PCFs. DV-NIZKs are believed to be easier to obtain than standard NIZKs, in the following sense: they are known to exist under the plain CDH assumption in pairing-free groups [CH19; QRW19; KNY+19], while NIZKs are only known in pairing groups, or using subexponential hardness assumptions [JJ21; CJJ+23]. Yet, efficiency-wise, we do not know of any concretely efficient construction of DV-NIZKs in pairing-free groups (efficient NIZKs are known in pairing groups [GS08; KW15; CH20], and known DV-NIZKs in pairing-free groups rely on the hidden bit model, for which no concretely efficient instantiation is known).

The silent generation of correlated randomness from PCGs or PCFs requires two parties to engage in an interactive protocol to securely generate the PCG/PCF keys. *Public-Key* PCFs reduce this interactive phase to a bare minimum, by replacing it with a public-key setup. More precisely, after publishing their public keys online, any pair of parties on a network can start generating correlated randomness, without *any interaction* beyond the initial PKI (Public Key Infrastructure). From PCGs/PCFs for OT, VOLE and OLE correlations, one can obtain NIZKs [BCG+18] or preprocessing NIZKs, in which the trusted party generates a verification key for the verifier and additionally a secret proving key for the prover [BCG+19b; BCG+20a]. Therefore, assume there exists an efficient public-key PCF for OT with a minimal interaction, it is natural to design an efficient DV-NIZK scheme with stronger security.

MPC For Boolean Circuit in the Preprocessing Model

In contrast to the secure computation of arithmetic circuits over large fields, the fastest way to run N-party MPC protocols for Boolean circuits remains the "naïve" method of generating many pairwise OTs, at a cost of $\Omega(m \cdot N^2)$ bits for m Beaver triples. This is in contrast to the *two-party* setting, where two-party Beaver triples can be generated very efficiently thanks to a recent line of work [BCG+18; BCG+19b; BCG+19a] on silent OT extension. In silent OT extension, two parties can generate m Beaver triples using only $O(\log m)$ communication. The state-of-the-art protocols in this area [CRR21; BCG+22; RRT23] achieve impressive throughputs of several million Beaver triples per second on one core of a standard laptop. Furthermore, the recent SoftSpoken OT extension protocol [Roy22] yields even faster OTs at the cost of increasing communication. For example, SoftSpoken can generate nearly 30M OT/s on *local network* at the cost of increasing the communication to $64m$ bits to generate m Beaver triples; other communication/computation tradeoffs are possible [Roy22, Table 1].[1]

The situation, however, is much less satisfying for the setting of secure computation of Boolean circuits with a larger number of parties. Protocols such as SPDZ [DPS+12] and Overdrive [KPR18] do not perform well when generating Beaver triples for Boolean circuits, even in the passive setting. This is due to the high overhead of embedding $\mathbb{F}_2$ in an extension field compatible with the number theoretic-transform used in efficient instantiations of the BGV encryption scheme [BGV14]. Furthermore, silent OT extension techniques build on Pseudorandom Correlation Generators (PCGs),

[1]Note that we need two calls to the OT functionality to generate one Beaver triple.

which typically work only in the two-party setting [BCG+19b]. To handle more parties, one needs the stronger notion of *programmable* PCG [BCG+20b], which, informally, allows partially specifying parts of the generated correlation. Unfortunately, while efficient programmable PCGs over large fields were introduced in [BCG+20b], building *concretely efficient*, programmable PCGs over $\mathbb{F}_2$ has remained elusive thus far, making N-party PCGs for $\mathbb{F}_2$ primarily of theoretical interest. The state-of-the-art is the recent work of Bombar et al. [BCC+23], which generates Beaver triples over any field $\mathbb{F}_q$ with $q \geq 3$. However, Bombar et al. [BCC+23] leave analyzing the concrete efficiency for future work.

1.2 Our Contribution

Building on the advancements in Pseudorandom Correlation Generators (PCGs) and Pseudorandom Correlation Functions (PCFs), our contributions fall into three main categories:

- We demonstrate a significant impact of PCGs, such as vector-OLE and OLE, on enhancing the efficiency of Private Set Intersection (PSI).
- We propose constructions of Designated-Verifier Zero-Knowledge Proof including efficient PCG-based ZKP for General Circuits with sublinear communication and Reusable DV-NIZK from Public-Key PCF-based OT.
- We address key open questions regarding PCGs/PCFs, specifically achieving PCGs for OLE over $\mathbb{F}_4$, enabling efficient MPC for Boolean circuits.

These contributions are presented in four publications [BC23; BCC+24; BCM+24; BBC+24], with the core work highlighted in this thesis as in the following three sections, while other contributions made during my PhD are presented in chronological order of writing in Section 1.2.4.

1.2.1 Improving PSI for Set with Small Entries [BC23]

We introduce new protocols for private set intersection (PSI), building upon recent constructions of PCG as vector OLE. Our new constructions improve over the state of the art on several aspects and perform especially well in the setting where the parties have databases with small entries. The two main contributions are as follows:

1. We introduce a new semi-honest PSI protocol that combines subfield vector OLE with hash-based PSI, i.e., it can be instantiated using several hashing techniques. Our protocol is the first PSI protocol to achieve communication complexity *independent* of the computational security parameter κ, and has communication lower than all previous known protocols for input sizes ℓ below 70 bits.

2. We enhance the security of our protocol to the malicious setting, using two different approaches. In particular, we show that applying the *dual execution technique* yields a malicious PSI whose communication remains independent of κ, and improves over all known PSI protocols for small values of ℓ.

As most previous protocols, our above protocols are in the random oracle model. We introduce a third protocol which relies on subfield ring-OLE to achieve maliciously secure PSI in the *standard model*, under the ring-LPN assumption. Our protocol enjoys extremely low communication, reasonable computation, and standard model security. Furthermore, it is *batchable*: the message of a client can be reused to compute the intersection of their set with that of multiple servers, yielding further reduction in the overall amortized communication.

These results appear in the proceedings of PKC 2023 (co-authored with Geoffroy Couteau).

1.2.2 Efficient Designated-Verifier Zero-Knowledge Proof [BCC+24; BCM+24]

Sublinear PCG-based ZKP for General Circuits [BCC+24]. We propose a generic compiler that can convert any zero-knowledge (ZK) proof for SIMD circuits to general circuits efficiently with an extension that can preserve the space complexity of the proof systems. Our compiler can immediately produce new results improving upon state-of-the-art. By plugging Antman [WYY+22] in our compiler, an interactive sublinear-communication VOLE-based ZK protocol, we improve the overall communication complexity for general circuits from $\mathcal{O}(C^{3/4})$ to $\mathcal{O}(C^{1/2})$. Our implementation shows that for a circuit of size 2^{27}, it achieves up to $83.6\times$ improvement in communication compared to the state-of-the-art implementation. Its end-to-end running time is at least 70% faster in a 10Mbps network.

These results appear in the proceedings of the Journal of Cryptology 2024 (co-authored with Haotian Chu, Geoffroy Couteau, Xiao Wang, Chenkai Weng, Kang Yang and Yu Yu).

Reusable DV-NIZK from Public-Key PCF-based OT [BCM+24]. We show that public-key PCFs can serve as a building block to construct reusable designated-verifier non-interactive zero-knowledge proofs (DV-NIZK) for NP in a black-box manner. In particular, assuming there exists an efficient public-key PCF for OT, one can compile any Σ-protocol with binary challenge into a DV-NIZK. we obtain a new DV-NIZK from polynomial assumptions over pairing-free groups for all languages that admit a Σ-protocol with bit challenges, with communication comparable to that of the Σ-protocol. Conceptually, our result can be seen as observing that a public-key PCF suffices to upgrade *non-reusable* DV-NIZKs (which exist from public key encryption [CHH+07]) into *reusable* DV-NIZKs.

These results appear in the proceedings of EUROCRYPT 2024 (co-authored with Geoffroy Couteau, Pierre Meyer, Alain Passelègue and Mahshid Riahinia).

1.2.3 FOLEAGE: $\mathbb{F}_4$OLEAGE-based MPC for Boolean Circuits [BBC+24]

As mentioned in Section 1.1.2, the MPC of boolean circuits is less satisfying than the MPC of arithmetic circuit. We introduce $\mathbb{F}_4$OLEAGE, which addresses this gap by introducing an efficient preprocessing protocol tailored to Boolean circuits, with semi-honest security and tolerating $N-1$ corruptions. $\mathbb{F}_4$OLEAGE has excellent concrete performance: It generates m multiplication triples over $\mathbb{F}_2$ using only $N \cdot m + O(N^2 \cdot \log m)$ bits of communication for N-parties, and can concretely produce over 12 million triples per second in the 2-party setting on one core of a commodity machine. Our result builds upon an efficient PCG for multiplication triples over the field $\mathbb{F}_4$.

This is achieved by introducing a number of protocol-level, algorithmic-level, and implementation-level optimizations on the recent PCG construction of Bombar et al. (CRYPTO 2023) from the Quasi-Abelian Syndrome Decoding assumption.

Specifically, we focus on secure computation of general Boolean circuits with multiple parties in the semi-honest setting. $\mathbb{F}_4$OLEAGE outperforms the state-of-the-art approach in both the *two-party* and *multi-party* setting. In particular, $\mathbb{F}_4$OLEAGE enjoys much lower communication in the preprocessing phase than all known alternatives and has a very low computational overhead. We expect $\mathbb{F}_4$OLEAGE to be the fastest alternative for large enough circuits on almost any realistic network setting, for any number of parties between two and several hundred.

These results appear in the proceedings of ASIACRYPT 2024 (co-authored with Maxime Bombar, Geoffroy Couteau, Alain Couvreur, Clément Ducros and Sacha Servan-Schreiber).

1.2.4 Other Contributions

Improved All-but-One Vector Commitment with Applications to Post-Quantum Signatures [BCS24]

Post-quantum digital signature schemes have recently received increased attention due to the NIST standardization project for additional signatures. MPC-in-the-Head and VOLE-in-the-Head are general techniques for constructing such signatures from zero-knowledge proof systems. A common theme between the two is an *all-but-one* vector commitment scheme which internally uses GGM trees. This primitive is responsible for a significant part of the computational time during signing and verification.

A more efficient technique for constructing GGM trees is the half-tree technique, introduced by Guo et al. (EUROCRYPT 2023). Our work builds an all-but-one vector commitment scheme from the half-tree technique, and further generalizes it to an all-but-τ vector commitment scheme. Crucially, our work avoids the use of the random oracle assumption in an important step, which means our binding proof is non-trivial and instead relies on the random permutation oracle. Since this oracle can be instantiated using fixed-key AES which has hardware support, we achieve faster signing and verification times. We integrate our vector commitment scheme into FAEST (faest.info), a round one candidate in the NIST standardization process, and demonstrates its performance with a prototype implementation. For $\lambda = 128$, our experimental results show a nearly 3.5-fold improvement in signing and verification times.

These results are detailed in a manuscript that has not yet been published (co-authored with Kelong Cong and Cyprien Delpech de Saint Guilhem).

Faster Signatures from MPC-in-the-Head [BCC+25]

We revisit the construction of signature schemes using the MPC-in-the-Head paradigm and obtain two main contributions:

1. We observe that previous signatures in the MPC-in-the-Head paradigm must rely on a salted version of the GGM puncturable pseudorandom function (PPRF) to avoid collision attacks. We design a new efficient PPRF construction that is provably secure in the multi-instance setting. The security analysis of our PPRF, in the ideal cipher model, is quite involved and forms a core technical contribution to our work. While previous constructions had to rely on a hash function, our construction uses only a fixed-key block cipher and is considerably more efficient as a result: we observe a $12\times$ to $55\times$ speed improvement for a recent signature scheme (Joux and Huth, CRYPTO'24). Our improved PPRF can be used to speed up many MPC-in-the-head signatures.

2. We introduce a new signature scheme from the regular syndrome decoding assumption, based on a new protocol for the MPC-in-the-head paradigm, which significantly reduces communication compared to previous works. Our scheme is conceptually simple, though its security analysis requires a delicate and nontrivial combinatorial analysis.

These results appear in the proceedings of ASIACRYPT 2024 (co-authored with Eliana Carozza, Geoffroy Couteau, Dahmun Goudarzi and Antoine Joux).

Efficient Multi-instance Vector Commitment and Application to Post-quantum Signatures [Bui25]

The MPC-in-the-Head (MPCitH) and the VOLE-in-the-Head (VOLEitH) paradigms have recently been utilized to develop post-quantum signatures. Both rely on a mechanism that allows the signer to commit to N values and then later open all-but-one. In particular, MPCitH-based signatures achieve this using a puncturable pseudorandom function (PPRF) primitive, while VOLEitH-based signatures utilize an all-but-one vector commitment scheme.

A novel and efficient multi-instance PPRF, introduced by Bui *et al.* (ASIACRYPT'24), provides a significant performance boost for MPCitH-based signatures, employing only a fixed-key block cipher to instantiate the PPRF while being provably secure in the ideal cipher model. This work presents an efficient multi-instance vector commitment derived from multi-instance PPRF. Our vector commitment scheme is secure in the multi-instance setting, when handling repetitive parallel executions. As a result, it can be directly applied to enhance the efficiency of VOLEitH-based signatures.

We implemented our vector commitment scheme into FAEST (faest.info), a round one candidate in the NIST post-quantum cryptography standardization. According to our experimental implementation, we achieve $10\% - 27\%$ improvement in both signing and verification times for various settings.

These results appear in the proceedings of ACISP 2025.

Structured-Seed Local Pseudorandom Generators and their Applications [BCM24]

In this work, we revisit the applications of local PRGs. Our main observation is that many of the standard applications of local PRGs do not require the full power of local PRGs. In particular, many applications only require the existence of a local pseudorandom mapping from n-bit seeds to m-bit strings, but *do not require the seeds to be sampled uniformly at random.* We formalize this observation by introducing the notion of *structured-seed* local pseudorandom generators, which generalize local PRGs to the setting where the seed should be sampled from a prescribed distribution with support over $\{0,1\}^n$ (instead of being sampled uniformly at random), and provide a sample of applications where structured-seed local PRGs can be used as a drop-in replacement to standard local PRGs. Concretely, we show how to use structured-seed local PRGs in the following applications:

- Indistinguishability obfuscation from well-founded assumptions [JLS21].
- Constant-overhead secure computation [IKO+08].
- Compact homomorphic secret sharing [BCM23].
- Hardness of learning DNFs [DV21].

These results are detailed in a manuscript that has not yet been published (co-authored with Geoffroy Couteau and Nikolas Melissaris).

Critical Round in Multi-Round Proofs: Compositions and Transformation to Trapdoor Commitments[ABB+24]

In many multi-round public-coin interactive proof systems, challenges in different rounds serve different roles, but a formulation that actively utilizes this aspect has not been studied extensively. In this paper, we propose new notions called critical-round special honest verifier zero-knowledge and critical-round special soundness. Our notions are simple, intuitive, easy to apply, and capture several practical multi-round proof protocols including, but not limited to, those from the MPC-in-the-Head paradigm.

We demonstrate the usefulness of these notions with two fundamental applications where three-round protocols are known to be useful, but multi-round ones generally fail. First, we show that critical-round proofs yield trapdoor commitment schemes. This result also enables the instantiation of post-quantum secure adaptor signatures and threshold ring signatures from MPCitH, resolving open questions in (Haque and Scafuro, PKC 2020) and in (Liu et al., ASIACRYPT 2024). Second, we show that critical-round proofs can be securely composed using the Cramer-Schoenmakers-Damgård method. This solves an open question posed by Abe et al. in CRYPTO 2024.

Overall, these results shed new light on the potential of multi-round proofs in both theoretical and practical cryptographic protocol design.

These results are detailed in a manuscript that has not yet been published (co-authored with Masayuki Abe, David Balbás, Miyako Ohkubo, Zehua Shang and Mehdi Tibouchi).

Personal Publications

[ABB+24] Masayuki Abe, David Balbás, Dung Bui, Miyako Ohkubo, Zehua Shang, and Mehdi Tibouchi. *Critical Round in Multi-Round Proofs: Compositions and Transformation to Trapdoor Commitments*. ePrint Archive. Available at https://eprint.iacr.org/2024/252. 2024.

[BBC+24] Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, Clément Ducros, and Sacha Servan-Schreiber. "FOLEAGE: $\mathbb{F}_4$OLE-Based Multi-party Computation for Boolean Circuits". In: *Advances in Cryptology – ASIACRYPT 2024*. Ed. by Kai-Min Chung and Yu Sasaki. Singapore: Springer Nature Singapore, 2024, pp. 69–101. ISBN: 978-981-96-0938-3.

[Bui25] Dung Bui. "Efficient Multi-instance Vector Commitment and Application to Post-quantum Signatures". In: *Information Security and Privacy – ACISP 2025*. Springer Nature Singapore, 2025. URL: https://eprint.iacr.org/2024/254.

[BCC+25] Dung Bui, Eliana Carozza, Geoffroy Couteau, Dahmun Goudarzi, and Antoine Joux. "Faster Signatures from MPC-in-the-Head". In: *Advances in Cryptology – ASIACRYPT 2024*. Ed. by Kai-Min Chung and Yu Sasaki. Singapore: Springer Nature Singapore, 2025, pp. 396–428. ISBN: 978-981-96-0875-1.

[BCC+24] Dung Bui, Haotian Chu, Geoffroy Couteau, Xiao Wang, Chenkai Weng, Kang Yang, and Yu Yu. "An Efficient ZK Compiler from SIMD Circuits to General Circuits". In: *Journal of Cryptology* 38.1 (Dec. 2024), p. 10. ISSN: 1432-1378. DOI: 10.1007/s00145-024-09531-4. URL: https://doi.org/10.1007/s00145-024-09531-4.

[BCS24] Dung Bui, Kelong Cong, and Cyprien Delpech de Saint Guilhem. *Improved All-but-One Vector Commitment with Applications to Post-Quantum Signatures*. ePrint Archive. Available at https://eprint.iacr.org/2024/255. 2024.

[BC23] Dung Bui and Geoffroy Couteau. "Improved Private Set Intersection for Sets with Small Entries". In: *PKC 2023, Part II*. Ed. by Alexandra Boldyreva and Vladimir Kolesnikov. Vol. 13941. LNCS. Springer, Heidelberg, May 2023, pp. 190–220. DOI: 10.1007/978-3-031-31371-4_7.

[BCM24] Dung Bui, Geoffroy Couteau, and Nikolas Melissaris. *Structured-Seed Local Pseudorandom Generators and their Applications*. ePrint Archive. Available at https://eprint.iacr.org/2024/253. 2024.

[BCM+24] Dung Bui, Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. "Fast Public-Key Silent OT and More from Constrained Naor-Reingold". In: *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*. Ed. by Marc Joye and Gregor Leander. Vol. 14656. Lecture Notes in Computer Science. Springer, 2024, pp. 88–118. DOI: 10.1007/978-3-031-58751-1_4.

1.3 Organization of this Thesis

The thesis is organized as follows:

- In Chapter 1, we introduce the concepts of MPC and correlated randomness, along with their applications. We then provide an overview of our contributions, which include the four main contributions detailed throughout the thesis, as well as additional work completed by the author during her PhD.
- In Chapter 2, we present the necessary preliminaries for understanding the content of this thesis.
- In Chapter 3, we describe our first contribution, which involves Private Set Intersection (PSI)-based PCG.
- In Chapter 4, we present two additional contributions related to Zero-Knowledge Proofs (ZKPs): sublinear privately verifiable ZKPs based on PCG Section 4.1 and a new DV-NIZK construction based on PCF Section 4.2.
- In Chapter 5, we introduce our final contribution on MPC for Boolean circuits, where we propose a framework to efficiently construct PCG over small fields.
- In Chapter 6, we conclude the thesis and discuss some open questions for future research.

For each chapter presenting a contribution, we begin with the motivation and related work, followed by any additional notations or preliminary concepts needed for that chapter. If no specific preliminaries are required, all necessary background material can be found in Chapter 2. We then present a detailed description of our contribution, along with a technical overview. Finally, the remaining sections provide supplementary information to fully understand the contribution.

Chapter 2

Preliminaries

In this chapter, we establish the foundational concepts and terminology used throughout this manuscript. We begin by introducing the notations and cryptographic definitions that are used in our work. Next, we formalize the computational hardness assumptions critical to our contributions. We then define key primitives related to correlated randomness, including (constrained) pseudorandom function (PRF), pseudorandom correlation generators (PCGs), pseudorandom correlation functions (PCFs), and function secret sharing (FSS). Additionally, we provide definitions for designated-verifier zero-knowledge proofs, both interactive and non-interactive. Finally, we then describe several essential ideal functionalities that are realized by our constructions.

Contents

2.1 Notations

Throughout the manuscript we use the following notations. The concrete notations used in each chapter are listed inside each chapter.

Set, Integers. We let λ, κ denote the computational and statistical security parameters, respectively. We write $[1, m]$ to denote a set $\{1, 2, \dots, m\}$, for simplicity, we sometimes use $[m]$. We use $\mathbb{N}, \mathbb{Z}$ for notations of natural, integer numbers. For the domain of bit string of length ℓ, we denote it as $\{0, 1\}^\ell$.

Negligible Function. A function $f(\lambda)$ is negligible if for every polynomial $\mathsf{poly}(\lambda)$, there exists a λ_0 such that for all $\lambda > \lambda_0$, $f(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$, meaning it decreases faster than any inverse polynomial in λ.

Vector, Matrix. We use bold lowercase for vector and bold uppercase for matrix. For a vector $\mathbf{x}$ we define by x_i its i-th coordinate. For two vector $\mathbf{u} = (u_1, \dots, u_t), \mathbf{v} = (v_1, \dots, v_t) \in R^t$ for some ring R, their tensor product $\mathbf{u} \otimes \mathbf{v}$ is defined by $\mathbf{u} \otimes \mathbf{v} = (u_i \cdot v_j)_{i,j \le t} = (v_1 \cdot \mathbf{u}, \dots, v_t \cdot \mathbf{u})$ and we denote by $\langle \mathbf{u}, \mathbf{v} \rangle$ their inner product. Similarly, we write $\mathbf{u} \boxplus \mathbf{v}$ to denote the outer sum of a vector, equal to $\mathbf{u} \boxplus \mathbf{v} = (u_i + v_j)_{i,j \le t} = (v_1 + \mathbf{u}, \dots, v_t + \mathbf{u})$. We let $\mathbf{u}[i]$ denote the value of index i in $\mathbf{u}$.

Polynomial-Time Algorithm. By an *efficient* algorithm $\mathcal{A}$ or PPT $\mathcal{A}$ we mean that Adv is modeled by a (possibly non-uniform) Turing Machine that runs in probabilistic polynomial time. An *expected* PPT algorithm is a probabilistic algorithm whose expected running time, averaged over its internal randomness, is bounded by a polynomial in the size of its input.

Assignation. For a finite set S, we write $x \leftarrow\$ S$ to denote that x is sampled uniformly at random from S. For an algorithm $\mathcal{A}$, we denote by $y \leftarrow \mathcal{A}(x)$ the output y after running $\mathcal{A}$ on input x.

Probabilities. We denote $\Pr[X = x]$ as the probability of a random variable X taking value x, and $\Pr_{x \in \mathcal{D}}[f(x) = y]$ to denote the probability that $f(x)$ is equal to some fixed value y, when x is sampled from the distribution $\mathcal{D}$.

Distribution. Two distributions D_0 and D_1 (or two families of distributions $\{D_0^\lambda\}_{\lambda \in \mathbb{N}}$ and $\{D_1^\lambda\}_{\lambda \in \mathbb{N}}$) are said to be *computationally indistinguishable* with respect to a security parameter λ if, for all probabilistic polynomial-time (PPT) algorithms $\mathcal{A}$ (called *distinguishers*), the difference in the probabilities of $\mathcal{A}$ outputting 1 on inputs sampled from D_0^λ and D_1^λ is negligible in λ.
Formally, two distributions D_0^λ and D_1^λ are computationally indistinguishable if:

$$\forall \text{PPT } \mathcal{A}, \quad \left| \Pr[\mathcal{A}(x) = 1 \mid x \sim D_0^\lambda] - \Pr[\mathcal{A}(x) = 1 \mid x \sim D_1^\lambda] \right| \le \text{negl}(\lambda),$$

We write $D_0 \approx D_1$ to mean that two distributions D_0 and D_1 are *computationally* indistinguishable to all efficient distinguishers $\mathcal{A}$.

Statistical indistinguishability is a stronger notion than computational indistinguishability. The indistinguishability here is based on their statistical (or total variation) distance.
Formally, two distributions D_0^λ and D_1^λ are statistically indistinguishable if their statistical distance is negligible in the security parameter λ. Formally:

$$\Delta(D_0^\lambda, D_1^\lambda) = \frac{1}{2}\sum_x \left|\Pr[x \in D_0^\lambda] - \Pr[x \in D_1^\lambda]\right| \leq \text{negl}(\lambda),$$

where $\Delta(D_0^\lambda, D_1^\lambda)$ is the statistical (or total variation) distance. We denote $D_0 \approx_s D_1$ to mean that D_0 and D_1 are *statistically* indistinguishable.

2.2 Cryptographic Definitions

2.2.1 Universal Composability (UC)

The Universal Composability (UC) paradigm [Can01] is a formal framework for defining and analyzing the security of cryptographic protocols in a way that ensures they remain secure even when composed with other protocols. In UC security, the security of a protocol π is defined by distinguishing between its real-world execution and an ideal-world execution of a functionality $\mathcal{F}$. The protocol is secure if, for every real-world adversary $\mathcal{A}$, there exists an ideal-world simulator $\mathcal{S}$ such that the outputs in both worlds are computationally indistinguishable from an environment $\mathcal{Z}$.

Consider a protocol π having n parties $(P_1, \ldots, P_n)$ designed to realize an ideal functionality $\mathcal{F}$. The security definition requires that for every real-world adversary $\mathcal{A}$, there exists an ideal-world simulator $\mathcal{S}$ such that:

1. Real World Execution: parties $P_1, \ldots, P_n$ execute protocol π under the corrupted adversary $\mathcal{A}$. This execution results in an output distribution, denoted by $\text{Real}_{\pi,\mathcal{A},\mathcal{Z}}(1^\lambda)$, where λ is the security parameter, and $\mathcal{Z}$ is the environment that provides inputs and observes outputs of the protocol execution.

2. Ideal World Execution: the same set of parties interact with an ideal functionality $\mathcal{F}$ and a simulator $\mathcal{S}$ that emulates the adversary's behavior. This produces an output distribution $\text{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda)$.

Definition 2.2.1 (UC Security)**.** *The protocol π securely realizes $\mathcal{F}$ if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that:*

$$\mathit{Real}_{\pi,\mathcal{A},\mathcal{Z}}(1^\lambda) \approx \mathit{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda)$$

where "$\approx$" denotes computational indistinguishability. In other words, no environment $\mathcal{Z}$ can distinguish between interactions with the real protocol π and the ideal functionality $\mathcal{F}$, formally:

$$\left|\Pr\left[\mathit{Real}_{\pi,\mathcal{A},\mathcal{Z}}(1^\lambda) = 1\right] - \Pr\left[\mathit{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

The simulation-based security is a specific case of UC security, the UC paradigm builds on simulation-based security, but it strengthens the model by requiring security under composition.

2.2.2 Commitment Schemes

A commitment scheme is a cryptographic protocol between a prover and a verifier, where the prover can securely commit to a message by sending a *commitment*. Later, the prover can reveal the original message by "opening" the commitment. This commitment has two key properties: *hiding* and *binding*. The hiding property ensures that the verifier learns nothing about the message from the commitment alone. The binding property guarantees that the prover cannot open a commitment to two different messages. A commitment scheme consists of three probabilistic PPT algorithms $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ for a message space $\mathcal{M}$:

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$. On input a security parameter λ, the algorithm outputs public parameters pp.
- $c \leftarrow \mathsf{Com}(\mathsf{pp}, m, r)$. On input the public parameters pp, a message $m \in \mathcal{M}$, and randomness r from a randomness space, the algorithm outputs a commitment c.
- $\{0, 1\} \leftarrow \mathsf{Open}(\mathsf{pp}, c, m, r)$. On input the public parameters pp, a commitment c, the original message m, and randomness r, the algorithm verifies whether c was correctly formed. It outputs 1 (*accept*) if valid or 0 (*reject*) otherwise.

A commitment scheme must satisfy the following properties:

- **Hiding:** A commitment c does not reveal any information about the committed message $m \in \mathcal{M}$.
$$\forall m_1, m_2 \in \mathcal{M}, \quad \mathsf{Com}(\mathsf{pp}, m_1, r_1) \approx \mathsf{Com}(\mathsf{pp}, m_2, r_2),$$
where r_1, r_2 are chosen uniformly at random, and $\approx$ denotes computational indistinguishability. In perfectly hiding schemes, the distributions are statistically indistinguishable.

- **Binding:** Once a commitment c is made, it is infeasible for the sender to open c to two different messages $m_1, m_2 \in \mathcal{M},\ m_1 \neq m_2$.
$$\Pr\left[\mathsf{Com}(\mathsf{pp}, m_1, r_1) = \mathsf{Com}(\mathsf{pp}, m_2, r_2) \text{ for } m_1 \neq m_2\right] \leq \text{negl}(\lambda),$$
where the probability is over the randomness r_1, r_2.

One of the well-known instantiation of commitments is the Pedersen commitment scheme [Ped92]. Pedersen commitment is perfectly hiding, and computationally binding assuming the hardness of DLog (Section 2.3.1) over $\mathbb{Z}_p$. We formalize the scheme as below.

Pedersen Commitment Scheme. Given a cyclic group G of prime order q with generator g. The Pedersen commitment scheme works over the group G with the message space and the randomness space $\mathbb{Z}_q$ as follows:

- $\mathsf{Pedersen.Setup}(1^\lambda)$. sample additional generator of G such that $h = g^a$ for some unknown $a \in \mathbb{Z}_q$ (ensuring g and h are independent under the Dlog assumption). Output $\mathsf{pp} = (G, q, g, h)$.
- $\mathsf{Pedersen.Com}(\mathsf{pp}, m, r \in \mathbb{Z}_q)$. Output $\mathsf{com} = g^m \cdot h^r$, and $\mathsf{aux} = (m, r)$.
- $\mathsf{Pedersen.Open}(\mathsf{pp}, \mathsf{com}, \mathsf{aux})$. Output 1 if $\mathsf{com} = g^m \cdot h^r$.

2.2.3 Information-Theoretic Message Authentication Codes (ITMACs)

An ITMAC is a cryptographic primitive that provides unconditional security, ensuring that an adversary cannot forge a valid tag for a message with non-negligible probability, even with unlimited computational resources. An ITMAC consists of three probabilistic PPT algorithms $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Verify})$ over the message domain $\mathcal{M}$:

- $k \leftarrow \mathsf{Gen}(1^\lambda)$. Generates a secret key k of length λ, where λ is the security parameter.
- $t \leftarrow \mathsf{Tag}(k, m)$. Computes a tag t for a message $m \in \mathcal{M}$ using the key k.
- $\{0, 1\} \leftarrow \mathsf{Verify}(k, m, t)$. Verifies if t is a valid tag for the message $m \in \mathcal{M}$ under the key k. Returns 1 (*accept*) if valid and 0 (*reject*) otherwise.

An ITMAC must satisfy the following properties:

- **Correctness:** For all keys k generated by Gen and all messages m:

$$\mathsf{Verify}(k, m, \mathsf{Tag}(k, m)) = 1.$$

- **Unforgeability:** Even with unlimited computational resources, an adversary cannot produce a valid tag t for a new message $m^* \notin \mathcal{M}_{\text{query}}$, where $\mathcal{M}_{\text{query}}$ is the set of messages for which the adversary has queried $\mathsf{Tag}(k, \cdot)$.

$$\Pr\left[\mathsf{Verify}(k, m^*, t^*) = 1 \land m^* \notin \mathcal{M}_{\text{query}}\right] \leq \epsilon,$$

  where ϵ is a negligible probability determined by the key and message lengths.

2.3 Computational Hardness Assumptions

2.3.1 Discrete-Logarithm-Based Assumptions

Let $\mathsf{DLog.Sample}(1^\lambda)$ be a polynomial-time algorithm that on input the security parameter λ, outputs $(\mathbb{G}_\lambda, g, q)$ such that $\mathbb{G}_\lambda$ is a cyclic group of prime order q, and g is a generator of $\mathbb{G}_\lambda$.

Assumption 2.3.1 (Discrete-Logarithm-Based Assumptions). *Let λ be the security parameter and* $(\mathbb{G}_\lambda, g, p) \leftarrow\!\!\$\ \mathsf{DLog.Sample}$.

- *The DLog assumption* is considered computationally hard in a cyclic group $\mathbb{G}_\lambda$ of prime order q. For any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$, the probability of $\mathcal{A}(g, y) = x$ is negligible:

$$\Pr[\mathcal{A}(g, y) = x \mid y = g^x, x \in \mathbb{Z}_q] \leq \text{negl}(\lambda),$$

- *The Computational Diffie-Hellman (CDH) assumption* is a variant of the DLog problem, such that for any PPT algorithm $\mathcal{A}$, the probability of $\mathcal{A}(g, g^a, g^b) = g^{ab}$ is negligible:

$$\Pr[\mathcal{A}(g, g^a, g^b) = g^{ab}] \leq \text{negl}(\lambda),$$

- *The Decisional Diffie-Hellman (DDH) assumption* strengthens the CDH problem by requiring the indistinguishability of g^{ab} from a random group element. For any PPT algorithm $\mathcal{A}$, the advantage in distinguishing g^{ab} from a random element is negligible:

$$\left|\Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, T) = 1]\right| \leq \text{negl}(\lambda),$$

  where T is sampled uniformly at random from $\mathbb{G}_\lambda$.

Assumption 2.3.2 (Power-DDH, [CNs07]). *The* power-DDH *assumption states that for a group* $\mathbb{G}_\lambda = \langle g \rangle$ *of prime order* p *generated from* $\mathsf{DLog.Sample}(1^\lambda)$*, for any polynomially-bounded* $\ell \in \mathbb{N}$*, it holds that:*

$$\left(g, g^r, g^{r^2}, \ldots, g^{r^{\ell-1}}, g^{r^\ell}\right) \approx \left(g, g^r, g^{r^2}, \ldots, g^{r^{\ell-1}}, g^t\right),$$

where $r, t \leftarrow\!\$\; \mathbb{Z}_p^*$.

Assumption 2.3.3 (Sparse Power-DDH). *The* sparse power-DDH *assumption states that for a group* $\mathbb{G}_\lambda = \langle g \rangle$ *of prime order* p *generated from* $\mathsf{DLog.Sample}(1^\lambda)$*, for any polynomially-bounded* $\ell \in \mathbb{N}$ *and* $S \subset [\ell]$*, it holds that:*

$$\left(g, (g^{r^i})_{i \in S}, (g^{r^i})_{i \in [\ell] \setminus S}\right) \approx \left(g, (g^{r^i})_{i \in S}, (g^{t_i})_{i \in [\ell] \setminus S}\right),$$

where $r \leftarrow\!\$\; \mathbb{Z}_p^*$*, and* $t_i \leftarrow\!\$\; \mathbb{Z}_p^*$ *for all* $i \in [\ell] \setminus S$.

2.3.2 Decisional Composite Residuosity Assumption

Let $\mathsf{SampleModulus}$ be a polynomial-time algorithm that on input the security parameter λ, outputs (N, P, Q), where $N = PQ$ for λ-bit primes P and Q.

Assumption 2.3.4 (Decisional Composite Residuosity assumption, [Pai99]). Let λ be the security parameter. We say that the Decision Composite Residuosity (DCR) problem is hard relative to $\mathsf{SampleModulus}$ if $(N, x) \approx (N, x^N)$ where $(N, P, Q) \leftarrow\!\$\; \mathsf{SampleModulus}(1^\lambda)$, $x \leftarrow\!\$\; \mathbb{Z}_{N^2}^*$, and x^N is computed modulo N^2.

Note that $\mathbb{Z}_{N^2}^*$ can be written as a product of subgroups $\mathbb{H} \times \mathbb{NR}_N$, where $\mathbb{H} = \{(1+N)^i : i \in [N]\}$ is of order N, and $\mathbb{NR}_N = \{x^N : x \in \mathbb{Z}_{N^2}^*\}$ is the subgroup of N-th residues that has order $\phi(N)$.

Paillier-ElGamal Cryptosystem. The Paillier-ElGamal cryptosystem [CS02; BCP03] is defined by a triple $(\mathsf{PaillierEG.Gen}, \mathsf{PaillierEG.Enc}, \mathsf{PaillierEG.Dec})$, and boils down to using the ElGamal cryptosystem over the group $(\mathbb{Z}_{N^2}^\star, \times)$ where N is a Blum integer of the form $N = PQ$, where P and Q are primes:

- $\mathsf{PaillierEG.Gen}(1^\lambda)$. Sample $g' \leftarrow\!\$\; [N^2]$, $d \leftarrow\!\$\; [N^2]$, set $g \leftarrow (g')^{2N} \bmod N^2$, and output $(\mathsf{pk} = g^d \bmod N^2, \mathsf{sk} = d)$.
- $\mathsf{PaillierEG.Enc}(\mathsf{pk}, x)$. Sample $r \leftarrow\!\$\; [N]$, and output $\mathsf{ct} = (g^r, \mathsf{pk}^r \cdot (1+N)^x)$.
- $\mathsf{PaillierEG.Dec}(\mathsf{sk}, \mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1))$. Set $\mathsf{ct}' \leftarrow \mathsf{ct}_1 \cdot (\mathsf{ct}_0)^{-d} \bmod N^2$, and output $x = \frac{\mathsf{ct}'-1}{N}$.

Assuming the DCR assumption (Assumption 2.3.4), the Paillier-ElGamal cryptosystem is semantically secure.

2.3.3 Learning Parity with Noise (LPN)

We define the LPN assumption over a ring $\mathcal{R}$ with dimension k, number of samples n, w.r.t. a code generation algorithm $\mathbf{C}$, and a noise distribution $\mathcal{D}$:

Definition 2.3.1 (Dual LPN). *Let $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_{k,n}(\mathcal{R})\}_{k,n\in\mathbb{N}}$ denote a family of efficiently sampleable distributions over a ring $\mathcal{R}$, such that for any $k, n \in \mathbb{N}$, $\mathsf{Im}(\mathcal{D}_{k,n}(\mathcal{R})) \subseteq \mathcal{R}^n$. Let $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(k, n, \mathcal{R})$ outputs a matrix $H \in \mathcal{R}^{k\times n}$. For dimension $k = k(\lambda)$, number of samples (or block length) $n = n(\lambda)$, and ring $\mathcal{R} = \mathcal{R}(\lambda)$, the (dual) $(\mathcal{D}, \mathbf{C}, \mathcal{R})$ - $\mathsf{LPN}(k, n)$ assumption states that*

$$\{(H, \mathbf{b})\textit{s.t. } H \leftarrow\!\!\$\ \mathbf{C}(k, n, \mathcal{R}), \mathbf{e} \leftarrow\!\!\$\ \mathcal{D}_{k,n}(\mathcal{R}), \mathbf{b} \leftarrow H \cdot \mathbf{s}\}$$
$$\approx \{(H, \mathbf{b})\textit{s.t. } H \leftarrow\!\!\$\ \mathbf{C}(k, n, \mathcal{R}), \mathbf{b} \leftarrow\!\!\$\ \mathcal{R}^n\}.$$

The dual LPN assumption is also called *syndrome decoding assumption* in the code-based cryptography literature. The dual LPN assumption as written above is equivalent to the *primal* LPN assumption with respect to G (a matrix $G \in \mathcal{R}^{n\times n-k}$ such that $H \cdot G = 0$), which states that $G \cdot \mathbf{s} + \mathbf{e}$ is indistinguishable from random, where $\mathbf{s} \leftarrow\!\!\$\ \mathcal{R}^{n-k}$ and $\mathbf{e} \leftarrow\!\!\$\ \mathcal{D}_{k,n}(\mathcal{R})$; the equivalence follows from the fact that $\dot{H}(G \cdot \mathbf{s} + \mathbf{e}) = H \cdot \mathbf{e}$.

The standard LPN assumption refers to the case where H is a uniformly random matrix over $\mathbb{F}_2$, and $\mathbf{e}$ is sampled from $\mathsf{Ber}_r(\mathbb{F}_2)$, where r is called the *noise rate*. Other common noise distributions include exact noise (the noise vector $\mathbf{e}$ is a uniformly random weight-rn vector from $\mathbb{F}_2^n$; this is a common choice in concrete LPN-based constructions) and regular noise (the noise vector $\mathbf{e}$ is a concatenation of rn random unit vectors from $\mathbb{F}_2^{1/r}$, widely used in the PCG literature [BCG+18; BCG+19b; BCG+19a]).

Known constructions of subfield-VOLE use various flavors of the dual LPN assumption with regular noise over a finite field. For example, the work of [BCG+18] suggests relying on an LDPC code, while [BCG+19a] uses quasi-cyclic codes, and [CRR21] uses a new family of codes, called Silver codes.

2.3.4 Ring Learning Parity with Noise (Ring-LPN)

We now define the Ring-LPN assumption, a variant of the dual LPN assumption over polynomial rings, first introduced in [HKL+12] The assumption has been used in multiple works since. Ring-LPN is the natural “ring analog” of LPN, in the same way that ring-LWE is the ring analog of LWE.

Definition 2.3.2 (Ring-LPN). *Let $\mathcal{R} = \mathbb{F}[X]/(F(X))$ for some field $\mathbb{F}$ and degree-N polynomial $F(X) \in \mathbb{Z}[X]$, and let $m, t \in \mathbb{N}$. Let HW_t be the distribution over R_p that is obtained via sampling t noise positions $A \leftarrow [0..N)^t$ as well as t payloads $\mathbf{b} \leftarrow \mathbb{Z}_p^t$ uniformly at random, and outputting $e(X) := \sum_{j=0}^{t-1} \mathsf{b}[j] \cdot X^{A[j]}$. The R-$\mathsf{LPN}_{p,q,t}$ problem is hard if for any PPT adversary $\mathcal{A}$, it holds that*

$$|\Pr[\mathcal{A}((a_i, a_i \cdot s + e_i)_{i=1}^m) = 1] - \Pr[\mathcal{A}((a_i, u_i)_{i=1}^m) = 1]| \leq \mathsf{negl}(\lambda)$$

where the probabilities are taken over the random choices of the values $a_1, \ldots, a_m, u_1, \ldots, u_m \leftarrow \mathcal{R}_p$, $s, e_1, \ldots, e_m \leftarrow \mathsf{HW}_t$ and the randomness of $\mathcal{A}$.

We note that, for our contribution about OLE-based PSI, we build upon the PCG-based OLE of [BCG+20b]. The latter uses a relatively new flavor of the ring-LPN assumption, over a polynomial ring where the polynomial splits completely; however, in this work, we do *not* need this new flavor, and instead rely solely on the (relatively well-established) standard ring-LPN assumption over a polynomial ring with an irreducible polynomial.

2.3.5 Quasi-Abelian Syndrome Decoding Problem (QA-SD)

The *Quasi-Abelian Syndrome Decoding* assumption (QA-SD) was introduced in [BCC+23]. It can be viewed as a generalization of the ring-LPN assumption over suitable multivriate polynomial rings. Following [BCC+23], we formalize it as a syndrome decoding assumption for quasi-abelian codes, which provides a convenient framework to study it and analyze its hardness guarantees. Let $\mathbb{G}$ be a finite Abelian group. The group algebra of $\mathbb{G}$ with coefficients in the finite field $\mathbb{F}_\mathsf{q}$ is the set of formal linear combinations $\left\{\sum_{g\in\mathbb{G}} a_g g \mid a_g \in \mathbb{F}_\mathsf{q}\right\}$, which is an $\mathbb{F}_\mathsf{q}$-vector space of dimension $|\mathbb{G}|$, endowed with the convolution product:

$$\left(\sum_{g\in\mathbb{G}} a_g g\right)\left(\sum_{g\in\mathbb{G}} b_g g\right) := \sum_{g\in G}\left(\sum_{h\in\mathbb{G}} a_h b_{hg^{-1}}\right) g.$$

It can be seen that this product is commutative. The *Hamming* weight $w_H(a)$ of an element $a \in \mathbb{F}_\mathsf{q}[\mathbb{G}]$ is the number of its non zero coordinates in the basis $(g)_{g\in\mathbb{G}}$. This is a well-defined notion since it does not depend on the ordering of the elements of $\mathbb{G}$.

Recall that a finite Abelian group is nothing more than a direct product of cyclic groups:

$$\mathbb{G} \simeq \mathbb{Z}/d_1\mathbb{Z} \times \cdots \times \mathbb{Z}/d_r\mathbb{Z},$$

where the d_i's can be equal. Then, the group algebra $\mathbb{F}_\mathsf{q}[\mathbb{G}]$ admits an explicit description as some particular multivariate polynomial ring:

$$\mathbb{F}_\mathsf{q}[\mathbb{G}] \simeq \mathbb{F}_\mathsf{q}[X_1, \dots, X_r]/(X_1^{d_1} - 1, \dots, X_r^{d_r} - 1),$$

where the isomorphism is given by $(k_1, \dots, k_r) \mapsto X_1^{k_1} \cdots X_r^{k_r}$, and extended by linearity. We are now ready to define the main hard problem, which can be stated as a search and a decisional variant.

Definition 2.3.3 ((Search) $\mathsf{QA\text{-}SD}(q, c, t, \mathbb{G})$)**.** *Let $\mathbb{G}$ be a finite Abelian group, $\mathbb{F}_\mathsf{q}[\mathbb{G}]$ its algebra with coefficients in the finite field $\mathbb{F}_\mathsf{q}$, and let $c \geq 2$ be some constant integer called the compression factor. Given a target Hamming weight $t \in \{1, \dots, |\mathbb{G}|\}$ and a probability distribution Φ_t which outputs elements $x \in \mathbb{F}_\mathsf{q}[\mathbb{G}]$ such that $\mathbb{E}(w_H(x)) = t$. Given access to a pair of the form $(\mathbf{a}, \langle\mathbf{a}, \mathbf{e}\rangle + e_0)$ where $\mathbf{a}$ is uniformly distributed over $\mathbb{F}_\mathsf{q}[\mathbb{G}]^{c-1}$ and $\mathbf{e}' := (e_0, \mathbf{e}) \in \mathbb{F}_\mathsf{q}[\mathbb{G}]^c$ is formed by independent elements distributed according to Φ_t, the goal is to recover the error term $\mathbf{e}'$.*

Definition 2.3.4 ((Decisional) $\mathsf{QA\text{-}SD}(q, c, t, \mathbb{G})$)**.** *Let $\mathbb{G}$ be a finite Abelian group, $\mathbb{F}_\mathsf{q}[\mathbb{G}]$ its algebra with coefficients in the finite field $\mathbb{F}_\mathsf{q}$, and let $c \geq 2$ be some constant integer called the compression factor. Given a target Hamming weight $t \in \{1, \dots, |\mathbb{G}|\}$ and a probability distribution Φ_t which outputs elements $x \in \mathbb{F}_\mathsf{q}[\mathbb{G}]$ such that $\mathbb{E}(w_H(x)) = t$, the* Quasi-Abelian Syndrome Decoding *problem asks to distinguish, with a non-negligible advantage, between the distributions:*

$$\mathcal{D}_0 : \quad \left((a^{(i)})_{i\in\{1,\dots,c-1\}}, u\right) \quad \textit{where } a^{(i)}, u \leftarrow\!\$\ \mathbb{F}_\mathsf{q}[\mathbb{G}]$$

$$\mathcal{D}_1 : \quad \left((a^{(i)})_{i\in\{1,\dots,c-1\}}, \sum_{i=1}^{c-1} a^{(i)} e_i + e_0\right) \quad \textit{where } a^{(i)} \leftarrow\!\$\ \mathbb{F}_\mathsf{q}[\mathbb{G}] \textit{ and } e_i \leftarrow\!\$\ \Phi_t.$$

We say that the $\mathsf{QA\text{-}SD}(q, c, t, \mathbb{G})$ assumption holds when this problem is hard for every non-uniform polynomial time distinguisher.

Remark 2.3.1. *When it is clear from the context, we might drop the dependency in q and simply write* $\mathsf{QA\text{-}SD}(c, t, \mathbb{G})$.

In general we consider Φ_t to output uniform elements of Hamming weight t, or a *regular* variant where the basis $(g)_{g\in\mathbb{G}}$ is split into t blocks of size $|\mathbb{G}|/t$ (except maybe the last one) such that each block contains exactly one t.

Relation to linear codes. Fix an ordering of the elements of $\mathbb{G}$ and for every element $a \in \mathbb{F}_\mathsf{q}[\mathbb{G}]$, denote by M_a the $|\mathbb{G}| \times |\mathbb{G}|$ matrix representing the multiplication by a. A code having a parity-check matrix formed by multiple blocks of the form M_a is known as a *Quasi-Abelian* code of group $\mathbb{G}$ (or a quasi-$\mathbb{G}$ code). Formally, it is an $\mathbb{F}_\mathsf{q}[\mathbb{G}]$-submodule of the free module $\mathbb{F}_\mathsf{q}[\mathbb{G}]^\ell$ for some integer $\ell > 0$. When $\mathbb{G} = \{1\}$, then $\mathbb{F}_\mathsf{q}[\mathbb{G}]$ is nothing but the finite field $\mathbb{F}_\mathsf{q}$, and therefore a quasi-$\{1\}$ code is simply an $\mathbb{F}_\mathsf{q}$-linear code. On the other hand, when $\mathbb{G} = \mathbb{Z}/n\mathbb{Z}$ is cyclic, then $\mathbb{F}_\mathsf{q}[\mathbb{G}] \simeq \mathbb{F}_\mathsf{q}[X]/(X^n - 1)$. Therefore, quasi-$\mathbb{Z}/n\mathbb{Z}$ codes are exactly *quasi-cyclic* codes, we refer to [BCC+23, Section 4] for more information on Quasi-Abelian codes.

Now, a sample $(\mathbf{a}, \langle\mathbf{a}, \mathbf{e}\rangle) + e_0)$ corresponds to a pair $(\mathbf{H}, \mathbf{He'})$ where

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{|\mathbb{G}|} & \mathbf{M}_{a_1} & \cdots & \mathbf{M}_{a_{c-1}} \end{bmatrix}$$

is by definition, a parity-check matrix of some random quasi-Abelian code of rate $1 - 1/c$, in systematic form, and $\mathbf{e'} = (e_0, \mathbf{e})$ is an error vector of length $c|\mathbb{G}|$ and weight ct, with a c-split structure (which is standard when dealing with structured variants of the decoding problem).

2.4 (Constrained) Pseudorandom Function (PRF)

2.4.1 Pseudorandom Functions

Definition 2.4.1 ((Weak) Pseudorandom Function (wPRF, PRF), [GGM84; NR95])**.** Let $\lambda \in \mathbb{N}$ be a security parameter. A (weak) pseudorandom function with domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$, and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, consists of the following two polynomial-time algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{msk})$: A probabilistic algorithm that on input the security parameter λ, outputs a master secret key $\mathsf{msk} \in \mathcal{K}$.
- $\mathsf{Eval}(\mathsf{msk}, x) \to y$: A deterministic algorithm that on input the master secret key msk, and an input value $x \in \mathcal{X}$, outputs a value $y \in \mathcal{Y}$.

We say that the pair $(\mathsf{KeyGen}, \mathsf{Eval})$ is a

- **pseudorandom function** (PRF) if for any PPT adversary $\mathcal{A}$, it holds that

$$\left|\Pr\left[\mathcal{A}^{\mathsf{Eval}(\mathsf{msk},\cdot)}(1^\lambda) = 1 \middle| \mathsf{msk} \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)\right] - \Pr\left[\mathcal{A}^{RF(\cdot)}(1^\lambda) = 1 \middle| RF \xleftarrow{\$} \mathcal{F}\right]\right| = \mathsf{negl}(\lambda),$$

  where $\mathcal{F}$ is the set of all functions with domain $\mathcal{X}$ and range $\mathcal{Y}$.

- **weak pseudorandom function** (wPRF) if for any PPT adversary $\mathcal{A}$ and any polynomially bounded number $Q \in \mathbb{N}$, it holds that

$$\left\{ \left((x_i, \mathsf{Eval}(\mathsf{msk}, x_i))_{i\in[Q]}\right) \middle| \begin{array}{c} \mathsf{msk} \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda) \\ \forall i \in [Q] : x_i \xleftarrow{\$} \mathcal{X} \end{array} \right\} \approx_c \left\{ \left((x_i, y_i)_{i\in[Q]}\right) \middle| \begin{array}{c} \forall i \in [Q] : \\ x_i \xleftarrow{\$} \mathcal{X},\ y_i \xleftarrow{\$} \mathcal{Y} \end{array} \right\}.$$

2.4.2 Constrained Pseudorandom Functions

Definition 2.4.2 (Constrained Pseudorandom Functions). Let λ be a security parameter. A Constrained Pseudorandom Function (CPRF) with domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$, and range $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$, that supports a class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$, where each $C_\lambda \in \mathcal{C}_\lambda$ has domain $\mathcal{X}_\lambda$ and range $\{0,1\}$, consists of the following four polynomial-time algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$: The master key generation algorithm is a probabilistic algorithm that on input the security parameter λ, outputs a public parameter pp and a master secret key $\mathsf{msk} \in \mathcal{K}$.
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x) \to y$: The evaluation algorithm is a deterministic algorithm that on input the public parameter pp, the master secret key msk, and an input $x \in \mathcal{X}$, outputs a value $y \in \mathcal{Y}$.
- $\mathsf{Constrain}(\mathsf{msk}, C) \to \mathsf{ck}_C$: The constrained key generation algorithm is a probabilistic algorithm that on input the master secret key msk, and a circuit $C \in \mathcal{C}$, outputs a constrained key ck_C.
- $\mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_C, x) \to y$: The constrained evaluation algorithm is a deterministic algorithm that on input the public parameter pp, a constrained key ck_C, and an input $x \in \mathcal{X}$, outputs a value $y \in \mathcal{Y}$.

Correctness. For any security parameter λ, any constrain $C \in \mathcal{C}$, and any input $x \in \mathcal{X}$ such that $C(x) = 0$, we have:

$$\Pr\left[\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x) \neq \mathsf{CEval}(\mathsf{pp}, \mathsf{ck}_C, x) : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{msk} \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ \mathsf{ck}_C \leftarrow \mathsf{Constrain}(\mathsf{msk}, C) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

1-Key Selective Security. We say that a CPRF is 1-key selectively secure if the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible:

- **Setup:** The challenger runs $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, initializes a set $S_{\mathsf{eval}} = \varnothing$, and chooses a random bit $b \leftarrow\$ \{0,1\}$. It then sends pp to $\mathcal{A}$.
- **Selective Choice of Constraint:** The adversary chooses a (single) circuit $C \in \mathcal{C}$ and sends it to the challenger.
- **Constrained Key Generation:** The challenger computes $\mathsf{ck}_C \leftarrow \mathsf{Constrain}(\mathsf{msk}, C)$ and returns the constrained key ck_C to $\mathcal{A}$.
- **Pre-Challenge Evaluation Queries:** $\mathcal{A}$ can adaptively send arbitrary input values $x \in \mathcal{X}$ to the challenger. The challenger computes $y \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$ and returns y to $\mathcal{A}$. It also updates $S_{\mathsf{eval}} \leftarrow S_{\mathsf{eval}} \cup \{x\}$.
- **Challenge Phase:** $\mathcal{A}$ sends an input $x^* \in \mathcal{X}$ as its challenge query to the challenger with the restriction that $x^* \notin S_{\mathsf{eval}}$ and $C(x^*) \neq 0$. If it holds that $b = 0$, then the challenger computes $y^* \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x^*)$. Otherwise, if $b = 1$, the challenger samples a random value $y^* \overset{\$}{\leftarrow} \mathcal{Y}$. Finally, the challenger returns y^* to $\mathcal{A}$.
- **Post-Challenge Evaluation Queries:** $\mathcal{A}$ continues the queries as before, with the restriction that it cannot query x^* as an evaluation query.

- **Guess:** $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

1-Key Selective Constraint-Hiding. We say that a CPRF is selectively 1-key constraint-hiding if the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible:

- **Setup:** The challenger runs $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and chooses a random bit $b \xleftarrow{\$} \{0,1\}$. It then sends pp to $\mathcal{A}$.
- **Selective Choice of Constraint:** The adversary chooses a (single) pair of circuits $(C_0, C_1) \in \mathcal{C}$ and sends the pair to the challenger.
- **Constrained Key Generation:** The challenger computes $\mathsf{ck}_b \leftarrow \mathsf{Constrain}(\mathsf{msk}, C_b)$, and returns ck_b to $\mathcal{A}$.
- **Evaluation Queries:** $\mathcal{A}$ can query the output of the evaluation algorithm on arbitrary inputs $x \in \mathcal{X}$, with the restriction that $C_0(x) = C_1(x)$. On such inputs, the challenger computes and returns $y \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$ to $\mathcal{A}$.
- **Guess:** $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

In both of the games described above, $\mathcal{A}$ wins if $b' = b$. We also define the advantage of $\mathcal{A}$ in winning a game as $|2 \cdot \Pr[\mathcal{A} \text{ wins}] - 1|$, where the probability is over the internal coins of $\mathcal{A}$ and the challenger.

No-Evaluation Security. 1-key selective no-evaluation security (resp. 1-key selective no-evaluation constraint-hiding) is defined similarly with the extra restriction that the adversary cannot issue any pre-challenge or post-challenge query (resp. any evaluation query).

2.5 Function Secret Sharing (FSS)

Function secret sharing (FSS), introduced in [BGI15; BGI16], allows a dealer to succinctly secret share a function with two parties. An FSS scheme splits a secret function $f : \mathcal{X} \to \mathbb{G}$, where $\mathbb{G}$ is some Abelian group into *keys* K_0, K_1 that can be used by party $\sigma \in \{0,1\}$ to evaluate the function on an input $x \in \mathcal{X}$ and obtain the share $[\![f(x)]\!]_\sigma$ of the result. We focus on FSS for *point functions* which are known as Distributed Point Functions (DPFs).

Distributed Point Functions. Let $\mathcal{X}$ be an input domain and $\mathbb{G}$ be an Abelian group. A *point function* $P_{\alpha,\beta} : \mathcal{X} \to \mathbb{G}$ is a function that evaluates to message $\beta \in \mathbb{G}$ on a single input $\alpha \in \mathcal{X}$, and evaluates to $0 \in \mathbb{G}$ on all other inputs $x \neq \alpha \in \mathcal{X}$. A *distributed* point function (Definition 2.5.1) is a point function that is encoded into a pair of keys. Each key can be used to obtain an additive *secret-share* of the point function $P_\alpha(x)$, for any input $x \in \mathcal{X}$.

Definition 2.5.1 (Distributed Point Function (DPF) [GI14; BGI16]). *Let λ be the security parameter, $\mathcal{X}$ be an input domain, and $\mathbb{G}$ be an Abelian group. A* DPF *scheme (with a full-domain evaluation procedure) consists of a tuple of efficient algorithms* $\mathsf{DPF} = (\mathsf{Gen}, \mathsf{FullEval})$ *with the following syntax.*

- $\mathsf{DPF.Gen}(1^\lambda, 1^n, \alpha, \beta) \to (K_0, K_1)$. *Takes as input a security parameter, a domain size n, and index $\alpha \in \mathcal{X}$ and a payload $\beta \in \mathbb{G}$. Outputs two evaluation keys K_0 and K_1.*
- $\mathsf{DPF.FullEval}(\sigma, K_\sigma) \to \mathbf{v}_\sigma$. *Takes as input the party index σ and an evaluation key K_σ. Outputs a vector $\mathbf{v}_\sigma$.*

These algorithms must satisfy correctness, security, and efficiency:

Correctness. *A DPF is said to be* correct *if for all $\alpha \in \mathcal{X}$, all $\beta \in \mathbb{G}$, and all pairs of keys generated according to* $\mathsf{DPF.Gen}(1^\lambda, 1^n, \alpha, \beta)$*, the sum of the individual outputs from* $\mathsf{DPF.FullEval}$ *result in the one-hot basis vector scaled by the message β,*

$$\Pr\left[\ \mathsf{FullEval}(0, K_0) + \mathsf{FullEval}(1, K_1) = \beta \cdot \mathbf{e}_{\boldsymbol{\alpha}}\ \right] = 1,$$

where $\mathbf{e}_{\boldsymbol{\alpha}} \in \mathbb{G}^{|\mathcal{X}|}$ is the α-th basis vector.

Security. *A DPF is said to be* secure *if each individual evaluation key output by* $\mathsf{DPF.Gen}$ *leaks nothing about (α, β) to a computationally bounded adversary. Formally, there exists an efficient simulator $\mathcal{S}$ such that $\{K_\sigma\} \approx \mathcal{S}(1^\lambda, 1^n, \sigma)$, where $\approx$ denotes the computational indistinguishability of distributions.*

Efficiency. *A DPF is said to be* efficient *if the size of each key is sublinear in the domain size. That is, for all $\sigma \in \{0, 1\}$, $|K_\sigma| = |\mathcal{X}|^\epsilon$ for some $\epsilon < 1$.*

FSS for the sum of point functions. We let SPFSS be an FSS scheme for the class of *sums of point functions*: Functions of the form $f(x) = \sum_i f_{s_i, y_i}(x)$, where each $f_{s_i, y_i}(\cdot)$ evaluates to y_i on s_i, and to 0 everywhere else. As in previous works, we will use efficient constructions of SPFSS in our constructions of PCGs.

2.6 Pseudorandom Correlation Generators (PCGs)

Pseudorandom Correlation Generators (PCGs) were introduced in a line of work [BCG+18; BCG+19b; BCG+19a]. They allow two parties to generate a target correlation by locally stretching seeds that are significantly smaller than the number of correlations. Slightly more formally, a PCG for a target correlation C (which samples pairs of long correlated strings (y_0, y_1)) consists of two algorithms $(\mathsf{Gen}, \mathsf{Expand})$ such that $\mathsf{Gen}(1^\lambda)$ outputs a pair of short, correlated seeds $(\mathsf{seed}_0, \mathsf{seed}_1)$ and $\mathsf{Expand}(\sigma, \mathsf{seed}_\sigma)$ outputs a long string $\tilde{y}_\sigma$.

- Correctness: $(\tilde{y}_0, \tilde{y}_1)$ are indistinguishable from a random sample from C,
- Security: given $(\mathsf{seed}_{1-\sigma}, \tilde{y}_\sigma)$ looks like a random sample from C conditioned on satisfying the target correlation with $\mathsf{Expand}(1 - \sigma, \mathsf{seed}_{1-\sigma})$, for $\sigma = \{0, 1\}$.

2.6.1 Defining Pseudorandom Correlation Generators

At a high level, a pseudorandom correlation generator (PCG) is a paradigm that takes as input a pair of short, correlated seeds and produces long correlated pseudorandom strings through a deterministic and locally executable expansion process.

Correctness. The output of the PCG should be computationally indistinguishable from truly random correlated strings.

Security. The security of PCGs a security model based on indistinguishability. Specifically, an adversary with access to one seed k_σ should not be able to distinguish the pseudorandom string $R_{1-\sigma}$ from a random string chosen subject to the correlation (R_0, R_1), where $R_\sigma = \text{PCG}(k_\sigma)$. In essence, the adversary's knowledge of one seed should not reveal more about the other party's pseudorandom string than what is already evident from their own string.

The formal definition of PCGs and the related definitions such as correlation generators are taken almost verbatim from [BCG+18; BCG+19b] as below.

Correlation Generators. The concept of a correlation generator can be considered as a PPT algorithm outputting correlated elements.

Definition 2.6.1 (Correlation Generator). *A PPT algorithm C is called a* correlation generator*, if C on input 1^λ outputs a pair of elements in $\{0,1\}^n \times \{0,1\}^n$ for $n \in poly(\lambda)$.*

Definition 2.6.2 (Reverse-sampleable Correlation Generator). *Let C be a correlation generator. We say C is* reverse sampleable *if there exists a PPT algorithm* RSample *such that for $\sigma \in \{0,1\}$ the correlation obtained via:*

$$\{(R'_0, R'_1) \mid (R_0, R_1) \leftarrow C(1^\lambda), R'_\sigma := R_\sigma, R'_{1-\sigma} \leftarrow \mathsf{RSample}(\sigma, R_\sigma)\}$$

is computationally indistinguishable from $C(1^\lambda)$.

Definition 2.6.3 (Pseudorandom Correlation Generator (PCG)). *Let C be a reverse-sampleable correlation generator. A pseudorandom correlation generator (PCG) for C is a pair of algorithms* (PCG.Gen, PCG.Expand) *with the following syntax:*

- $\mathsf{PCG.Gen}(1^\lambda)$*: A PPT algorithm that, given a security parameter λ, outputs a pair of seeds (k_0, k_1).*
- $\mathsf{PCG.Expand}(\sigma, k_\sigma)$*: A polynomial-time algorithm that, given party index $\sigma \in \{0,1\}$ and a seed k_σ, outputs a bit string $R_\sigma \in \{0,1\}^n$.*
- ***Correctness.*** *The correlation obtained via:*

  $$\{(R_0, R_1) \mid (k_0, k_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \textit{ for } \sigma \in \{0,1\}\}$$

  is computationally indistinguishable from $C(1^\lambda)$.
- ***Security.*** *For any $\sigma \in \{0,1\}$, the following two distributions are computationally indistinguishable:*

  $$\{(k_{1-\sigma}, R_\sigma) \mid (k_0, k_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma)\}$$

  and

  $$\left\{ (k_{1-\sigma}, R_\sigma), \middle| \begin{array}{l} (k_0, k_1) \leftarrow \mathsf{PCG.Gen}(1^\lambda), \\ R_{1-\sigma} \leftarrow \mathsf{PCG.Expand}(1-\sigma, k_{1-\sigma}), R_\sigma \leftarrow \mathsf{RSample}(\sigma, R_{1-\sigma}) \end{array} \right\}$$

  where RSample *is the reverse sampling algorithm for correlation C.*

For a PCG construction to be non-trivial, it is required that the seeds are significantly shorter than the output size. Our work considers two primary types of PCGs, vector OLEs and OLEs. Vector OLEs have a significantly lower setup cost compared to OLEs, making it ideal for applications where high efficiency is crucial. On the other hand, OLEs are suited for constructing more advanced cryptographic primitives that require higher security guarantees or more concrete properties.

2.6.2 Vector Oblivious Linear Evaluation (Vector OLE)

The vector OLE correlation [BCG+19b; BCG+19a] over a field $\mathbb{F}$ is the following correlation:

$$\{((\mathbf{u}, \mathbf{v}), (\Delta, \mathbf{w})) | \, \mathbf{u}, \mathbf{v} \leftarrow\!\!\$ \, \mathbb{F}^n; \Delta \leftarrow\!\!\$ \, \mathbb{F}; \mathbf{w} \leftarrow \Delta \cdot \mathbf{u} + \mathbf{v}\}$$

Figure 2.1: $\mathcal{F}_{\mathsf{VOLE}}^{n,\mathbb{F}}$ in the malicious setting

PARAMETERS: Two parties, a sender and a receiver, an integer n, the size of the output vector, a finite field $\mathbb{F}$.
FUNCTIONALITY:

- **Initialize:** Upon receiving $(\mathsf{init}, \mathsf{InputS})$ and $(\mathsf{init}, \mathsf{InputR})$ from sender and receiver, sample $\Delta \leftarrow\$ \mathbb{F}$ if receiver is honest or receive $\Delta \in \mathbb{F}$ from the adversary otherwise. It stores global key Δ and sends Δ to receiver, and ignores all subsequent init commands.
- **Extend:** This procedure can be run multiple times. Upon receiving $(\mathsf{extend}, \mathsf{n})$ from sender and receiver, do:
    1. If receiver is corrupted then wait for $\mathcal{A}$ to send $\mathbf{w} \in \mathbb{F}^n$; samples $\mathbf{u} \leftarrow\$ \mathbb{F}^n$ and computes $\mathbf{v} := \mathbf{w} - \Delta \cdot \mathbf{u}$.
    2. If sender is corrupted then wait for $\mathcal{A}$ to send vectors $\mathbf{u} \in \mathbb{F}^n, \mathbf{v} \in \mathbb{F}^n$ and computes $\mathbf{w} := \Delta \cdot \mathbf{u} + \mathbf{v}$.
    3. Otherwise, samples $\mathbf{u} \leftarrow\$ \mathbb{F}^n, \mathbf{v} \leftarrow\$ \mathbb{F}^n$ and computes $\mathbf{w} := \Delta \cdot \mathbf{u} + \mathbf{v}$.
- **Return:** Functionality sends $(\mathsf{OutputS}, \mathbf{u},\ \mathbf{v})$ to sender and $(\mathsf{OutputR}, \Delta, \mathbf{w})$ to receiver.

We represent on Figure 2.1 the ideal functionality $\mathcal{F}_{\mathsf{VOLE}}^{n,\mathbb{F}}$ of vector-OLE in malicious setting. We define *subfield vector OLE* correlation as $\mathbf{u}$ is defined over a subfield of $\mathbb{F}$. In our concrete instantiations, we instantiate this functionality using the general template in [BCG+19b; BCG+19a; WYK+21], which can be instantiated under various flavors of the learning parity with noise assumption (LPN) and Puncturable Pseudorandom Function (PPRF). Using the protocol of [BCG+19a] to distribute the seeds[1], the theoretical communication cost to have $\mathcal{F}_{\mathsf{VOLE}}^{n,\mathbb{F}}$ is $(\log(cn/t)|\mathsf{OT}| + 10|\mathbb{F}|) \cdot t$, where t is Hamming weight of noises for the underlying LPN assumption, $|\mathsf{OT}| = O(\kappa)$ is the size of chosen OT [LWY+22] and c is the ratio between number of samples and the length of VOLE.

2.6.3 Oblivious Linear Evaluation (OLE)

We start with the ring OLE correlation [BCG+20b], a ring-OLE correlation over a ring $\mathcal{Q}$ is the following correlation:

$$\{((x_0, z_0), (x_1, z_1))|\ x_0, x_1, z_0 \leftarrow\$ \mathcal{Q}, z_1 \leftarrow x_0 \cdot x_1 - z_0\}$$

We define a *subfield ring OLE* as a ring OLE where x_0 belongs to a subring of $\mathcal{Q}$. Its functionality and random OLE functionality are shown in Figure 2.2. A Ring-OLE is typically constructed from a Distributed Point Function (DPF) (see Section 2.5 for details) under the ring-learning parity with noise (ring-LPN)assumption [BCG+20b]. The communication for setting up the seed of ring-OLE is $(ct)^2 \cdot (34 \log n + 10 \log|\mathbb{F}| + (2\lambda + 3)\log(2n) + 4\lambda) + ct \cdot (\log n + \log|\mathbb{F}|)$ where c is compression factor, t is number of noise such that ring-learning parity with noise (ring-LPN) assumption holds with these parameters [BCG+20b]. Random OLE can be *silently* generated from ring-OLE with an appropriate choice of polynomial ring $\mathcal{Q}$ as shown in [BCG+20b].

[1]This protocol uses a length-t reverse VOLE protocol as a black box, which we instantiate with the construction of [ADI+17].

Figure 2.2: $\mathcal{F}^{n,\mathcal{Q}}_{\mathsf{rOLE}}, \mathcal{F}^{\mathbb{F}}_{\mathsf{OLE}}$ in the malicious setting

PARAMETERS: There are two parties, a sender and a receiver, a finite polynomial ring $\mathcal{Q} = \mathbb{F}[x]/(F(x))$, a length n (for $\mathcal{F}^{n,\mathcal{Q}}_{\mathsf{rOLE}}$), and a finite field $\mathbb{F}$ (for $\mathcal{F}^{\mathbb{F}}_{\mathsf{OLE}}$).

FUNCTIONALITY $\mathcal{F}^{n,\mathcal{Q}}_{\mathsf{rOLE}}$:

- Upon receiving $(\mathsf{init}, \mathsf{InputS})$ and $(\mathsf{init}, \mathsf{InputR})$ from sender and receiver:
  1. If receiver is corrupted then wait for $\mathcal{A}$ to send $(x_1, z_1) \in \mathcal{Q}$; samples $x_0 \leftarrow\$ \mathcal{Q}$ and computes $z_0 := x_0 \cdot x_1 - z_1$.
  2. If sender is corrupted then wait for $\mathcal{A}$ to send $(x_0, z_0) \in \mathcal{Q}$; samples $x_1 \leftarrow\$ \mathcal{Q}$ and computes $z_1 := x_0 \cdot x_1 - z_0$.
  3. Otherwise, Sample uniformly random $x_0 \leftarrow\$ \mathcal{Q}$, $x_1, z_1 \leftarrow\$ \mathcal{Q}$, let $z_0 = x_0 \cdot x_1 - z_1 \in \mathcal{Q}$.
- Functionality sends Output $(\mathsf{OutputS}, x_0, z_0)$, $(\mathsf{OutputR}, x_1, z_1)$ to the sender and the receiver respectively.

FUNCTIONALITY $\mathcal{F}^{\mathbb{F}}_{\mathsf{OLE}}$: It is the same as in one of $\mathcal{F}^{n,\mathcal{Q}}_{\mathsf{rOLE}}$ when replacing $n = 1$, $\mathcal{Q} = \mathbb{F}$.

2.7 Pseudorandom Correlation Functions (PCFs)

At a high level, a *pseudorandom correlation function* (PCF) compresses, in short correlated keys, (superpolynomially large) correlated pseudorandom strings for some ideal correlation, e.g., strings of Beaver triples [Bea92][2]. Specifically, from correlated keys, parties can locally compute an unlimited number of correlations while maintaining the security.

We consider two different flavors of PCFs: *weak* PCFs (wPCF) and *strong* PCFs (sPCF). Analogously to PRFs, wPCFs guarantee security given access only to evaluations on uniformly random and independent inputs, while sPCFs guarantee security even for adaptively chosen inputs. Note that contrary to PRFs, the PCF literature treats *weak* PCFs as the default notion.

Reverse-Sampleable Correlations

We recall the definition of reverse-sampleable correlations from Definition 2.6.2, adjusting the notations to align with those used in our later definition of PCFs.

Definition 2.7.1 (Reverse-Sampleable Correlation). *Let $1 \leq \ell_0(\lambda), \ell_1(\lambda) \leq \mathsf{poly}(\lambda)$. Let $\mathcal{Y}$ be a probabilistic algorithm that, on input 1^λ, returns a pair of outputs $(y_0, y_1) \in \{0,1\}^{\ell_0(\lambda)} \times \{0,1\}^{\ell_1(\lambda)}$, defining a correlation on the outputs.*

We say that $\mathcal{Y}$ defines a reverse-sampleable *correlation if there exists a probabilistic polynomial time algorithm* $\mathsf{RSample}$ *which takes as input 1^λ, $\sigma \in \{0,1\}$, and $y_\sigma \in \{0,1\}^{\ell_\sigma(\lambda)}$, and outputs $y_{1-\lambda}^{\ell_{1-\sigma}(\lambda)}$, such that for all $\sigma \in \{0,1\}$ the following distributions are statistically close:*

$$\begin{aligned} &\{(y_0, y_1) \colon (y_0, y_1) \leftarrow\$ \mathcal{Y}(1^\lambda)\}, \\ \textit{and } &\{(y_0, y_1) \colon (y'_0, y'_1) \leftarrow\$ \mathcal{Y}(1^\lambda), y_\sigma \leftarrow y'_\sigma, y_{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_\sigma)\}\ . \end{aligned}$$

Definition 2.7.2 (OT Correlation). *A* (1-out-of-2, bit) OT correlation *can be defined as being sampled as a pair $((r_0, r_1), (b, r_b))$, where $r_0, r_1, b \leftarrow\$ \{0,1\}$.*

[2]Recall that a Beaver triple is a triplet additive shares $([a], [b], [c])$ where $a, b \leftarrow\$ \mathcal{R}$ for some ring $\mathcal{R}$, and $c = ab$.

Remark 2.7.1 (An OT Correlation is Reverse-Sampleable). A (1-out-of-2, bit) OT correlation is reverse-sampleable. *Indeed, observe that the reverse-sampling can be performed as follows.*

$\mathsf{RSample}(1^\sigma, \sigma, y_\sigma)$:

- *If* $\sigma = 0$*, parse* y_σ *as* $y_\sigma = (r_0, r_1)$*, sample* $b \leftarrow\!\$ \{0,1\}$*, and output* (b, r_b)*;*
- *Otherwise (i.e., if* $\sigma = 1$*) parse* y_σ *as* $y_\sigma = (b, r)$*, sample* $r' \leftarrow\!\$ \{0,1\}$*, and output* $((1-b)\cdot r + b\cdot r', b\cdot r + (1-b)\cdot r')$*.*

2.7.1 Weak Pseudorandom Correlation Functions (wPCFs)

We start by defining the notion of a *weak* pseudorandom correlation function.

Definition 2.7.3 ((Weak) Pseudorandom Correlation Function (wPCF), [BCG+20a, Definition 4.3]). *Let* $\mathcal{Y}$ *be a reverse-sampleable correlation with output length functions* $\ell_0(\lambda), \ell_1(\lambda)$ *and let* $\lambda \leq n(\lambda) \leq \mathsf{poly}(\lambda)$ *be an input length function. Let* $(\mathsf{wPCF.Gen}, \mathsf{wPCF.Eval})$ *be a pair of algorithms with the following syntax:*

- $\mathsf{wPCF.Gen}(1^\lambda)$ *is a probabilistic polynomial time algorithm that on input* 1^λ*, outputs a pair of keys* (k_0, k_1)*; we assume that* λ *can be inferred from the keys.*
- $\mathsf{wPCF.Eval}(\sigma, k_\sigma, x)$ *is a deterministic polynomial time algorithm that on input* $\sigma \in \{0,1\}$*, key* k_σ *and input value* $x \in \{0,1\}^{n(\lambda)}$*, outputs a value* $y_\sigma \in \{0,1\}^{\ell_\sigma(\lambda)}$*.*

We say that $(\mathsf{wPCF.Gen}, \mathsf{wPCF.Eval})$ *is a pseudorandom correlation function (PCF) for* $\mathcal{Y}$*, if the following conditions hold:*

- **(Weakly) pseudorandom** $\mathcal{Y}$**-correlated outputs.** For every non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$, it holds that for all sufficiently large λ,

$$|\Pr[\mathsf{Exp}^{\mathsf{w\text{-}pr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{w\text{-}pr}}_{\mathcal{A},N,1}(\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{Exp}^{\mathsf{w\text{-}pr}}_{\mathcal{A},N,b}$ ($b \in \{0,1\}$) is defined as in Figure 2.3. In particular, the adversary is given access to $N(\lambda)$ samples.

Figure 2.3: (Weakly) Pseudorandom $\mathcal{Y}$-correlated outputs of a (w)PCF

```
$\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)$ :
  $(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
  For $i = 1, \ldots, N(\lambda)$ :
      $x^{(i)} \leftarrow\!\$ \{0,1\}^{n(\lambda)}$
      $(y_0^{(i)}, y_1^{(i)}) \hookleftarrow \mathcal{Y}(1^\lambda)$
  $b \leftarrow \mathcal{A}(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i \in [N(\lambda)]})$
  Output $b$
```

```
$\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)$ :
  $(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
  For $i = 1, \ldots, N(\lambda)$ :
      $x^{(i)} \leftarrow\!\$ \{0,1\}^{n(\lambda)}$
      For $\sigma \in \{0,1\}$:
        $y_\sigma^{(i)} \leftarrow\!\$ \mathsf{wPCF.Eval}(\sigma, k_\sigma, x^{(i)})$
  $b \leftarrow \mathcal{A}(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i \in [N(\lambda)]})$
  Output $b$
```

Figure 2.4: Security of a wPCF

```
$\underline{\mathsf{Exp}^{\text{w-sec}}_{\mathcal{A},N,\sigma,0}(\lambda)}$ :
    $(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
    For $i = 1, \ldots, N(\lambda)$ :
        $x^{(i)} \leftarrow\$ \{0,1\}^{n(\lambda)}$
        $y^{(i)}_\sigma \leftarrow\$ \mathsf{wPCF.Eval}(\sigma, k_\sigma, x^{(i)})$
        $y^{(i)}_{1-\sigma} \leftarrow\$ \mathsf{RSample}(1^\lambda, \sigma, y^{(i)}_\sigma)$
    $b \leftarrow \mathcal{A}(1^\lambda, \sigma, k_\sigma, (x^{(i)}, y^{(i)}_{1-\sigma})_{i \in [N(\lambda)]})$
    Output $b$
```

```
$\underline{\mathsf{Exp}^{\text{w-sec}}_{\mathcal{A},N,\sigma,1}(\lambda)}$ :
    $(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
    For $i = 1, \ldots, N(\lambda)$ :
        $x^{(i)} \leftarrow\$ \{0,1\}^{n(\lambda)}$
        $y^{(i)}_{1-\sigma} \leftarrow\$ \mathsf{wPCF.Eval}(1-\sigma, k_{1-\sigma}, x^{(i)})$
    $b \leftarrow \mathcal{A}(1^\lambda, \sigma, k_\sigma, (x^{(i)}, y^{(i)}_{1-\sigma})_{i \in [N(\lambda)]})$
    Output $b$
```

- **Security.** For every $\sigma \in \{0,1\}$ and every non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$, it holds that for all sufficiently large λ,

$$|\Pr[\mathsf{Exp}^{\text{w-sec}}_{\mathcal{A},N,\sigma,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{w-sec}}_{\mathcal{A},N,\sigma,1}(\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

  where $\mathsf{Exp}^{\text{w-sec}}_{\mathcal{A},N,\sigma,b}$ ($b \in \{0,1\}$) is defined as in Figure 2.4 where RSample is the algorithm for reverse sampling $\mathcal{Y}$ as in Definition 2.7.1. In particular, the adversary is given access to $N(\lambda)$ samples (or simply N if there is no ambiguity).

2.7.2 Strong Pseudorandom Correlation Functions

A strong PCF is syntactically defined in the same way as a weak PCF, but it instead satisfies stronger notions of *pseudorandom $\mathcal{Y}$-correlated outputs* and *PCF security*. For simplicity, we only provide these modified properties.

We say that $(\mathsf{sPCF.Gen}, \mathsf{sPCF.Eval})$ is an (N, B, ϵ)-secure strong pseudorandom correlation function (sPCF) for $\mathcal{Y}$, if the following conditions hold:

- **Strongly pseudorandom $\mathcal{Y}$-correlated outputs.** For every non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$ asking at most $N(\lambda)$ queries to the oracle $\mathcal{O}_b(\cdot)$ (as defined in Figure 2.5), it holds that for all sufficiently large λ,

$$|\Pr[\mathsf{Exp}^{\text{s-pr}}_{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{s-pr}}_{\mathcal{A},1}(\lambda) = 1]| \leq \epsilon(\lambda)$$

  where $\mathsf{Exp}^{\text{s-pr}}_{\mathcal{A},b}$ ($b \in \{0,1\}$) is defined as in Figure 2.5.

Figure 2.5: Strongly Pseudorandom $\mathcal{Y}$-correlated outputs of a sPCF

```
$\underline{\mathsf{Exp}^{\text{s-pr}}_{\mathcal{A},b}(\lambda)}$ :
    $(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$, $\mathcal{Q} \leftarrow \varnothing$
    $b \leftarrow\!\!\$\; \mathcal{A}^{\mathcal{O}_b(\cdot)}(1^\lambda)$
    **Output** $b$

$\underline{\mathcal{O}_0(x)}$ :
    **If** $(x, y_0, y_1) \in \mathcal{Q}$,
      **Else** $(y_0, y_1) \leftarrow\!\!\$\; \mathcal{Y}(1^\lambda)$,
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(x, y_0, y_1)\}$
      **Output** $(y_0, y_1)$

$\underline{\mathcal{O}_1(x)}$ :
    **For** $\sigma \in \{0,1\}$:
      $y_\sigma \leftarrow \mathsf{sPCF.Eval}(1^\lambda, \sigma, k_\sigma, x)$
      **Output** $(y_0, y_1)$
```

- **Strong Security.** For every $\sigma \in \{0,1\}$ and every non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$ asking at most $N(\lambda)$ queries to the oracle $\mathcal{O}_b(\cdot)$ (as defined in Figure 2.6) and RSample is the algorithm for reverse sampling $\mathcal{Y}$ as in Definition 2.7.1, it holds that for all sufficiently large λ,

$$|\Pr[\mathsf{Exp}^{\text{s-sec}}_{\mathcal{A},0,\sigma}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{s-sec}}_{\mathcal{A},1,\sigma}(\lambda) = 1]| \leq \epsilon(\lambda)$$

where $\mathsf{Exp}^{\text{s-sec}}_{\mathcal{A},\sigma}$ is defined as in Figure 2.6.

Figure 2.6: Strong PCF Security

```
$\underline{\mathsf{Exp}^{\text{s-sec}}_{\mathcal{A},b,\sigma}(\lambda)}$ :
    $(k_0, k_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$, $\mathcal{Q} \leftarrow \varnothing$
    $b \leftarrow\!\!\$\; \mathcal{A}^{\mathcal{O}_b(\cdot)}(1^\lambda, \sigma, k_\sigma)$
    **Output** $b$

$\underline{\mathcal{O}_0(x)}$ :
    $y_{1-\sigma} \leftarrow \mathsf{sPCF.Eval}(1-\sigma, k_{1-\sigma}, x)$
    **Output** $y_{1-\sigma}$

$\underline{\mathcal{O}_1(x)}$ :
    $y_\sigma \leftarrow \mathsf{sPCF.Eval}(\sigma, k_\sigma, x)$
    $y_{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_\sigma)$
    **Return** $y_{1-\sigma}$
```

2.8 Designated-Verifier Zero-Knowledge Proofs

Let $\mathcal{R}$ be an efficiently decidable binary relation for an NP language $\mathcal{L}$. If $x \in \mathcal{L}$ and $(x, w) \in \mathcal{R}$ then x is a statement and w is a witness. An interactive/non-interactive argument for $\mathcal{R}$ is a tuple of three probabilistic polynomial time interactive/non-interactive algorithms $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ called the common reference string generator, the prover, and the verifier. with the following properties:

- $(\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Gen}(1^\lambda)$. On input 1^λ generates public parameters *par* (such as group parameters), a crs, and a trapdoor $\mathcal{T}$. For simplicity of notation, we assume that any group parameters are implicitly included in the crs.

If the trapdoor $\mathcal{T}$ of the non-interactive proof system is set to $\perp$ (or, alternatively, if it is included in the crs), we call the argument system *publicly verifiable*. Otherwise, we call it a *designated-verifier* interactive/non-interactive argument system, i.e., Privately Verifiable ZKP for interactive proof and DV-NIZK for non-interactive one.

2.8.1 Privately Verifiable ZKPs

A privately verifiable interactive argument for $\mathcal{R}$ is a tuple of three probabilistic polynomial time interactive algorithms $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ called the common reference string generator, the prover and the verifier (Gen is defined as above) with the following properties:

- We write $tr \leftarrow \langle \mathsf{P}(x), \mathsf{V}(y)\rangle$ for the public transcript produced by P and V when interacting on inputs x and y. This transcript ends with V either accepting or rejecting. We sometimes shorten the notation by saying $\langle \mathsf{P}(x), \mathsf{V}(y)\rangle = b$, where $b = 0$ corresponds to V rejecting and $b = 1$ corresponds to V accepting.

Definition 2.8.1 (Perfect completeness). *A proof system* $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *for* $\mathcal{R}$ *is perfectly complete, if*

$$\Pr\left[\langle \mathsf{P}(\mathsf{crs}, x, w), \mathsf{V}(\mathsf{crs}, \mathcal{T}, x)\rangle = 1 \,\middle|\, \begin{array}{c} (\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (x, w) \in \mathcal{R} \end{array}\right] = 1$$

Definition 2.8.2 (Computational soundness). *A proof system* Π *is computationally sound if for every efficient adversary* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} \langle \mathcal{A}, \mathsf{V}(\mathsf{crs}, \mathcal{T}, x)\rangle = 1 \\ x \notin \mathcal{L} \end{array} \,\middle|\, \begin{array}{c} (\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Gen}(1^\lambda) \\ x \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \end{array}\right] = \mathsf{negl}(\lambda)$$

An argument $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ is public coin if the verifier's messages are chosen uniformly at random independently of the messages sent by the prover, i.e, the challenges correspond to the verifier's randomness ρ.

Definition 2.8.3 (Special honest-verifier zero-knowledge (SHVZK)). *A public coin argument* Π *is an SHVZK if there exists a probabilistic polynomial time simulator* Sim *such that for all non-uniform polynomial time adversaries* $\mathcal{A}$ *we have*

$$\begin{aligned} &\Pr\left[\begin{array}{c} \mathcal{A}(tr) = 1 \\ (x, w) \in \mathcal{R} \end{array} \,\middle|\, \begin{array}{c} (\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (x, w, \rho) \leftarrow \mathcal{A}(\mathsf{crs}); tr \leftarrow \langle \mathsf{P}(\mathsf{crs}, x, w), \mathsf{V}(\mathsf{crs}, \mathcal{T}, x)\rangle \end{array}\right] \\ \approx &\Pr\left[\begin{array}{c} \mathcal{A}(tr) = 1 \\ (x, w) \in \mathcal{R} \end{array} \,\middle|\, \begin{array}{c} (\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (x, w, \rho) \leftarrow \mathcal{A}(\mathsf{crs}); tr \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathcal{T}, x, \rho) \end{array}\right] \end{aligned}$$

where ρ *is the public coin randomness used by the verifier.*

2.8.2 Non-Interactive Zero-Knowledge Proofs (NIZKs)

A non-interactive argument for $\mathcal{R}$ is a tuple of three probabilistic polynomial time interactive algorithms $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ (Setup is defined as above) called the common reference string generator, the prover, and the verifier with the following properties:

- $\mathsf{P}(\mathsf{crs}, x, w)$. On input of a crs, a statement x with witness w, outputs a proof π for $x \in \mathcal{L}$.
- $\mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T})$. On input of a crs, a statement, a proof, and a trapdoor, accepts or rejects the proof.

which satisfies the completeness, soundness, and zero-knowledge properties defined below.

If the trapdoor $\mathcal{T}$ of the non-interactive proof system is set to $\perp$ (or, alternatively, if it is included in the crs), we call the argument system *publicly verifiable.* Otherwise, we call it a *designated-verifier* non-interactive argument system. If the soundness guarantee holds with respect to a computationally unbounded adversary, we have a NIZK-proof system.

Definition 2.8.4 (Perfect completeness). *A proof system* $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ *for* $\mathcal{R}$ *is perfectly complete, if*

$$\Pr\left[\mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T}) = 1 \;\middle|\; \begin{array}{c} (\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x, w) \in \mathcal{R}\ ,\ \pi \overset{\$}{\leftarrow} \mathsf{P}(\mathsf{crs}, x, w) \end{array}\right] = 1$$

The soundness notion can be divided into *non-adaptive* and *adaptive*; it is non-adaptive if the malicious prover needs to choose the statement x before generating the crs while it is adaptive if the adversary can dynamically choose the statement after generating crs. We consider a strong variant of adaptive soundness, denoted *unbounded adaptive* soundness, where the adversary is given oracle access to a verification oracle. Note that in the publicly-verifiable setting, this is equivalent to the standard soundness notion (computational soundness), where the adversary must forge valid proof on an incorrect statement without the help of any oracle. However, in the designated-verifier setting, the standard soundness notion only guarantees that the argument system remains sound as long as the prover receives at most logarithmically responses on previous proofs. On the other hand, if the argument system satisfies unbounded soundness (*reusable soundness*), its soundness is maintained even if the adversary receives an arbitrary (polynomial) number of responses on previous proofs.

Definition 2.8.5 (Unbounded adaptive soundness). *A proof system* Π *is unbounded adaptive soundness if for every PPT adversary* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} \mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T}) = 1 \\ x \notin \mathcal{L} \end{array} \;\middle|\; \begin{array}{c} (\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}(\mathsf{crs},.,.,\mathcal{T})}(\mathsf{crs}) \end{array}\right] = \mathsf{negl}(\lambda)$$

where $\mathcal{A}$ *can make polynomially many queries to an oracle* $\mathcal{O}(\mathsf{crs}, ., ., \mathcal{T})$ *which, on input* (x, π)*, outputs* $\mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T})$.

Knowledge extractability (soundness) is a strengthening of the soundness property which guarantees that if the prover produces an accepting proof then there exists an efficient simulator can actually *extract* a witness for the statement. So the extractor is defined by $\mathsf{Ext}(\pi, x, \mathcal{T}) \to w$ where $(x, w) \in \mathcal{R}$.

Definition 2.8.6 (Unbounded adaptive knowledge soundness). *A proof system* Π *is unbounded adaptive knowledge extractability if for every PPT adversary* $\mathcal{A}$*, there exists an efficient extractor* Ext *such that*

$$\Pr\left[\begin{array}{c} (\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}(\mathsf{crs},.,.,\mathcal{T})}(\mathsf{crs}) \\ w \leftarrow \mathsf{Ext}(\pi, x, \mathcal{T}) \end{array} : (x, w) \in \mathcal{R} \textit{ iff } \mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T}) = 1\right] \approx 1$$

where $\mathcal{A}$ can make polynomially many queries to an oracle $\mathcal{O}(\mathsf{crs}, ., ., \mathcal{T})$ which, on input (x, π), outputs $\mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T})$.

We consider the notion of *adaptive* zero-knowledge where the adversary can choose the statement after seeing the crs. The definition of ZK below is often referred to as "single-theorem ZK" in which the prover generates a single proof (and the length of the common reference string can be larger than the length of the statement to prove) and multi-theorem zero-knowledge (where the adversary can adaptively ask for polynomially many proofs on arbitrary pairs (x, w) for the same common reference string). Note that, there is a generic compiler from single-theorem ZK to multi-theorem ZK where zero-knowledge holds polynomially many statements via the "OR trick". The same transformation directly applies to both the selective and adaptive ZK setting and also both the publicly verifiable and the designated verifier setting.

Definition 2.8.7 (Adaptive Zero-Knowledge)**.** *We say a non-interactive argument* Π *is Adaptive Single-Theorem (Multi-Theorem) Zero-Knowledge if there exists a polynomial time simulator* $\mathsf{SimProver} = (S_1, S_2)$ *where* $(\mathsf{crs}, \mathcal{T}) \leftarrow S_1(1^\lambda)$ *outputs a simulated common reference string and a simulation trapdoor and* $\pi \leftarrow S_2(\mathsf{crs}, \mathcal{T}, x)$ *produces a simulated argument such that*

Adaptive Single-Theorem Zero-Knowledge. For all interactive adversaries PPT $\mathcal{A}$, we require

$$\Pr\left[\begin{array}{c|c} \mathcal{A}(\pi) = 1 & (\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (x, w) \in \mathcal{R} & (x, w) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ & \pi \overset{\$}{\leftarrow} \mathsf{P}(\mathsf{crs}, x, w) \end{array}\right] - \Pr\left[\begin{array}{c|c} \mathcal{A}(\pi) = 1 & (\mathsf{crs}, \mathcal{T}) \leftarrow S_1(1^\lambda) \\ (x, w) \in \mathcal{R} & (x, w) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ & \pi \leftarrow S_2(\mathsf{crs}, \mathcal{T}, x) \end{array}\right] = \mathsf{negl}(\lambda)$$

Adaptive Multi-Theorem Zero-Knowledge. A PPT $\mathcal{A}$ has a negligible advantage in distinguishing the experiments $\mathsf{Exp}^{\mathsf{zk},0}_{\mathcal{A}}(1^\lambda)$ and $\mathsf{Exp}^{\mathsf{zk},1}_{\mathcal{A}}(1^\lambda)$ given in Figure 2.8.

Figure 2.7: $\mathsf{Exp}^{\mathsf{zk},0}_{\mathcal{A}}(1^\lambda)$ and $\mathsf{Exp}^{\mathsf{zk},1}_{\mathcal{A}}(1^\lambda)$

```
$\mathsf{Exp}^{\mathsf{zk},0}_{\mathcal{A}}(1^\lambda)$ :
    $(\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Setup}(1^\lambda)$
    **Return** $b \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{prove}}(\mathsf{crs}, ., .)}(\mathsf{crs})$

$\mathcal{O}_{\mathsf{prove}}(\mathsf{crs}, x, w)$
    If $(x, w) \in \mathcal{R}$ then
    **Return** $\pi \leftarrow \mathsf{P}(\mathsf{crs}, x, w)$
    else **Return** $\perp$
    end if
```

```
$\mathsf{Exp}^{\mathsf{zk},1}_{\mathcal{A}}(1^\lambda)$ :
    $(\mathsf{crs}, \mathcal{T}) \leftarrow S_1(1^\lambda)$
    **Return** $b \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sim}}(\mathsf{crs}, \mathcal{T}, ., .)}(\mathsf{crs})$

$\mathcal{O}_{\mathsf{sim}}(\mathsf{crs}, \mathcal{T}, x, w)$
    If $(x, w) \in \mathcal{R}$ then
        **Return** $\pi \leftarrow S_2(\mathsf{crs}, \mathcal{T}, x)$
    else **Return** $\perp$
    end if
```

Figure 2.8: Distinghuish games between outputs of oracles $\mathcal{O}_{\mathsf{prove}}(\mathsf{crs}, x, w)$ and $\mathcal{O}_{\mathsf{sim}}(\mathsf{crs}, \mathcal{T}, x, w)$, for the (adaptive) multi-theorem zero-knowledge property of a non-interactive argument system. $\mathcal{A}$ outputs $b \in \{0, 1\}$.

2.9 Ideal Functionalities

2.9.1 Ideal Functionality of PSI

The ideal functionality of PSI in the semi-honest and malicious settings are shown on Figure 2.9. In the semi-honest setting, the size of a corrupted party's input set is fixed to n, while in the malicious setting, the functionality allows a malicious party to use a possibly larger set of bounded size $n' > n$. Since PSI is a special case of secure computation, the security analysis of a two-party PSI protocol is performed via the standard simulation paradigm.

Figure 2.9: $\mathcal{F}_{\mathsf{psi}}$ in the malicious setting

PARAMETERS:

- An arbitrary field $\mathbb{F}, n, n_X, n_Y$ public parameters for honest parties and corrupt parties respectively where $n \leq n_X, n_Y$.
- There are two parties, a sender with a input set $X \subseteq \mathbb{F}$ and a receiver with a input set $Y \subseteq \mathbb{F}$.

FUNCTIONALITY:

- Wait for sender to send the input set (InputS, X), if $|X| > n_X$ and the sender is malicious or if $|X| \neq n$ and the sender is honest then abort.
- Wait for receiver to send the input set (InputR, Y). If $|Y| > n_Y$ and the receiver is malicious or $|Y| \neq n$ and the receiver is honest, abort.
- The functionality outputs the intersection $(\mathsf{OutputR}, X \cap Y)$ to receiver.

2.9.2 Ideal Functionalities for Interactive ZKPs

To establish the security of interactive zero-knowledge (ZK) proofs within the Universal Composability (UC) framework, we define the zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}$ in Figure 2.10 and the commitment functionality $\mathcal{F}_{\mathsf{Com}}$ in Figure 2.11.

Figure 2.10: $\mathcal{F}_{\mathsf{ZK}}$ Ideal Functionality for Interactive Zero-Knowledge Proofs

PARAMETERS:

- Let $\mathcal{C}$ be a publicly known circuit representing a statement to be proven. The protocol involves a prover P, who possesses a witness $\boldsymbol{w}$, and a verifier V.

FUNCTIONALITY:

- Upon receiving $(\mathsf{prove}, \boldsymbol{w})$ from P and (verify) from V:
  - If $\mathcal{C}(\boldsymbol{w}) = 0$, output (true) to V.
  - Otherwise, output (false) to V.

Figure 2.11: $\mathcal{F}_{\mathsf{Com}}$ Ideal Functionality for Commitment

PARAMETERS:

- Let $\mathcal{C}$ be a publicly known circuit. The protocol involves a prover P, who holds a witness $\boldsymbol{w}$, and a verifier V.

FUNCTIONALITY:

- Upon receiving $(\mathsf{Commit}, \boldsymbol{w})$ from P and (Commit) from V:
    - Generate a unique tag $[\![\boldsymbol{w}]\!]$ and store the tuple $([\![\boldsymbol{w}]\!], \boldsymbol{w})$.
    - Return $[\![\boldsymbol{w}]\!]$ to both parties.
- Upon receiving $(\mathsf{Open}, [\![\boldsymbol{w}]\!])$:
    - If a tuple $([\![\boldsymbol{w}]\!], \boldsymbol{w})$ was previously stored, output $([\![\boldsymbol{w}]\!], \boldsymbol{w})$ to V.
    - Otherwise, abort.

Chapter 3

Private Set Intersection

In this chapter, we present our first contribution about constructing private set intersection (PSI) schemes based on pseudorandom correlation generators (PCGs). We describe several constructions:

- Semi-Honest Hash-Based PSI Protocols. New protocols using various hashing techniques achieve very low communication, especially for small database entries.
- Maliciously Secure Hash-Based PSI Protocols. A revival of the dual execution technique, previously considered outdated, leads to protocols with lower communication than existing alternatives for small database entries.
- Maliciously Secure Polynomial-Based PSI Protocol. Introduces a protocol with competitive communication, security under the standard model (ring-LPN assumption), and efficient client-server interactions, allowing the client to reuse a single encoding for intersections with multiple servers.

We provide a detailed technical overview of our contributions in Section 3.3. Section 3.4 covers our ROM-based semi-honest and malicious protocols. Section 3.5 covers our standard model PSI. This chapter appears in the full version [BC23].

Contents

3.1 Motivations and Related Works

Private Set Intersection (PSI) is a cryptographic primitive that allows parties to jointly compute the set of all common elements between their datasets, without leaking any value outside of the intersection. It is a special case of secure multi-party computation (MPC). PSI enjoys a wide array of real-life applications; it is perhaps the most actively researched concrete functionality in secure computation, and has been the target of a tremendous number of works, see [PSZ14; PSS+15; KKR+16; RR17; KRT+19; PSW+18; PRT+19; PRT+20; CM20; RS21; GPR+21; RT21a] and references therein for a sample. As a consequence of this intense research effort, modern PSI protocols now achieve impressive efficiency features, communicating only a few hundred bits per database items, and processing millions of items in seconds.

Paradigms for PSI

Many approaches to PSI have been designed over the year. While some early proposals relied on generic secure computation methods [HEK12], the most efficient solutions to date rely on specialized techniques. While this fails to capture all proposals, the most prominent modern PSI protocols can be categorized in four groups.

Key exchange-based. Early PSI protocols relied on techniques based on the Diffie-Hellman key-exchange [HFH99], and were extended to the malicious setting in [DKT10; JL10]. While these protocols have relatively low communication, the need to compute multiple exponentiations for each item gives them relatively poor performances. Nevertheless, recent improvements to these old protocols [RT21a] show that this approach can be somewhat competitive for very small sets, when communication is a very scarce resource.

Hashing-based. Cuckoo hashing [PR04] is an efficient hashing technique where items are hashed into a linear number of bins while guaranteeing that no bin will contain more than one item. For most of the past decades, the top-performing PSI protocols [PSZ14; PSS+15; KKR+16; HV17; RR17; CLR17; OOS17; PSW+18; PSZ18] relied on Cuckoo hashing techniques, or variants thereof [PRT+19], usually combined with fast oblivious transfer extension [IKN+03]. While the more recent OKVS-based protocols now outperform hash-based protocols, the protocol of [KKR+16] remains among the most competitive PSI protocols.

OKVS-based. Starting with the work of [PRT+19], a new approach to PSI has been designed, which relies on a carefully chosen encoding of the datasets with oblivious data structures. The first protocols required expensive polynomial interpolation [PRT+19; CM20], but a more efficient data structure called PaXoS was introduced in [PRT+20], which significantly changed the state of affairs, and led to the most efficient PSI protocol known to date [RS21]. The required data structures have been recently abstracted out in [GPR+21] under the name *oblivious key-value stores* (OKVS), and the work introduced significant improvements in the efficiency and security guarantees of OKVS constructions. As a consequence, a combination of [RS21] with the latest OKVS of [GPR+21] leads to a very low-communication PSI protocol (for mid to large databases), still with a very good concrete efficiency.

Polynomial manipulation. Eventually, starting with the work of [FNP04], several protocols have been designed that represent the datasets as polynomials over finite fields, and compute set operations through operations on these polynomials. Notable works include [KS05; DMR+09; HN10; Haz15; FHN+16; GS19; GN19]. While these protocols are usually much slower than those based on Cuckoo hashing or OKVS, they offer some advantages, such as realizing stronger functionalities (like threshold private set intersection [GS19]) and providing security in the standard model (in contrast, the most efficient OKVS- or hash-based protocols require the random oracle model, or some ad hoc family of correlation-robustness assumptions). On the downside, achieving malicious security with these protocols can be subtle, since one must handle the case where the parties input zero polynomials. In fact, previous protocols [GN19] which claimed to be secure against malicious adversaries in this setting have later been broken [AMZ21].

Improving PSI with pseudorandom correlation generators

Pseudorandom correlation generators (PCG) have been introduced in the works of [BCG+17; BCG+18; BCG+19b] and have been the subject of a long and fruitful line of work [BCG+17; BCG+18; BCG+19b; BCG+19a; SGR+19; BCG+20b; BCG+20a; YWL+20; CRR21; WYK+21]. At a high level, a PCG allows two parties to securely stretch long pseudorandom correlated strings from short, correlated seeds. Securely sharing correlated random strings is a crucial component in most modern secure computation protocols, which operate in the preprocessing model; PCG allows to realize this functionality with almost no communication. Among their many applications, PCGs allow to construct *silent oblivious transfer extension* protocols [BCG+19a], which can realize (pseudorandom) OT extension with minimal (logarithmic) communication.

Since the top-performing PSI protocols rely on efficient OT extension, using PCG-based techniques to improve their efficiency is a natural idea. And indeed, this was done recently for OKVS-based PSI in [RS21], leading to the most efficient PSI protocol known to date (OKVS stands for oblivious key-value store [GPR+21]; the use of OKVS is the leading paradigm for the design of PSI protocols). To give a single datapoint, computing the intersection between two databases of size $n = 2^{20}$ with the protocol of [RS21] communicates as little as $426n$ bits in total. In addition, some of the tools used in [RS21] have been significantly improved since: replacing their OKVS (which is the PaXoS OKVS of [PRT+20]) by the more recent 3H-GCT OKVS of [GPR+21], and replacing their PCG (which is the one from [WYK+21]) by the recent PCG of [CRR21], the cost goes down to an impressive $247n$ bits of total communication. In comparison, even the *insecure* approach of exchanging the hashes of all items in the databases already requires $160n$ bits of communication. OKVS-based PSI protocols are now firmly established as the leading paradigm in the field, and the use of PCGs to reduce their communication overhead even more seems to further widen the gap with the other paradigms.

Related Work

At the time of writting this work, in a concurrent and independent work, recently accepted at CCS'22, Rindal and Raghuraman [RR22] introduced a new PSI protocol, using an approach similar to ours: the authors also leveraged subfield-VOLE to achieve communication independent of the computational security parameter κ. Our results have been obtained independently of theirs, around the same time period. Although their main result bears similarities to our first two contributions, we highlight some important distinctions between our work and theirs:

- The work of [RR22] uses an OKVS-based construction, and achieves a receiver-to-sender communication of $(\lambda + 2\log n) \cdot n$. In contrast, we use a hash-based protocol, and achieve an $(\ell - \log n) \cdot n$ receiver-to-sender communication. Therefore, we get smaller communication overall in the setting where the databases have small entries, but a slightly larger computation.

- For malicious security, the work of [RR22] only considers the standard paradigm of previous works (e.g. [RS21]), hence having a $O(\kappa \cdot n)$ receiver-to-sender (and overall) communication. In contrast, we give two protocols, including one based on dual execution which achieves communication independent of κ (and smaller concrete communication for databases with small entries).

- Eventually, our last contribution, a "batchable" ring-OLE-based malicious PSI in the standard model with low communication, is unique to our work.

3.2 Detailed Contributions

In this chapter, we thoroughly investigate how the use pseudorandom correlation generators (PCGs) can reduce communication in PSI protocols. We obtain several contributions:

- A new family of semi-honest hash-based PSI protocols. Our protocols can be instantiated using several hashing techniques, and achieve very low communication, especially for databases whose entries have a small bitlength.

- New maliciously secure hash-based PSI protocols. Here, interestingly, we revive the dual execution technique, which had been used previously to design malicious PSI protocols in [RR17], but was considered outdated. We show that, combined with our new approach, it leads to very competitive protocols, which achieve lower communication than all known alternatives for databases with small entries.

- Eventually, we design a new maliciously secure polynomial-based PSI protocol. Our protocol enjoys several powerful features: competitive communication, security in the standard model under the ring-LPN assumption (in contrast, other maliciously secure PSI use the ROM), and the possibility for a client to publish a single encoding of its database, and later retrieve the intersection of its database with that of multiple servers independently, with a single server-to-client message, plus minimal (database-independent) additional communication.

Low communication PSI for databases with small entries. Modern PSI protocols have communication $O(\lambda \cdot n)$, where n is the database size, and λ is a computational security parameter. More precisely, the receiver-to-sender communication is $O(\lambda \dot{n})$, while the sender-to-receiver communication is $O(\kappa \cdot n)$, where κ is a *statistical* security parameter (typically, $\lambda = 128$ and $\kappa = 40$). We introduce a new protocol, that combines hashing techniques (e.g. Cuckoo hashing or its variants, as initially used in [KKR+16]) with a new PCG-based oblivious pseudorandom function (OPRF). In contrast to all previous works, our work avoids the $O(\lambda \cdot n)$ overhead: it reduces the receiver-to-sender communication to be roughly $\ell \cdot n$ (where ℓ is the bitsize of the database items), leading to a significant reduction in the overall communication. To our knowledge, our protocol is the first to achieve communication independent of λ (up to low order terms). To give a datapoint, for $n = 2^{20}$, with 64-bit entries, our protocol communicates $210n$ bits, and with 32-bit entries, it communicates only $148n$ bits. For the same parameters, the leading OKVS-based PSI of [RS21] communicates $197n$ bits, even after improving it with all relevant optimization (such as using the 3H-GCT OKVS of [GPR+21], and the recent PCG of [CRR21]). We provide further datapoints and comparisons to the state of the art on Table 3.1, when instantiating our protocols with various hashing methods. For all other protocols, we take into account the optimization of [TLP+17] which reduces the costs of sending n elements of bitlength $\lambda + 2 \cdot \log n$ to $n \cdot (\lambda + \log n)$. GCH stands for Generalized Cuckoo hashing (here, with 2 hash functions and 3 items per bin), 2CH for 2-choice hashing, and SH for simple hashing (N is the number of bins).

Fast maliciously-secure PSI for small entries. We then turn our attention to maliciously secure PSI. We provide two alternative protocols that achieve malicious security; both use standard paradigms

Table 3.1: Comparison of the communication cost of several PSI protocols in the semi-honest setting and in the malicious setting, for various choices of the database size n (we assume that both parties have a database of the same size). ℓ denote the bit-length of the inputs in the database; we set the statistical security parameter κ to 40 (for usual applications) or 30 (which can be suitable for lower risk applications).

	$n = 2^{14}$	$n = 2^{16}$	$n = 2^{20}$	$n = 2^{24}$
Semi-honest setting				
KKRT16[KKR+16]	$930n$	$936n$	$948n$	$960n$
PRTY19[PRT+19] low[*]	$491n$	$493n$	$493n$	$494n$
PRTY19[PRT+19] fast[*]	$560n$	$571n$	$579n$	$587n$
CM20[CM20]	$668n$	$662n$	$674n$	$676n$
PRTY20[PRT+20]	$1244n$	$1192n$	$1248n$	$1278n$
RS21[RS21]	$2024n$	$898n$	$406n$	$374n$
RS21[RS21] enhanced[**]	$280n$	$260n$	$263n$	$275n$
Ours ($\ell = 64$, GCH)	$246n$	$220n$	$210n$	$209n$
Ours ($\ell = 48$, GCH)	$215n$	$189n$	$179n$	$178n$
Ours ($\ell = 32$, GCH)	$184n$	$158n$	$148n$	$147n$
Ours ($\ell = 64$, 2CH)	$214n$	$190n$	$183n$	$185n$
Ours ($\ell = 48$, 2CH)	$193n$	$169n$	$162n$	$164n$
Ours ($\ell = 32$, 2CH)	$171n$	$148n$	$141n$	$142n$
Ours ($\ell = 64$, SH, $N = n/10$)	$332n$	$302n$	$284n$	$276n$
Ours ($\ell = 48$, SH, $N = n/10$)	$261n$	$230n$	$209n$	$198n$
Ours ($\ell = 32$, SH, $N = n/10$)	$191n$	$158n$	$133n$	$120n$
Ours ($\ell = 64$, SH, $N = 1$)[***]	$154n$	$131n$	$125n$	$128n$
Ours ($\ell = 48$, SH, $N = 1$)[***]	$138n$	$115n$	$109n$	$112n$
Ours ($\ell = 32$, SH, $N = 1$)[***]	$122n$	$99n$	$93n$	$96n$
Malicious setting				
RS21[RS21] enhanced[**]	$343n$	$320n$	$315n$	$318n$
Ours ($\ell = 48$, SH, $N = n/10$)	$430n$	$393n$	$356n$	$332n$
Ours ($\ell = 40$, SH, $N = n/10$)	$359n$	$321n$	$281n$	$253n$
Ours ($\ell = 32$, SH, $N = n/10$)	$289n$	$249n$	$205n$	$175n$

[*] PRTY19 has two variants, SpOT-low (lowest communication, higher computation) and SpOT-fast (higher communication, better computation). Both use expensive polynomial interpolation and require significantly more computation compared to all other protocols in this table.

[**] Using the 3H-GCT OKVS of [GPR+21] instead of PaXoS, and the VOLE of [CRR21] instead of the one from [WYK+21].

[***] Using $N = 1$ requires an expensive degree-n polynomial interpolation.

for upgrading PSI to malicious security. The first protocol combines our new PCG-based OPRF with simple hashing, and applies the standard paradigm used in most previous OKVS-based PSI to achieve malicious security (e.g. [RS21]). This requires increasing the sender-to-receiver message length, from $O(\kappa \cdot n)$ to $O(\lambda \cdot n)$ (κ is a statistical security parameter, λ is a computational security parameter; typically, $\kappa = 40$ and $\lambda = 128$) to allow for extraction of the sender input.

More interestingly, our second protocol applies *dual execution* [RR17] to our PCG-based protocol with simple hashing. We observe that, in our context, this allows to achieve malicious security without having to increase the length of the sender-to-receiver message, at the cost of increasing the receiver-to-sender communication by a factor 2. Since our approach makes this communication as low as $O(\ell \cdot n)$, this turns out to be an excellent tradeoff whenever the database entries are not too large. Therefore, our results show that the landscape of maliciously secure PSI is more subtle than previously thought: for large entries, the standard approach still dominates, but for smaller entries (e.g. $\ell \leq 40$), the dual execution technique leads to better performances. This revives the dual execution technique, which was previously considered obsolete compared to the modern alternatives.

Efficient PSI in the standard model. Eventually, our last contribution is a new “polynomial-based” PSI protocol that does not rely on the random oracle model, following the high level structure of previous works [KS05; GS19; GN19]. To this end, we introduce the notion of PCG for the *subfield ring-OLE* correlation, and show how a simple variant of the recent PCG for ring-OLE of [BCG+20b] leads to efficient instantiations of this primitive. Then, we describe a new PSI protocol built on top of this PCG, which enjoys a number of very interesting features.

Security features. Our PSI protocol is in the standard model: unlike our first protocol, it does not require the random oracle model, or any tailor-made correlation-robustness assumptions. We rely solely on the (relatively well-established) ring-LPN assumption over polynomial rings with irreducible polynomials. To our knowledge, our protocol is the first standard model protocol which offers competitive performances compared to protocols using the random oracle heuristic or tailored assumptions. Furthermore, our PSI protocol enjoys full malicious security (for both parties) *almost for free*. This stems from the use of PCGs, which allows to confine the “price” of achieving malicious security to the distributed seed generation only, which has logarithmic communication and computation (in the set size n).

We note that, though malicious security comes for free communication- and computation-wise, the tweaks used to guarantee malicious security in our protocol are not straightforward. In fact, achieving malicious security efficiently in polynomial-based PSI protocols is known to be complex and error prone. For example, previous works [GN19] used a superficially similar approach and claimed malicious security, but their protocol was found to be insecure in a recent preprint, which described powerful concrete attacks on this proposal [AMZ21]. Leveraging the specific structure of our protocol, we manage to get around these nontrivial subtleties with careful structural checks, for a minimal cost (independent of the database size).

Efficiency features. Our PSI protocol enjoys a very low communication, considerably lower than all previous PSI protocols in the standard model which we are aware of (excluding iO- or FHE-based protocol, which can have very low communication but poor concrete efficiency). In fact, communication-wise, our PSI protocol is even on par with the best *ROM-based* PSI protocols of previous works. Concretely, for sets of size n with ℓ-bit entries, our protocol communicates $(2\ell + 3\kappa + 3\log n) \cdot n + o(n)$ bits. To give a single datapoint, for $\ell = 32$ and $n = 2^{20}$, we estimate the total communication to be $278n$ bits. This is on par with the best maliciously secure protocol [RS21], which communicates $279n$ bits in the same setting, with comparable computation (it also uses polynomial interpolation), but without standard model security.

On Table 3.2, we compare our protocol to the current fastest maliciously secure PSI protocols [PRT+20; RT21b; RS21], we apply the encoding technique of [TLP+17] to all protocols. For fairness of comparison, since our standard model PSI uses interpolation, we compare it to RS21 with an interpolation-based OKVS (which has better communication), and we compare our other PSIs with RS21 instantiated with (computationally) efficient OKVS. As the table shows, the communication of our protocol is almost on par with that of the best protocol (the protocol of [RS21], enhanced with the latest VOLE protocol) for small-ish input size, and large enough set sizes. Yet, our protocol is in the standard model under the ring-LPN assumption, while [RS21] is only proven secure in the ROM.

Table 3.2: Comparison of the communication cost of several PSI protocols in the malicious model, for various choices of the database size n (we assume that both parties have a database of the same size) and statistical security parameter $\kappa = 40$. ℓ denote the bit-length of the inputs in the database.

Protocol	Communication					Hardness	Standard
	$n = 2^{16}$	$n = 2^{18}$	$n = 2^{20}$	$n = 2^{22}$	$n = 2^{24}$	Assumption	Model
Our Standard PSI						Ring-LPN + OT	✓
$\ell = 64$	$724n$	$423n$	$342n$	$324n$	$323n$		
$\ell = 48$	$692n$	$391n$	$310n$	$292n$	$291n$		
$\ell = 32$	$660n$	$359n$	$278n$	$260n$	$259n$		
RS21[RS21] enhanced	$318n$	$286n$	$279n$	$279n$	$280n$	LPN + OT	✗
Our Direct PSI						LPN + OT	✗
$\ell = 64$	$421n$	$385n$	$374n$	$369n$	$365n$		
$\ell = 48$	$348n$	$311n$	$298n$	$292n$	$286n$		
$\ell = 32$	$277n$	$237n$	$223n$	$215n$	$208n$		
Our Dual PSI							
$\ell = 64$	$609n$	$535n$	$511n$	$499n$	$489n$		
$\ell = 48$	$465n$	$388n$	$361n$	$345n$	$333n$		
$\ell = 32$	$321n$	$240n$	$210n$	$192n$	$176n$		
PRTY20[PRT+20]			$1766n$			OT	✗
RT21[RT21b]			$512n$			DH	✗
RS21[RS21] enhanced	$320n$	$315n$	$315n$	$317n$	$318n$	LPN + OT	✗

Batch non-interactive PSI. On top of these security and efficiency features, the structure of our protocol allows to obtain a powerful interaction pattern: it leads to a batch non-interactive PSI, where after a short interaction with each server, a client C with set X can broadcast a *single* encoding of its database, and receive afterwards at anytime a single message from each server S_i with set X_i (plus, in the malicious setting, a small database-size-independent 2-round structural check), from which they can decode $X \cap X_i$. To achieve this feature, we build upon the fact that the PCG for subfield ring-OLE correlations is *programmable*, which means that we can enforce that a target party will receive the same pseudorandom string across executions with many different parties. Concretely, we achieve the following form of *batch non-interactive PSI* between a client C with database X and multiple servers S_i with datasets X_i (all of size n):

1. In a preprocessing phase, C interacts with each of the servers, using $O(\log n)$ communication *and* computation in each interaction, in a small constant number of rounds.

2. Then, C performs a single $\tilde{O}(n)$ cost local computation, and broadcasts a single $2\ell n$-size *encoding* E_X of X.
3. Each server S_i can, at any time, send a single message $M_i = m(X_i, E_X)$, of length $3(\kappa + \log n)n$, using $\tilde{O}(n)$ computation.
4. Eventually, given X and M_i, the client C can run a $\tilde{O}(n)$ cost decoding procedure and recover $X \cap X_i$, without further interaction.

When the number of servers becomes large, our batch PSI protocol leads to strong savings for the client compared to executing a PSI protocol individually with each server. Furthermore, in this setting, the amortized communication (per PSI instance) is reduced to $(2\ell/N_S + 3\kappa + \log n) \cdot n + o(n)$, where N_S denotes the number of servers. Even for relatively small number of servers, the amortized communication quickly outperforms that of even the best ROM-based maliciously secure PSI protocols. For example, for $n = 2^{24}$ and $\ell = 32$, the amortized communication per secure set intersection approaches $195n$ bits with our protocol, versus $280n$ for [RS21].

3.3 Technical Overview

Notation. We typically write $\mathbb{F}_q$ to denote a field with and arbitrary subfield $\mathbb{F}_p$, where p is a prime power and $q = p^t$. We use $\mathcal{R}_\mathsf{p} = \mathbb{F}_\mathsf{p}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$ for the ring over the field $\mathbb{F}_p$ where $F(x)$ is some polynomial, and also denote $\mathcal{R}_\mathsf{q} = \mathbb{F}_{\mathsf{p}^\mathsf{t}}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$. Note that all operations in our paper are field/ring operations not modular arithmetic.

Our starting point is the classical KKRT protocol [KKR+16], which combines Cuckoo hashing with a batch related-key oblivious pseudorandom function (BaRK-OPRF). We assume some familiarity with the KKRT protocol in this technical overview. For completeness, we provide a high level overview of KKRT, the notion of BaRK-OPRF (batch related-key oblivious pseudorandom function) . Our construction will also rely on a functionality that distributes *subfield vector-OLE* correlation (the sVOLE functionality): Alice gets $(\mathbf{u}, \mathbf{v})$, and Bob gets $(\Delta, \mathbf{w} = \Delta\mathbf{u} + \mathbf{v})$. Such correlation can be distributed with very low communication using pseudorandom correlation generators.

3.3.1 New sVOLE-Based PSI for Databases with Small Entries

Subfield-VOLE leads to a simple and natural construction of BaRK-OPRF. Let ℓ be the bitlength of Alice's inputs, and let $\mathbf{x} = (x_1, \cdots, x_n)$ be the inputs of Alice, viewed as elements of $\mathbb{F}_{2^\ell}$. We assume for simplicity that ℓ divides λ, the computational security parameter. Alice and Bob use an sVOLE protocol (e.g. [CRR21]) over the field $\mathbb{F}_{2^\lambda}$, with subfield $\mathbb{F}_{2^\ell}$; let $(\mathbf{u}, \mathbf{v})$ be the output of Alice, and $(\Delta, \mathbf{w})$ be the output of Bob. Recall that $\mathbf{w} = \Delta \cdot \mathbf{u} + \mathbf{v}$. Alice sends $\mathbf{z} = \mathbf{x} - \mathbf{u}$ to Bob, who defines the BaRK-OPRF keys to be Δ and $(K_1, \cdots, K_n) = \Delta \cdot \mathbf{z} + \mathbf{w}$. The BaRK-OPRF is defined as follows: $F_{\Delta, K_i}(y) = H(i, K_i - \Delta \cdot y)$ (all operations are over $\mathbb{F}_{2^\lambda}$). Eventually, Alice outputs $(H(i, v_i))_{i \le n}$. Observe that

$$\begin{aligned} H(i, v_i) &= H(i, w_i - \Delta u_i) = H(i, K_i - \Delta(z_i + u_i)) \\ &= H(i, K_i - \Delta \cdot x_i) = F_{\Delta, K_i}(x_i) \end{aligned}$$

The use of sVOLE, rather than OT extension as in the original KKRT BaRK-OPRF, has two main advantages: first, the bitwise AND is now replaced by a field multiplication. In particular, this means that we do not need anymore to use error-correcting codes, and that $y \cdot \Delta$ retains the entire entropy of Δ. In other words, it suffices for Δ to be λ-bit long to achieve λ bits of security for the construction (in contrast, KKRT had to use around 5λ bits). Second, and most importantly, the use of *subfield* VOLE allows us to completely decorrelate the size of $\mathbf{u}$ from that of Δ, something which can fundamentally not be achieved with the INKP OT extension. Concretely, this means that $\mathbf{u}$ only needs to mask the input vector $\mathbf{x}$ of Alice. If $\mathbf{x} \in \mathbb{F}_{2^\ell}^n$, then so do $\mathbf{u}$ and $\mathbf{z}$: the communication now depends solely on the input size.

In total, our BaRK-OPRF communicates $\ell \cdot n$ bits, plus the cost of distributing the seeds for the sVOLE generator. Using the protocol of [BCG+19a] to distribute the seeds, the cost is logarithmic in n, hence its effect on the overall communication vanishes for large enough n.

Specifically, using the protocol of [BCG+19a] to distribute the seeds[1] adds a $(2\log n + 9) \cdot t\lambda$ overhead, where t is a computational security parameter for the underlying LPN assumption, which is slightly smaller than λ (for example, according to Table 1 of [BCG+19a], $t = 118$ suffices to get 128 bits of security for the underlying LPN assumption, when $n = 2^{20}$). This cost is logarithmic in n, hence its effect on the overall communication vanishes for large enough n. Concretely, for $n = 2^{20}$, this amounts to a total communication of $(\ell + 0.7) \cdot n$ bits (where the seed distribution contributes only $0.7n$).

Combining the new OPRF with permutation-based hashing. Plugging our new BaRK-OPRF into KKRT, and using the same parameters for Cuckoo hashing, leads to a protocol with total communication $(1.3 \cdot \ell + 3 \cdot (\kappa + 2\log n))n + o(n)$ bits (where the $o(n)$ terms capture the costs of distributing the PCG seeds). Concretely, for $n = 2^{20}$ and $\ell = 32$ (resp. 64), this already brings the cost down, from $1008n$ bits to $282n$ bits (resp. $324n$ bits). However, this can be further improved using the well-established notion of *permutation-based* hashing [PSS+15]. Concretely, in *permutation-based* hashing, an item x is written as $x_L || x_R$, where x_L is $\log(1.3n)$-bit long. The item x is inserted by mapping x_R to the bin $x_L \oplus f(x_R)$, where f is a k-wise independent hash function, for some large enough k. This guarantees that no collision occurs, because if two items x, x' end up mapping the same value to the same bin, this means that $x_R = x'_R$ and $x_L \oplus f(x_R) = x'_L \oplus f'(x'_R)$, hence $x = x'$. When multiple hash functions are used, as in Cuckoo hashing, the index of the hash function must be appended to x_R.

Interestingly, our use of sVOLE is crucial to enabling a permutation-hashing-based optimization: the latter only provides savings when the communication involves a $O(\ell \cdot n)$ component (which neither KKRT nor any modern OKVS-based PSI has). In our protocol, however, it further reduces the communication to $(1.3 \cdot (\ell - \log(1.3n) + 1) + 3 \cdot (\kappa + 2\log n))n + o(n)$ bits, which gives $275n$ bits for $n = 2^{20}$ and 32-bit items, or $317n$ bits for 64-bit items. In itself, this is a really small communication improvement. However, it has an important consequence: it implies that the Alice-to-Bob communication is now completely dominated by the Bob-to-Alice communication. Concretely, this means that we can easily afford to use a much higher number of bins (which is $1.3n$ currently) if it can allow us to reduce the number of hash functions (which is 3). This brings us to our last optimization.

Packing multiple items per bin with generalized Cuckoo hashing. In this last optimization, our goal is to reduce the number of hash functions used in the Cuckoo hashing protocol, from 3 to 2, by increasing the number of bins to compensate. Unfortunately, this does not work directly with standard cuckoo hashing even while using a reasonably small stash since the cost of handling the stash is high, and nullifies all communication benefits of using two hash functions in the first place.

Instead, we use a different approach: we add one degree of freedom to the Cuckoo hashing

[1]This protocol uses a length-t reverse VOLE protocol as a blackbox, which we instantiate with the construction of [ADI+17].

parameters, *by allowing bins to contain multiple items.* This generalization of Cuckoo hashing is not new: it has been studied in details in several works [DW07; Wie+17], because it comes with a much nicer cache-friendliness than standard Cuckoo hashing.

In (d, k)-Cuckoo hashing, n items are mapped to $(1+\varepsilon)\cdot n$ bins using k hash functions, and each bin is allowed to contain up to d items. Allowing more items per bins significantly improves the efficiency; for example, $(3, 2)$-Cuckoo hashing is known to perform strictly better than standard $(1, 3)$-Cuckoo hashing in terms of occupancy (i.e., the total number of slots $N = d\cdot(1+\varepsilon)\cdot n$ which must be used to guarantee a $o(1)$ failure probability). Based on existing analysis of this variant [Wie+17], it seems reasonable to expect that $(3, 2)$-Cuckoo hashing already achieves a strictly smaller failure probability compared to $(1, 3)$-Cuckoo hashing, with a smaller number of bins.

We relied on extensive computer simulations on small values of n (from 256 to 2048) to select parameters, and extrapolated from these results parameters for larger values of n. More precisely, we ran 10^7 experiments with $(3, 2)$-Cuckoo hashing for $n \in \{2^8, 2^9, 2^{10}\}$ (we also experimented with 2^{11}, but with a smaller number of experiments) with $c\cdot n$ bins for various values of c. Even for a value as low as $c = 0.65$ and values of n as low as 2^9, our experiments never reported any insertion failure, indicating that the empirical failure probability should already be way below 2^{-20}. Since the theoretical failure probability is known to scale as $O(1/n^\delta)$ for some constant δ with reasonably small constant factors, we extrapolate that for large enough values of n, e.g. $n \geq 2^{18}$, the failure probability should be well below 2^{-40}.

Alternative hashing variants. Alternatively, when allowing multiple items per bins, we can consider other hashing variants. Two natural choices are two-choice hashing [PRT+19], where each bin can have up two d items and each item is placed in the least-full of two bins, and simple hashing, where a single hash function is used to map the items to bins (standard results show that, when hashing n items to $O(n)$ bins this way, the maximum load with be of the order of $\log n/\log\log n$ with high probability). As we will see, these choices of hashing lead to various communication versus computation tradeoffs in our protocols, and the optimal choice also depends on the database size.

A membership BaRK-OPRF. There remains a non-trivial task: to use some of the above hashing variants, we need a protocol to handle hashing with up to d items per bins. Intuitively, denoting $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(d)})$ the d entries of the bin i, we want to construct a new kind of *membership* OPRF (similar in spirit to the notion of multi-point OPRF in the literature), where Bob obtains $F_{\Delta,K_i}(y)$ and Alice obtains the set $F_{\Delta,K_i}(\mathbf{x_i}) = \{F_{\Delta,K_i}(x_i^{(j)})\}_{j\leq d}$. This implies that $F_{\Delta,K_i}(y) \in F_{\Delta,K_i}(\mathbf{x_i})$ if and only if y is equal to any entry of $\mathbf{x_i}$, and $F_{\Delta,K_i}(y)$ looks pseudorandom to Alice otherwise.

Going back to the BaRK-OPRF, recall that for a bin i where Alice placed x_i and Bob placed y_i, Alice computes $H(i, v_i)$ and Bob computes $H(i, K_i - \Delta y_i) = H(i, \Delta\cdot(x_i - y_i) + v_i)$. Here, we view the $x_i - y_i$ term as $P_{x_i}(y_i)$, where $P_{x_i} = X - x_i$ is a degree-1 polynomial with root x_i. This view suggests a natural generalization of this approach, where the P_{x_i} polynomials are replaced by higher degree polynomials. Define $P_{\mathbf{x_i}}$ to be the polynomial $\prod_{j=1}^{d}(X - x_i^{(j)})$, and let $(c_{j,i})_{0\leq j\leq d-1}$ denote its coefficients: $P_{\mathbf{x_i}}(X) = X^d + \sum_{j=0}^{d-1} c_{j,i}\cdot X^j$. Our new *membership* BaRK-OPRF is a direct generalization of the BaRK-OPRF from Section 3.3.1, which we sketch below.

Our construction. Let m be the bitlength of Alice's inputs inside the bins, and let $(\mathbf{x_1}, \cdots, \mathbf{x_N})$ be the inputs of Alice in each of the N bins, where the inputs in each bin are viewed as length-d vectors of elements of $\mathbb{F}_{2^m}$. We assume for simplicity that m divides λ, the computational security parameter. Alice and Bob use d sVOLE protocol (e.g. [CRR21]) over the field $\mathbb{F}_{2^\lambda}$, with subfield $\mathbb{F}_{2^m}$, *with the same value* Δ.[2] Let $(\mathbf{u_j}, \mathbf{v_j})_{j\leq d}$ be the outputs of Alice, and $(\Delta, (\mathbf{w_j})_{j\leq d})$ be the output of

[2]Note that all known sVOLE protocols allow Bob to choose the value of Δ, hence Bob can enforce the use of the same

Bob. Recall that $\mathbf{w_j} = \Delta \cdot \mathbf{u_j} + \mathbf{v_j}$.

For each $\mathbf{x_i}$, let $(c_{0,i}, \cdots, c_{d-1,i})$ be the coefficients of the polynomial $P_{\mathbf{x_i}}$ (omitting the coefficient of X^d, which is always 1). Let $\mathbf{c_j}$ denote the vector $(c_{j,i})_{i \leq N}$ for $j = 0$ to $d-1$. Alice sends $\mathbf{z_j} = \mathbf{c_j} - \mathbf{u_j}$ for $j = 0$ to $d-1$ to Bob, who defines the membership BaRK-OPRF keys to be Δ and $K_i = (k_{j,i})_{0 \leq j \leq d-1} = (\Delta \cdot z_{j,i} + w_{j,i})_{0 \leq j \leq d-1}$ for $i = 1$ to N. Define the following degree-d polynomial P_{Δ,K_i} over $\mathbb{F}_q$: $P_{\Delta,K_i}(X) = \Delta \cdot X^d + \sum_{j=0}^{d-1} k_{j,i} \cdot X^j$. The OPRF is defined as follows: $F_{\Delta,K_i}(y) = H(i, P_{\Delta,K_i}(y))$ (all operations are over $\mathbb{F}_{2^\lambda}$). Eventually, for each bin i, Alice sets her d tuple of outputs to be $F_{\Delta,K_i}(\mathbf{x_i}) = \{H(i, \sum_{j=0}^{d-1} v_{j,i} \cdot (x_i^{(k)})^j\}_{k \leq d}$. Observe that, since $k_{j,i} = \Delta z_{j,i} + w_{j,i} = \Delta c_{j,i} + v_{j,i}$ for all i, j, we have $H(i, P_{\Delta,K_i}(y)) = H\left(i, \Delta \cdot \left(y^d + \sum_{j=0}^{d-1} c_{j,i} y^j\right) + \sum_{j=0}^{d-1} v_{j,i} y^j\right)$, which is equal to $H\left(i, \Delta \cdot P_{\mathbf{x_i}}(y) + \sum_{j=0}^{d-1} v_{j,i} y^j\right)$. Therefore, if there exists $k \in \{1, \cdots, d\}$ such that $y = x_i^{(k)}$, we have $P_{\mathbf{x_i}}(y) = 0$, and $H(i, P_{\Delta,K_i}(y)) = H(i, \sum_{j=0}^{d-1} v_{j,i} \cdot (x_i^{(k)})^j) \in F_{\Delta,K_i}(\mathbf{x_i})$. On the other hand, whenever $P_{\mathbf{x_i}}(y) \neq 0$, then the $\Delta \cdot P_{\mathbf{x_i}}(y)$ term in the hash makes the output pseudorandom from the viewpoint of Alice, under the correlation robustness of the hash function.

Tying up loose ends. Using the new construction from the previous Section, together with $(3, 2)$-Cuckoo hashing, leads to a total communication of $(0.65 \cdot 3(\ell - \log(0.65n) + 1) + 2 \cdot (\kappa + 2\log n))n + o(n)$ bits, where the $o(n)$ corresponds to the cost of setting up the PCG seeds. For $n = 2^{20}$ and 32 bits items, this gives $148n$ bits of communication. We mention a few remaining details. First, in the construction of membership BaRK-OPRF, Alice and Bob need to invoke $d = 3$ length-N sVOLE. In fact, it suffices to invoke a single length-$3N$ sVOLE, and to cut the output in three equal length parts, to obtain the necessary correlation. This means that the concrete cost of distributing the sVOLE seeds remains that of generating a single sVOLE (e.g. $\approx 0.7n$ bits for $n = 2^{20}$).

Second, in the above, we overlooked an important subtlety: a bin can possibly contain less than d items. In KKRT, this was handled by adding dummy items to empty bins. We use instead a more efficient approach with a negligible extra cost called a *variant* of our OPRF (details in section 3.4).

3.3.2 Malicious Security

We then turn our attention to maliciously secure PSI. Here, it is well known that Cuckoo hashing and two-choice hashing are not usable. Consequently, we focus on simple hashing as our choice of the underlying hash technique. Using maliciously secure subfield-VOLE, which can be implemented very efficiently [BCG+19a; CRR21], we enhance our membership BaRK-OPRF to the malicious setting, with a minimal overhead. Then, we apply two standard methods to achieve security against malicious adversaries in our PSI protocol:

First method: direct approach. The first method increases the PRF output length to λ. Using the analysis of [RS21], this suffices to allow for extracting the input of a malicious sender. However, this makes the communication depend linearly on λ, which severely harms communication complexity.

Second method: dual execution. To recover a λ-independent communication complexity, we then turn our attention to the dual execution technique [RR17]. Here, the idea is simple: the parties will invoke the malicious BaRK-OPRF twice, exchanging their roles. Then, the sender sends, for each entry x of his database, a value of the form $\mathsf{PRF}_A(x) \oplus \mathsf{PRF}_B(x)$, where $\mathsf{PRF}_A(x)$ is obtained by the sender when invoking the BaRK-OPRF functionality as sender, and $\mathsf{PRF}_B(x)$ is the PRF output obtained when invoking the functionality as receiver. Here, it becomes possible to extract the input set of each party simply from its call as receiver to the BaRK-OPRF functionality, which does not require to

Δ across all instances.

increase the output length of the OPRF. The price to pay is that the protocol now uses two calls to the BaRK-OPRF. Concretely, the total communication becomes $(2\cdot N\cdot d(\ell-\log(N))+(\kappa+\log n))n+o(n)$, where N is the number of bins, d the maximum load of a bin, and ℓ the input size (e.g., for $n = 2^{20}$, one can choose $N = n/10$ and $d = 47$, see [RR17, Figure 5]). For small database entries, this outperforms all known malicious PSI protocols.

3.4 PSI from Subfield-VOLE

3.4.1 Membership Batched OPRF

Our BaRK-OPRF allows the sender to hold a set of keys $(\mathbf{k_i})_{i\le N}$ such that each key is assigned with a tuple of d input elements of the receiver and then the receiver learns a PRF output on each element in this tuple corresponding with the same key. More formally, denoting $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(d)})$ consisting of d entries, the sender gets $F(i, y)$ and the receiver obtains a set $\{F(i, x_i^{(j)})\}_{j\le d}$ such that $F(i, y) \in \{F(i, x_i^{(j)})\}_{j\le d}$ if and only if y is equal to any entry of $\mathbf{x_i}$, and $F(i, y)$ looks pseudorandom to the receiver otherwise.

Figure 3.1: Ideal functionality $\mathcal{F}_{\mathsf{oprf}}$

PARAMETERS:

$\mathbb{F}_p$ is a finite field. There are 2 parties, a sender and a receiver with input set $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\} \subseteq \mathbb{F}_p$ where $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(d)})$.

FUNCTIONALITY:

- Wait for input $(\mathsf{sender}, \mathsf{id})$ from the sender and $(\mathsf{receiver}, \mathsf{id}, \mathsf{X})$ from the receiver. The functionality samples a PRF F then $\forall x \in \mathbf{x_i}$ outputs $F(i, x)$ to the receiver for $i \in [1, N]$.
- When the sender inputs any $(i, y) \in [1, N] \times \mathbb{F}_p$, functionality gives $F(i, y)$ to the sender.

Main construction. Assume that the receiver inputs the set of $n = Nd$ elements: $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\} \subseteq \mathbb{F}_p$ where $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(d)})$. First, the sender and the receiver invoke the $\mathcal{F}_{\mathsf{sVOLE}}$ protocol of dimension n, with their roles reversed, to get a random sVOLE correlation. Specifically, the receiver learns a pair of vectors $(\mathbf{u}, \mathbf{v})$ where $\mathbf{u} \in \mathbb{F}_p^n$, $\mathbf{v} \in \mathbb{F}_q^n$, the sender gets $\Delta \in \mathbb{F}_q$ and $\mathbf{w} := \Delta \cdot \mathbf{u} + \mathbf{v}$. Denoting $\mathbf{u} = (\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_N})$ where $(u_{j,i})_{1\le j\le d}$ are d entries of vector $\mathbf{u_i}$. This notation is the same for $\mathbf{v}, \mathbf{w}$. Consider $\mathbf{x_i}$ and its associated polynomial as $P_{\mathbf{x_i}}(X) = \prod_{j=1}^{d}(X - x_i^{(j)}) = X^d + \sum_{j=1}^{d} c_{j,i} \cdot X^{j-1}$ where $c_{j,i} \in \mathbb{F}_p$ for $i \in [1, N]$, $j \in [1, d]$.

Now, the receiver defines $\mathbf{c_i} := (c_{j,i})_{j\le d}$, $\mathbf{c} := (\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_N})$, and then $\forall i \in [1, N]$ sends to the sender $\mathbf{z_i} := \mathbf{c_i} - \mathbf{u_i} \in \mathbb{F}_p^d$. Above, the $\mathbf{u_i}$ are masks for the coefficients $\mathbf{c_i}$ of (the polynomial associated) $\mathbf{x_i}$. Indeed, $\mathbf{u_i}$ are distributed uniformly at random in the subfield $\mathbb{F}_p$, then the vector $\mathbf{z_i}$ is a uniformly random over $\mathbb{F}_p^n$ from the viewpoint of the sender. The two parties will run a coin flipping protocol to get a random value $t \leftarrow \mathbb{F}_q$. For $i \in [1, N]$, the receiver defines the PRF output on each input $x \in \mathbf{x_i}$ as $F(i, x) = \mathsf{H}\left(i|t|x\ ,\ \sum_{j=1}^{d} v_{j,i} \cdot x^{j-1}\right)$.

On the other hand, after receiving the vectors $\mathbf{z_i}$, for $i \in [1, N]$, the sender defines the vector $\mathbf{k_i} := \mathbf{w_i} + \Delta \cdot \mathbf{z_i}$. As a consequence, for any input $(i, y) \in [1, N] \times \mathbb{F}_p$, its PRF output is computed as: $F(i, y) = \mathsf{H}\left(i|t|y\ ,\ \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}\right)$.

Correctness and Security. To see why PRF output is defined as above. Observe that $\mathbf{k_i} := \mathbf{w_i} + \Delta \cdot \mathbf{z_i} = \mathbf{v_i} + \Delta \cdot \mathbf{c_i}$. Then, we have

$$\Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1} = \Delta \cdot y^d + \sum_{j=1}^{d} (v_{j,i} + \Delta \cdot c_{j,i}) \cdot y^{j-1}$$
$$= \Delta \cdot (y^d + \sum_{j=1}^{d} c_{j,i} \cdot y^{j-1}) + \sum_{j=1}^{d} v_{j,i} \cdot y^{j-1} = \Delta \cdot P_{\mathbf{x_i}}(y) + \sum_{j=1}^{d} v_{j,i} \cdot y^{j-1}$$

so if $y \in \mathbf{x_i}$ then $P_{\mathbf{x_i}}(y) = 0$ which leads to $F(i, y) \in \{F(i, x_i^{(j)})\}_{j \le d}$.

Theorem 3.4.1. *The protocol* Π_{oprf} *(Figure 3.2) instantiated with random oracles* H, H'*, securely realizes the ideal functionality of* $\mathcal{F}_{\mathsf{oprf}}$ *(Figure 3.1) against a malicious setting in the* $\mathcal{F}_{\mathsf{sVOLE}}$ *hybrid model.*

Proof. **Corrupted sender.** The Sim interacts with the sender as follows:

- Sim emulates $\mathcal{F}_{\mathsf{sVOLE}}$, waits for sender to send $\Delta, \mathbf{w}$.
- Sim samples uniformly $h_r \leftarrow \mathbb{F}_q$ and then sends h_r to $\mathcal{A}$.
- After receiving h_s, Sim samples uniformly vectors $(\mathbf{z_i})_{1 \le i \le N}$ instead of $\mathbf{z_i} := \mathbf{c_i} - \mathbf{u_i}$.
- Sim samples uniformly $t_r \leftarrow \mathbb{F}_q$ and programs $\mathsf{H}'(t_r) := h_r$.
- On behalf of receiver, Sim sends $\mathbf{z_i}$ and t_r to $\mathcal{A}$.
- Sim computes $\mathbf{k_i} = \mathbf{w_i} + \Delta \mathbf{z_i}$. Whenever $\mathcal{A}$ queries $\mathsf{H}(i|t|y, q)$ where $q = \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}$ and $\mathsf{H}(i|t|y, q)$ has not been previously queried, Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ with (i, y) being the input of sender, Sim samples uniformly a value as $F(i, y)$ and programs
$$\mathsf{H}\left(i|t|y, \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}\right) := F(i, y)$$

This simulation is indistinguishable from the real world by the following hybrids:

- *Hybrid 0.* The same as the real protocol with an honest receiver and $\mathcal{F}_{\mathsf{sVOLE}}$ is executed honestly.
- *Hybrid 1.* The same as hybrid 0 except the Sim emulates $\mathcal{F}_{\mathsf{sVOLE}}$, receives $\Delta, \mathbf{w}$ from the $\mathcal{A}$.
- *Hybrid 2.* On behalf of receiver, Sim samples $h_r \leftarrow \mathbb{F}_q$ and sends it to $\mathcal{A}$ instead of $h_r := \mathsf{H}'(t_r)$ where t_r is sampled in $\mathbb{F}_q$. Then before sending the value t_r to $\mathcal{A}$, Sim samples $t_r \leftarrow \mathbb{F}_q$ and programs $\mathsf{H}'(t_r) := h_r$. The probability of abort when $\mathsf{H}'(t_r)$ has been queried previous is negligible $O(1/\mathbb{F}_q) = O(2^{-\lambda})$ since t_r is sampled uniformly. This hybrid has an identical distribution.
- *Hybrid 3.* Sim samples uniformly vectors $(\mathbf{z_i})_{1 \le i \le N}$ from $\mathbb{F}_p$ as opposed to $\mathbf{z_i} := \mathbf{c_i} - \mathbf{u_i}$. Since $\mathbf{u_i}$ is distributed uniformly at random and $\mathbf{c_i}$ is arbitrary vector in $\mathbb{F}_p$ then this hybrid is indistinguishable from the previous hybrid.
- *Hybrid 4.* After receiving t_s from $\mathcal{A}$. Sim computes $\mathbf{k_i} = \mathbf{w_i} + \Delta \mathbf{z_i}$ and $t := t_r \oplus t_s$. Whenever $\mathcal{A}$ queries $\mathsf{H}(i|t|y, q)$ where $q = \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}$ and $\mathsf{H}(i|t|y, q)$ has not been previously queried, Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ with (i, y) being the input of sender, Sim samples uniformly a value as $F(i, y)$ and programs
$$\mathsf{H}\left(i|t|y, \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}\right) := F(i, y)$$

Figure 3.2: Our batch BaRK-OPRF Π_{oprf} based on subVOLE

PARAMETERS:

- Given $\mathbb{F}_p \subseteq \mathbb{F}_q$ where $\mathbb{F}_q \approx O(2^\lambda)$, $\mathsf{H} : \{0,1\}^* \times \mathbb{F}_q \to \{0,1\}^v$ and $\mathsf{H}' : \mathbb{F}_q \to \mathbb{F}_q$ are random oracles.
- The sender has no input and the receiver inputs a set $X = \{\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_N}\} \subseteq \mathbb{F}_p$ where $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(d)})$ and $n = Nd$.

PROTOCOL:

1. The sender and the receiver invoke to the $\mathcal{F}_{\mathsf{sVOLE}}$ of dimension n in the $\mathbb{F}_q$ over the $\mathbb{F}_p$ with the inverse role. The receiver gets two random vectors $\mathbf{u} \in \mathbb{F}_p^n, \mathbf{v} \in \mathbb{F}_q^n$ and the sender receives $\Delta \in \mathbb{F}_q$, $\mathbf{w} := \Delta\mathbf{u} + \mathbf{v} \in \mathbb{F}_q^n$. Denoting $\mathbf{u} = (\mathbf{u_1}, \mathbf{u_2}, \dots, \mathbf{u_N})$ where $\mathbf{u_i} = (c_{j,i})_{1 \le j \le d}$. This denotation is the same for $\mathbf{v}, \mathbf{w}$.
2. The receiver samples $t_r \leftarrow \mathbb{F}_q$ and sends $h_r := \mathsf{H}'(t_r)$ to the sender.
3. The sender samples $t_s \leftarrow \mathbb{F}_q$ and sends $h_s := \mathsf{H}'(t_s)$ to the receiver.
4. The receiver determines the associated polynomial for each $\mathbf{x_i}$ as
$$P_{\mathbf{x_i}}(X) = \prod_{j=1}^{d} (X - x_i^{(j)}) = X^d + \sum_{j=1}^{d} c_{j,i} \cdot X^{j-1}$$
where $c_{j,i} \in \mathbb{F}_p$ for $i \in [1, N], j \in [1, d]$.
5. Denoting $\mathbf{c_i} := (c_{j,i})_{1 \le j \le d}$; $\mathbf{c} := (\mathbf{c_1}, \mathbf{c_2}, \dots, \mathbf{c_N})$, the receiver computes $\mathbf{z_i} := \mathbf{c_i} - \mathbf{u_i} \in \mathbb{F}_p^d$, and then sends $\mathbf{z_i}$ and t_r to the sender.
6. The sender aborts if $\mathsf{H}'(t_r) \neq h_r$.
7. The sender sends t_s to the receiver, the receiver aborts if $\mathsf{H}'(t_s) \neq h_s$ and both parties define $t = t_s \oplus t_r$.
8. The receiver outputs the PRF values on the input $x \in \mathbf{x_i}$ for $i \in [1, N]$ as
$$F(i, x) = \mathsf{H}\left(i|t|x \ , \ \sum_{j=1}^{d} v_{j,i} \cdot x^{j-1} \right)$$
9. For $i \in [1, N]$, the sender defines $\mathbf{k_i} = \mathbf{w_i} + \Delta\mathbf{z_i}$. For any input $(i, y) \in [1, N] \times \mathbb{F}_p$, the sender computes the PRF output by below formula
$$F(i, y) = \mathsf{H}\left(i|t|y \ , \ \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1} \right)$$

Sim will abort if there exists (i, y) such that $\mathsf{H}(i|t|y, q)$ has been previously queried. Since t_s being an arbitrary value fixed by $\mathcal{A}$ from the beginning and t_r is uniformly sampled before sending then t have an uniformly distribution over $\mathbb{F}_q \approx O(2^\lambda)$ which leads to a negligible probability of abort.

Corrupted receiver. The Sim interacts with the receiver as follows:

- Sim emulates $\mathcal{F}_{\mathsf{sVOLE}}$ functionality, waits for the receiver to send $\mathbf{u}, \mathbf{v}$.
- Sim samples uniformly $h_s \leftarrow \mathbb{F}_q$ and then sends h_s to $\mathcal{A}$.

- After corrupted receiver sends $(\mathbf{z_i})_{1\leq i\leq N}$, Sim defines $\mathbf{c_i} := \mathbf{z_i} + \mathbf{u_i}$. From $\mathbf{c_i}$, Sim extracts the set $X = \{\mathbf{x_i}\}_{i\leq N}$.
- After receiving t_r from $\mathcal{A}$, Sim samples uniformly $t \in \mathbb{F}_q$, defines $t_s := t - t_r$ and programs $\mathsf{H}'(t_s) := h_s$ then sends t_s to $\mathcal{A}$.
- Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ with X being the input of receiver then Sim samples a sequence of uniformly random values which defined as $\{F(i,x) \mid \forall x \in \mathbf{x_i}\}_{i\leq N}$. Sim programs

$$\mathsf{H}\left(i|t|x\ ,\ \sum_{j=1}^{d} v_{j,i}\cdot x^{j-1}\right) := F\left(i,x\right)$$

This simulation is indistinguishable from the real world by the following hybrids:

- *Hybrid 0.* The same as the real protocol with a honest sender and $\mathcal{F}_{\mathsf{sVOLE}}$ is executed honestly.
- *Hybrid 1.* The same as hybrid 0 except the Sim emulates $\mathcal{F}_{\mathsf{sVOLE}}$, receiving $\mathbf{u}, \mathbf{v}$ from the $\mathcal{A}$.
- *Hybrid 2.* On behalf of sender, Sim samples $h_s \leftarrow \mathbb{F}_q$ and sends it to $\mathcal{A}$ instead of $h_s := \mathsf{H}'(t_s)$ where t_s is sampled in $\mathbb{F}_q$. After receiving t_r from $\mathcal{A}$, Sim samples $t \leftarrow \mathbb{F}_q$ and defines $t_s := t \oplus t_r$. Sim programs $\mathsf{H}'(t_s) := h_s$ then sends t_s to $\mathcal{A}$. The probability of abort when $\mathsf{H}'(t_s)$ has been queried previous is negligible $O(1/\mathbb{F}_q) = O(2^{-\lambda})$ since t is sampled uniformly and t_r is an arbitrary value. This hybrid has an identical distribution.
- *Hybrid 3.* After receiving $(\mathbf{z_i})_{i\leq N}$ from $\mathcal{A}$, Sim computes $\mathbf{c_i} = \mathbf{z_i} + \mathbf{u_i}$ for all $i \in [1,N]$.
  Now for each $i \in [1,N]$, Sim defines the polynomial $P_{\mathbf{x_i}}(X) = X^d + \sum_{j=1}^{d} c_{j,i}\cdot X^{j-1}$ and then extracts a set $\mathbf{x_i}$ which includes all the root of $P_{\mathbf{x_i}}$. Formally,

  $$\mathbf{x_i} = \{x \in \mathbb{F}_p \mid P_{\mathbf{x_i}}(x) = 0\}$$

  This hybrid can not be distinguished with the previous since Sim only extracts set.
- *Hybrid 4.* Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ with $X = \{\mathbf{x_i}\}_{i\leq N}$ being the input of receiver then Sim samples a sequence of uniformly random values over $\{0,1\}^v$ which defined as $\{F(i,x) \mid \forall x \in \mathbf{x_i}\}_{i\leq N}$.
  For each $x \in \mathbf{x_i}$, Sim aborts if any $\mathsf{H}(i|t|x,q)$ where $q := \sum_{j=1}^{d} v_{j,i}\cdot x^{j-1}$ has been made by $\mathcal{A}$. The probability of aborting is negligible since t is uniformly distributed over $\mathbb{F}_q \approx O(2^\lambda)$. Sim programs

  $$\mathsf{H}\left(i||t||x\ ,\ \sum_{k=1}^{d} v_{(i-1)d+k}\cdot x^{k-1}\right) := F\left(i,x\right)$$

  Since the $F(i,x)$ is sampled uniform then this hybrid is indistinguishable with previous one.
- *Hybrid 5.* Sim will abort protocol if corrupted receiver is able to learn the PRF value on a element which is not in any set $\mathbf{x_i}$ for $i \in [1,N]$. This means that the $\mathcal{A}$ made a query $\mathsf{H}\left(i|t|x,h\right)$ such that $x \in \mathbb{F}_p \setminus \mathbf{x_i}$ for $i \in [1,N]$ and $h = \Delta\cdot y^d + \sum_{j=1}^{d} k_{j,i}\cdot x^{j-1}$. Observe that

  $$\Delta\cdot y^d + \sum_{j=1}^{d} k_{j,i}\cdot x^{j-1} = \Delta\cdot P_{\mathbf{x_i}}(x) + v_{j,i}\cdot x^{j-1}$$

  Since Δ is distributed uniformly over $\mathbb{F}_q$ in the viewpoint of $\mathcal{A}$ and $P_{\mathbf{x_i}}(x) \neq 0$ for $x \notin \mathbf{x_i}$ then the probability of aborting is negligible at most $O(1/2^q)$. This concludes the proof.

□

Note that the output v of H is chosen depending on the concrete structure of PSI and the target setting (semi-honest or malicious). This parameter is detailed in the Section 3.4.2 for a semi-honest setting and the Section 3.4.3 for a malicious setting.

3.4.2 Semi-honest PSI from mOPRF

A variant of BaRK-OPRF. We now propose a variant of our BaRK-OPRF to deal with the case when the size of each tuple input is not necessarily equal to d. This means that the receiver now can divide the input set to N tuples $\mathbf{x_i}$ and each tuple has less than or equal to d items. Meanwhile, the sender is not allowed to learn about how many exactly items are in each tuple. This functionality can be obtained from our BaRK-OPRF plus a small extra cost, i.e., a *subfield* VOLE of length N over the subfield $\mathbb{F}_2$.

The idea is as follows. The receiver's input set $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\} \subseteq \mathbb{F}_p$ where $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(j_i)})$, $j_i \leq d$. The polynomial associated to $\{\mathbf{x_i}\}_{i \leq N}$ will be expressed as a polynomial of degree d: $P_{\mathbf{x_i}}(X) = \prod_{j=1}^{j_i}(X - x_i^{(j)}) = \sum_{j=1}^{d+1} c_{j,i} \cdot X^{j-1}$ where $c_{j,i} \in \mathbb{F}_p$.

As a result, the set of the coefficients of $P_{\mathbf{x_i}}(X) = (c_{1,i}, c_{2,i}, \ldots, c_{d+1,i}\}$. We remark that, compared to the associated polynomial in our original BaRK-OPRF which has a constant coefficient of degree d of 1, in our variant version this coefficient will equal 0 or 1 since the degree of $P_{\mathbf{x_i}}(X)$ is *less* than or equal to d. So, it requires $(d+1)$ masks for this polynomial instead of d, but the mask for the coefficient of degree d only needs to be in $\mathbb{F}_2$. For each tuple, we require an additional value $u_i \in \mathbb{F}_2$, so in total we need an additional subfield VOLE of length N over the subfield $\mathbb{F}_2$.

More formally, the sender and receiver invoke a subfield VOLE of length n over the subfield $\mathbb{F}_p$ as before (all the notations in Figure 3.2 are reused), and additionally invoke another subfield VOLE instance over the subfield $\mathbb{F}_2$ of length N with an inverse role, while the receiver gets $\mathbf{u}' \in \mathbb{F}_2^N$, and $\mathbf{v}' \in \mathbb{F}_q^N$ the sender holds $\Delta \in \mathbb{F}_q$ (Δ is the same for each time invoking *subfield* VOLE) and $\mathbf{w}' := \Delta \cdot \mathbf{u}' + \mathbf{v}'$. The receiver sends to the sender vectors $\mathbf{z_i}$ as before, and an extra vector $\mathbf{z}'$ defined as $z_i' := c_{d+1,i} - u_i'$ for $i \in [1, N]$. The receiver outputs on input $x \in \mathbf{x_i}$ are computed as $F(i, x) = \mathsf{H}(i|t|x\ ,\ v_i' \cdot x^d + \sum_{j=1}^{d} v_{j,i} \cdot x^{j-1})$. On the other hand, the sender defines their PRF values on input (i, y) where $i \in [1, N]$, $y \in \mathbb{F}_p$ as $F(i, y) = \mathsf{H}(i|t|y\ ,\ (w_i' + \Delta z_i') \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1})$.

Main construction of a new PSI. The sender and the receiver have two input sets $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$. Assume that all of these elements have the bit-length ℓ. Intuitively, our BaRK-OPRF is constructed from subVOLE to handle the case when having multiple items per bin. Then this specialized BaRK-OPRF can combine with some hashing techniques to form an efficient PSI protocol. In the next part Section 3.4.2, we discuss these types of hashing. Our PSI protocol is described in Figure 3.3; it builds upon the protocol of [KKR+16] using GCH and BaRK-OPRF. For simplicity, we describe our protocol directly with generalized Cuckoo hashing; adapting the protocol to other variants is immediate. We elaborate on our protocol below. In our protocol, the receiver first uses (d, k)-Cuckoo hashing to map his input set Y to a table with N bins, note that the bit-length of the values stored in a bin is $\ell - \log N$ insted of ℓ. Depending on the size of n, we use one of two approaches to handle the bins which are not full (the threshold was chosen empirically to optimize communication).

- If $n \geq 2^{20}$, the variant of our BaRK-OPRF (using an additional subfield VOLE over $\mathbb{F}_2$) is used; for such sizes, the concrete cost of implementing the additional sVOLE vanishes.

- Otherwise, when $n < 2^{20}$, the receiver adds dummy items to bins such that each bin contains *exactly* d items. To avoid collisions between the dummy items and the elements in the same bin of the sender, we pad an extra bit to all items in the following way: $i|x|b$ where i is the index of hash function corresponding with the stored value x while $b = 1$ if x is a dummy item added and $b = 0$ otherwise.

In both case, the sender computes $k \cdot n$ PRF evaluations and sends (shuffled) to the receiver, who compares them with his OPRF outputs and outputs the intersection set. To reduce the computational cost in this step, the sender can send separately each set H_i $(i \in [1, k])$ which contains the PRF outputs of each $x \in X$ with the related bin $h_i(x)$. Then for each element, the receiver only needs to search for one set (among k sets H_i) of n items instead of $k \cdot n$.

Alternative hashing methods. There are two hashing schemes that can be fit into our PSI structure.

2-choice hashing [PRT+19] is a variant of Cuckoo hashing where one item x is assigned to one of two bins $h_1(x)$ or $h_2(x)$. However, there is no restriction on the number of items per bin and an item is put in a bin which already has fewer items. [PRT+19] proposes both theoretical references and heuristic parameters for 2-choice hashing, which require only a small number of dummy items. Let us assume we have n items and 2 hash functions; using 2-choice hashing allows to map n items to N bins in time $O(n \log n)$ where each bin contains at most $L = \lceil n/N \rceil + 1$ items with a probability $1 - O(1/N)^{L-1}$.

Simple hashing uses one hash function h to map an item x to bin $h(x)$. For security, the number of items per bin can leak some information then it requires padding each bin with dummy items until having an equal number of items per bin. With very high probability, for $N = O(n \log n)$ bins, the maximum possible items per bin is $O(\log n)$. The percentage of the occupation of dummy items is higher than others. However, simple hashing avoids ambiguities about where an item can be placed, a property which is crucial in the malicious setting.

Parameters. In this section, we discuss concrete parameters used in our new PSI semi-honest protocol. We use $\lambda = 128$ and $\kappa = 40$. The protocol contains several parameters:

The length of OPRF output. The output domain of PRF would be $\{0, 1\}^v$ where $v = \kappa + 2\log_2(n)$ guarantees a $2^{-\kappa}$ bound on the collision probability of PRF outputs among the two size-n sets. Furthermore, communicating the hashes can be reduced to communicating only $\approx \kappa + \log n$ bits per hash, using a heuristic technique of [TLP+17] that directly leads to an optimization of our PSI protocol.

The size of $\mathbb{F}_p$ and $\mathbb{F}_q$ in BaRK-OPRF. After using permutation-based hashing, each element is mapped to a bin with a stored value in this bin, the bit-length reduces from ℓ to $\ell - \log N$. The input set of BaRK-OPRF in PSI protocol constructs from stored values concatenating with some extra bits. Then the bit-length of an input element of BaRK-OPRF is computed as $\ell - \log N + 1$ if $n \geq 2^{20}$ or $\ell - \log N + 2$ otherwise, i.e., the size of $q = 2^{\ell - \log N + 1}$ or $q = 2^{\ell - \log N + 2}$ respectively.

Generalized Cuckoo hashing. We use a (d, k)-general cuckoo hashing scheme without stash. The parameters are chosen such that the failure probability is $2^{-\kappa}$. When $d = 1, k = 3$ these parameters are identical with KKRT except for the number of bins increases slightly to $N = 1.3n$ which is a trade-off to obtain no stash. Even with the higher number of bins, our PSI protocol significantly outperforms KKRT.
To minimize the overall communication, we set $k = 2$ to reduce the cost of sending $k \cdot n$ PRF outputs. We used a Python script to simulate randomly assigning n values to $N = c \cdot n$ bins using $(d, 2)$-Cuckoo hashing, for several values of d and c, and for $n = 2^9, 2^{10}, 2^{11}, 2^{12}$. For a value of c as low as 0.65, we never observed any insertion failure over 10^7 trials for each values of n (for $n = 2^{12}$,

Figure 3.3: Our new semi-honest PSI protocol from BaRK-OPRF

PARAMETERS:

- The sender and the receiver have respectively input sets $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, all elements of bit-length ℓ.
- A (d, k)-generalized Cuckoo hashing (GCH) scheme mapping n items to N bins by k hash functions $h_1, h_2, \ldots, h_k : \{0,1\}^* \to [N]$ where $Nd > n$ and $d = O(1)$ (see Section 3.4.2).

PROTOCOL:

1. The receiver uses (d, k)-Cuckoo hashing with k hash functions to map the elements in Y to the table $\mathcal{B}$ consisting of N bins, where each bin i has $j_i \leq d$ items.

   Denote $y_{j,i}$ is an element in Y assigned to position j of bin i and its stored value in table $\mathcal{B}$ is $y'_{j,i}$.

2. Depending on the size of n, there are two alternatives:

   (a) $n \geq 2^{20}$, the sender and receiver invoke our variant of Π_{oprf} where the receiver uses the input set $Y_{\mathcal{B}} = \{\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_N}\}$ defined as follows:

   - $\mathbf{y_i} = \{r_{1,i}, r_{2,i}, \ldots, r_{j_i,i}\}$.
   - $r_{j,i} = t \parallel y'_{j,i}$ where t is index of a hash function such that $h_t(y_{j,i}) = i$.

   (b) $n < 2^{20}$, the sender and receiver directly invoke the Π_{oprf} where the receiver uses the input set $Y_{\mathcal{B}} = \{\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_N}\}$ defined as follows:

   - $\mathbf{y_i} = \{r_{1,i}, r_{2,i}, \ldots, r_{d,i}\}$.
     - For $j \leq j_i$: $r_{j,i} = t \parallel y'_{j,i} \parallel 1$ where t is index of hash function such that $h_t(y_{j,i}) = i$.
     - Otherwise, $r_{j,i} = t \parallel$ dummy value $\parallel 0$ where $t \leftarrow_r [1, k]$.

3. The receiver obtains n instances OPRF:

$$Y' = \{\mathsf{PRF}(i\ ,\ r_{i,j}) \mid i \in [1, N]\ ,\ j \leq j_i\}$$

4. The sender uses the k hash functions to map the n element in X to the N bins. Let x_t denote the value stored at bin $h_t(x)$ when mapping x for $t \in [1, k]$.

5. The sender computes the sets of $k \cdot n$ PRF outputs:

   (a) For $n \geq 2^{20}$: $H_t = \{\mathsf{PRF}(h_t(x)\ ,\ t \parallel x_t) \mid x \in X\}$ for $t \in [1, k]$.

   (b) For $n < 2^{20}$: $H_t = \{\mathsf{PRF}(h_t(x)\ ,\ t \parallel x_t \parallel 1) \mid x \in X\}$ for $t \in [1, k]$.

   Then the sender randomly permutes and sends each set to the receiver.

6. The receiver finds the intersection:

   - if $y \in Y$ is mapped to the position j of bin i by function h_t then check whether $\mathsf{PRF}(i, r_{i,j}) \in H_t$ ($r_{i,j}$ is defined depending on n).
   - Outputs the intersection set.

we could only do 10^6 trials), when using $d = 3$ items per bins. For $d = 2$, the failure probability became noticeable already for $c \approx 1$. Based on known theoretical analysis of (d, k)-Cuckoo hashing, the failure probability is known to scale inverse polynomially with n. Therefore, we expect that for reasonably large values of n (e.g. $n \geq 2^{18}$), our parameters should guarantee a failure probability significantly below 2^{-40}.

2-choice hashing. Following the analysis of [PRT+19], we set the number N of bins to $n/3$, and the maximum load $d = L + 1$ to 4. This guarantees a failure probability which we empirically estimate to be $1/N^{L-1}$, which is below 2^{-40} for all values of n above 2^{14}.

Simple hashing. Eventually, for simple hashing, we set arbitrarily the number of bins N to $n/10$, and derive the corresponding value of d from Figure 5 in [RR17]. We note that the parameters for simple hashing are much less heuristic that the other two, in that concrete bound can actually be achieved which are relatively close to the heuristic (computer-estimated) bounds. For example, [PRT+19] experimentally observes that for a 2^{-40} failure probability, setting $d = 47$ suffices when using $N = n/10$ bins. Using a standard Chernoff bound, it is in fact straightforward to prove formally that $d = 49$ already suffices to reach this failure probability, which is very close to the experimental bound. In contrast, experimental bounds in more complex hashing variants are typically much more distant from provable bounds. The choice of $N = n/10$ is entirely arbitrary: any smaller N leads to better communication, but requires using higher values of d, leading to worse computation (due to the need to perform N polynomial interpolations with degree-d polynomial). This allows for a smooth tradeoff between communication and computation, where better computational power can be used to further reduce the communication. At the extreme end of the spectrum, using $N = 1$ and $d = n$ requires one expensive degree-n polynomial interpolation, but can achieve extremely low communications, e.g., $93n$ bits of communication for $\ell = 32$ and $n = 2^{20}$.

Efficiency. We compare the communication of our protocols, using three hashing methods, on Table 3.1. Briefly, though, compared to the protocol of [RS21], and when using a standard choice of parameters for our protocol (e.g., $n = 2^{20}$, and using generalized Cuckoo hashing with $d = 3$ and $N = 0.65n$), our protocol requires essentially a length-$1.9n$ VOLE (with a small subfield), $0.65n$ degree-3 polynomial interpolations (roughly $3n$ multiplications over a small field), and computing n hashes. In contrast, the enhanced version of [RS21] (using the OKVS of [GPR+21] and the VOLE of [CRR21]) will require solving a linear system to set up an OKVS (this requires on the order of $(1.3 \log n + \lambda)^3$ multiplications over $\mathbb{F}_{2^{128}}$, plus $O(\lambda n)$ operations), computing a length-$1.3n$ VOLE (over $\mathbb{F}_{2^{128}}$), and computing $2n$ hashes. The cost of the VOLE dominates that of performing n hashes, so for sufficiently large set sizes ($n \gg 2^{20}$), the protocol of [RS21] should become roughly 30% more efficient than our protocol computation-wise. For smaller sets (e.g. $n \approx 2^{16}$), the cost of setting up the OKVS becomes more significant, requiring around $20n$ field multiplications over $\mathbb{F}_{2^{128}}$, hence the computational efficiency of our protocol becomes roughly on par with that of [RS21]. Of course, real runtimes can vary due to e.g. cache misses, so these estimations should only be viewed as a first order approximation indicating that the computational efficiency of our protocols is close to that of [RS21] (but likely slightly larger).
In terms of computation, the main computational overhead comes from performing N polynomial interpolations of *only degree-d* polynomials. Based on our analysis, to achieve $2^{-\kappa} = 2^{-40}$ probability of insertion failure, the following parameters can be chosen:

- $N = 0.65n$ and $d = 3$ for generalized Cuckoo hashing (GCH),
- $N = 0.33n$ and $d = 4$ for two-choice hashing,
- $N = n/10$ and $d \approx 46$ for simple hashing.

As the above illustrates, the cost of performing N polynomial interpolations will be very small for GCH, two-choice hashing, but becomes higher for simple hashing (though performing $n/10$

degree-46 interpolations remains reasonably fast).

3.4.3 Malicious PSI from mOPRF

In this section, we propose a maliciously secure PSI protocol based on our BaRK-OPRF (Section 3.4.1) and simple hashing combining a permutation-based hash function. The PSI protocol is shown in Figure 3.4 and its security against a corrupted adversary is proven in Theorem 3.4.2. The estimated overhead communication cost of this PSI is $Nd(\ell - \log N) + (\lambda + \log n)n + o(n)$. Observe that the PSI protocol in Section 3.4.2 is insecure against malicious settings since the general hashing scheme does not allow the simulation in ideal world. To handle this we use simple hashing schemes with only one permutation-based hash function. This protocol is constructed from the natural approach used recently in [PRT+19; PRT+20; CM20; RS21], i.e., Alice (a sender) and Bob (a receiver) invoke the $\mathcal{F}_{\mathsf{oprf}}$ then Bob gets the PRF values on his input and Alice enables to compute the PRF on any input so Alice computes on her input after that she sends these PRF values to Bob; Bob compares and outputs the intersection.

Figure 3.4: Our malicious PSI protocol based on $\mathcal{F}_{\mathsf{oprf}}$

PARAMETERS:

- Alice (sender) and Bob (receiver) have respectively input set $X = \{x_1, x_2, \dots, x_n\} \in \mathbb{F}_p$ and $Y = \{y_1, y_2, \dots, y_n\} \in \mathbb{F}_p$, all elements of bit-length ℓ.
- A random hash functions $h : \{0,1\}^* \to [N]$.
- A Permutation-based hashing $\mathsf{Per}_{h,X}$ maps a set X to table $\mathcal{B}_X$ consisting of N bins such that each bin has d slots where $Nd > |X|$, and $d = O(1)$. Denote $\mathsf{Per}(x) := (i, x')$ where x' is the stored value of x in bin i which defined by h and x then $\mathsf{Per}^{-1}(i, x') = x$.

PROTOCOL:

1. Bob uses Per to map Y to $\mathcal{B}_Y$, for each empty slot in each bin $\mathcal{B}_Y[i]$, put here a dummy item of length $\ell - \log N$.
2. Alice sends $(\mathsf{sender}, \mathsf{id})$ and Bob sends $(\mathsf{receiver}, \mathsf{id}, \mathcal{B}_\mathsf{Y})$ to $\mathcal{F}_{\mathsf{oprf}}$ then
   - Bob receives the $Y' = \{F(i, y') \mid y' \in \mathcal{B}_Y[i]\}_{i \le N}$.
3. For each $x \in X$, Alice queries x to $\mathcal{F}_{\mathsf{oprf}}$ with corresponding input (i, x') such that $\mathsf{Per}(x) = (i, x')$, then Alice gets $F(i, x')$. Alice sends to Bob
$$U = \{F(i, x') \mid x \in X \wedge \mathsf{Per}(x) = (i, x')\}$$
4. Now for each $y \in Y$, $\mathsf{Per}(y) = (i, y')$, if $F(i, y') \in U$ then Bob outputs y as an element in the intersection.

Intuitively, in a malicious setting, when the sender is corrupted, the simulation needs to extract the sender's input set X from the queries to $\mathcal{F}_{\mathsf{oprf}}$ and the set U. Denote $F(y) := F(i, y')$ where $\mathsf{Per}(y) = (i, y')$ and the set of all elements queried to $\mathcal{F}_{\mathsf{oprf}}$ is X' where $n' = |X'|$. The extraction procedure is that $X = \{x \in \mathbb{F}_p \mid x \in X' \wedge F(x) \in U\}$. Observe that if there exist two distinct elements $x_1, x_2 \in X'$ such that $F(x_1) = F(x_2) \in U$ then more than one element is extracted to X. The probability of existing collision is $2^{-v+2\log n'}$ then one approach to avoid collision is choosing $v = 2\lambda$. However, when $v = 2\lambda$, the overhead communication cost significantly increases.

Therefore, another approach is that Sim only extracts elements $x \in X'$ if its PRF is distinct and appears in U, i.e., $x \in X'$ such that $F(x) \in U$ and $\nexists x' \in X'$ where $F(x) = F(x')$. [RS21] proposed this simulation and claimed that if the output domain of PRF $v = \lambda$ then this simulation is correct and can not be distinguishable from the real protocol. We point out the proof of [RS21] has a gap and show that the output of PRF should be $\lambda + \log n$.

Indeed, if there exist some $x_1, x_2 \in X'$ such that $F(x_1) = F(x_2)$ then Sim only needs to extract x_1, x_2 when one of them is in Y. Let assume $x_1 \in Y$, the probability of $F(x_2) = F(y)$ for some $y \in Y$ is $2^{-v+\log(n_Y)}$ since Y is first fixed before the function F is sampled. [RS21] shows $n_Y = O(\lambda)$ then the security can hold if $v = \lambda$. However, this should be $v = \lambda + \log n_Y$ since $n_Y = O(\text{poly}(\lambda))$ instead of $O(\lambda)$. In particular, PSI protocols in [RS21] are targeted on large input set because of the usage of vector OLE.

Theorem 3.4.2. *The PSI protocol on Figure 3.4 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{psi}}$ over the field $\mathbb{F}_p$ for set size n and malicious set size $n_X = n$, $n_Y = Nd$ with statistical security against malicious adversaries in $\mathcal{F}_{\mathsf{oprf}}$ hybrid model.*

Alice is corrupted. Sim interacts with Alice as below:

- Sim emulates $\mathcal{F}_{\mathsf{oprf}}$, extracts the set X' containing all elements queried to $\mathcal{F}_{\mathsf{oprf}}$. Then Sim defines a set
$$X^* = \{x \in X' \mid \nexists x' \in X', x \neq x' : F(x) = F(x')\}$$
- From the set U, Sim defines
$$X := \{x \in X^* \mid F(x) \in U\}$$
then inputs X to $\mathcal{F}_{\mathsf{psi}}$ and obtains the set $X \cap Y$.

The simulation is indistinguishable from the real protocol by following hybrids:

- *Hybrid 0.* The same as real protocol. Bob is honest with his input Y and $\mathcal{F}_{\mathsf{oprf}}$ is executed perfectly.
- *Hybrid 1.* Sim emulates the functionality $\mathcal{F}_{\mathsf{oprf}}$, receiving the queries (i, x') from $\mathcal{A}$. For each query (i, x'), Sim determines an element x such that $\mathsf{Per}(x) = (i, x')$ then let X' be the set containing of all such elements. Sim defines
$$X^* = \{x \in X' \mid \nexists x' \in X, x \neq x' : F(x) = F(x')\}$$
- *Hybrid 2.* After $\mathcal{A}$ sends the set U, on behalf of Bob, Sim gets U then defines the set
$$X := \{x \in X^* \mid F(x) \in U\}$$
Sim will abort if there exist two distinct values $x_1, x_2 \in X'$ such that $F(x_1) = F(x_2)$ and one of them being in Y. Since Y is first fixed and then the function F is sampled then w.l.o.g assume $x_1 \in Y$, the probability of $F(x_2) = F(y)$ for some $y \in Y$ is $n_Y/2^v$ which is negligible when $v = \lambda + \log(n_Y)$.
Observe that X^* can have more than n elements but $|X|$ is always less than n since $|U| = n$ and X^* contains only elements with distinct PRF values.
- *Hybrid 3.* Sim inputs X to $\mathcal{F}_{\mathsf{psi}}$ and obtains the set $X \cap Y$. Sim outputs it as the output of a honest Bob.

Bob is corrupted. Sim interacts with Bob as below:

- Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ functionality. Sim gets $\mathcal{B}_Y$ being the input set of receiver to $\mathcal{F}_{\mathsf{oprf}}$ and then Sim samples a uniformly random sequence $Z = \{Z_{i,y'}\}_{i \le N}$ as the output PRF value of the set $\mathcal{B}_Y$, where $Z_{i,y'}$ corresponds to $y' \in \mathcal{B}_Y[i]$.
- From $\mathcal{B}_Y$, Sim extracts Y, inputs Y being the receiver's input to $\mathcal{F}_{\mathsf{psi}}$, receiving $I = X \cap Y$.
- Sim sends to Bob a set U containing of
  - The set of $|I|$ values: $\{Z_{i,y'} | y \in I \wedge \mathsf{Per}(y) = (i, y')\}$.
  - $n - |I|$ uniformly random values in $\{0,1\}^v \setminus Z$.

The simulation can not distinguish from the real protocol by following hybrids:

- *Hybrid 0.* The same as real protocol except Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ functionality. Sim gets a set $\mathcal{B}_Y$ as the input set of Bob to $\mathcal{F}_{\mathsf{oprf}}$ and then Sim samples a uniformly random sequence $Z = \{Z_{i,y'}\}_{i \le N}$ as the output PRF value of the set $\mathcal{B}_Y$, where $Z_{i,y'} \in \{0,1\}^v$ corresponds to $y' \in \mathcal{B}_Y[i]$.
- *Hybrid 1.* Sim computes the set
$$Y = \{\mathsf{Per}^{-1}(i, y') \mid \forall y' \in \mathcal{B}_Y[i], i \in [1, N]\}$$
Note that $|Y| \le Nd$ since corrupted Bob is only allowed to input up to Nd elements to $\mathcal{F}_{\mathsf{oprf}}$. Sim inputs Y to $\mathcal{F}_{\mathsf{psi}}$ on behalf of receiver, receiving $I = X \cap Y$.
- *Hybrid 2.* Sim sends to Bob the set U of n elements consisting of
  - The set of $|I|$ values: $\{Z_{i,y'} | y \in I \wedge \mathsf{Per}(y) = (i, y')\}$.
  - $n - |I|$ uniformly random values in $\{0,1\}^v \setminus Z$.

  The simulation can not be distinguishable from the real protocol since the output of F is pseudorandom over $\{0,1\}^v$ while $\{0,1\}^v \setminus Z = O(2^\lambda)$. This concludes the proof.

In general, the malicious PSI (Figure 3.4) has a communication cost that depends on the security parameter λ and is dominated by λn. We now present a new PSI protocol that is secure in malicious setting via a dual execution while its communication cost only depends on the statistic parameter κ and the set size n. The idea of using a dual execution has been used in [RR17] but when combining this with our BaRK-OPRF it achieves efficient results, i.e., the total communication cost is only $2Nd(\ell - \log N) + n(\kappa + \log n) + o(n)$.

3.4.4 Malicious Dual Execution

Intuitively, we execute $\mathcal{F}_{\mathsf{oprf}}$ twice while Alice and Bob have the same role. The main idea behind our approach is as follows:

- Alice with the input set X invokes $\mathcal{F}_{\mathsf{oprf}}$ as a receiver to learn the set of PRF values of each element in X. Denote $F_A(x)$ for all $x \in X$. While Bob queries Y to $\mathcal{F}_{\mathsf{oprf}}$ to get $F_A(y)$.
- Alice and Bob follow exactly the previous steps with the roles reversed. While Bob with input set Y can only get the set of PRF values $F_B(y)$. Alice queries X to $\mathcal{F}_{\mathsf{oprf}}$ and obtains $F_B(x)$ for all $x \in X$.
- Alice computes the set $E = \{F(x) := F_A(x) \oplus F_B(x) \mid x \in X\}$ and sends it to Bob. Bob computes $F(y)$ for all $y \in Y$ then check whether it is in E or not. Formally, Bob outputs
$$\{y \in Y \mid F(y) \in E\}$$

Informally, there are two crucial properties to guarantee the correctness of this construction:

- Since Alice only knows $F_A(x)$ for $x \in X$ and for $x \notin X$ the PRF value $F_A(x)$ is pseudorandom in the view of Alice; Alice only can obtain the correct value of $F(x)$ for $x \in X$. Similarly, Bob can compute correctly only $F(y)$ for $y \in Y$.
- The outputs of PRF are pseudorandom, with a high probability

$$\forall x \in X, \forall y \in Y : F(x) = F(y) \Leftrightarrow x = y$$

For security, this scheme can be proven against a malicious adversary since it is possible to extract the input set of both sender and receiver based on $\mathcal{F}_{\mathsf{oprf}}$ ideal functionality. We leave the detailed construction in Figure 3.5 and the formal proof in the Theorem 3.4.3. It requires the output domain of PRF is $v = \kappa + 2\log(Nd) \approx \kappa + 2\log n$ so that the probability of existing two distinct elements $x \in X$ and $y \in Y$ such that $F(x) = F(y)$ is negligible $2^{-\kappa}$ where $|X|, |Y| \leq Nd$.

Figure 3.5: Our second malicious PSI protocol based on $\mathcal{F}_{\mathsf{oprf}}$ via dual execution

PARAMETERS:

- Alice (sender) and Bob (receiver) have respectively input set $X = \{x_1, x_2, \ldots, x_n\} \in \mathbb{F}_p$ and $Y = \{y_1, y_2, \ldots, y_n\} \in \mathbb{F}_p$, all elements of bit-length ℓ.
- A random hash functions $h : \{0,1\}^* \to [N]$.
- A Permutation-based hashing $\mathsf{Per}_{h,X}$ maps a set X to table $\mathcal{B}_X$ consisting of N bins such that each bin has d slots where $Nd > |X|$, and $d = O(1)$. Denote $\mathsf{Per}(x) := (i, x')$ where x' is the stored value of x in bin i which defined by h and x then $\mathsf{Per}^{-1}(i, x') = x$.

PROTOCOL:

1. Preprocessing phase.
   - Alice and Bob use Per to map their own set to $\mathcal{B}_X, \mathcal{B}_Y$ respectively.
   - For each empty slot in each bin $\mathcal{B}_X[i]$ and $\mathcal{B}_Y[i]$, put here a dummy item of length $\ell - \log N$.
2. Alice sends $(\mathsf{receiver}, \mathsf{id}, \mathcal{B}_\mathsf{X})$ and Bob sends $(\mathsf{sender}, \mathsf{id})$ to $\mathcal{F}_{\mathsf{oprf}}$ then
   - Alice receives the $X' = \{F_A(x) \mid x \in X\}$ where $F_A(x) := PRF(\mathsf{Per}(x))$.
   - For each $y \in Y$, Bob queries $\mathsf{Per}(y)$ to $\mathcal{F}_{\mathsf{oprf}}$ and get $F_A(y)$.
3. Similarly, Alice sends $(\mathsf{sender}, \mathsf{id})$ and Bob sends $(\mathsf{receiver}, \mathsf{id}, \mathcal{B}_\mathsf{Y})$ to $\mathcal{F}_{\mathsf{oprf}}$ then
   - Bob receives the $Y' = \{F_B(y) \mid y \in Y\}$ where $F_B(y) := PRF(\mathsf{Per}(y))$.
   - For each $x \in X$, Alice queries $\mathsf{Per}(x)$ to $\mathcal{F}_{\mathsf{oprf}}$ and get $F_B(x)$.
4. Alice now sends to Bob the set

$$E = \{F_A(x) \oplus F_B(x) \mid x \in X\}$$

Now for each $y \in Y$, if $F_A(y) \oplus F_B(y) \in E$ then Bob outputs y as an element in the intersection.

Theorem 3.4.3. *The PSI protocol on Figure 3.5 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{psi}}$ (Figure 2.9) over the field $\mathbb{F}_\mathsf{p}$ for set size n and malicious set size $n_X = n, n_Y = Nd$ with statistical security against malicious adversaries in $\mathcal{F}_{\mathsf{oprf}}$ hybrid model.*

Proof. **Alice is corrupted**. Sim interacts with Alice as below:

- When $\mathcal{A}$ plays the role of receiver in $\mathcal{F}_{\mathsf{oprf}}$, Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ to get $\mathcal{B}_X$. Sim samples a uniformly random sequence $Z = \{Z_{i,x'}\}_{i \leq N}$ as the output PRF values of the set $\mathcal{B}_X$, where $Z_{i,x'}$ corresponds to $x' \in \mathcal{B}_X[i]$.
- From $\mathcal{B}_X$, Sim extracts $X^* = \{\mathsf{Per}^{-1}(i, x') \mid \forall x' \in \mathcal{B}_X[i], i \in [1, N]\}$.
- When $\mathcal{A}$ plays the role of sender in $\mathcal{F}_{\mathsf{oprf}}$, Sim defines a set $\tilde{X}$ containing of all elements x such that $\mathsf{Per}(x)$ has been queried to $\mathcal{F}_{\mathsf{oprf}}$; Sim samples a uniformly random values as $\{F_B(x)\}_{x \in \tilde{X}}$ and give them to $\mathcal{A}$.
- On behalf of Bob, Sim receives the set E from $\mathcal{A}$, Sim defines the set

$$X = \{x \in X^* \cap \tilde{X} \mid \mathsf{Per}(x) = (i, x') \wedge F_B(x) \oplus Z_{i,x'} \in E\}$$

  Sim sends X to $\mathcal{F}_{\mathsf{psi}}$ functionality, receiving $I := X \cap Y$.

This simulation can not distinguish from the real protocol by the following hybrids:

- *Hybrid 0.* The same as real protocol, Bob is honest with the input set Y, and $\mathcal{F}_{\mathsf{oprf}}$ is implemented honestly.
- *Hybrid 1.* Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ then functionality then
  - When Alice is a receiver in $\mathcal{F}_{\mathsf{oprf}}$, Sim learns $\mathcal{B}_X$ which is the input set of Alice and then Sim samples a uniformly random sequence $Z = \{Z_{i,x'}\}_{i \leq N}$ as the output PRF values of the set $\mathcal{B}_X$, where $Z_{i,x'} \in \{0,1\}^v$ corresponds to $x' \in \mathcal{B}_X[i]$.
  - When Alice is a sender in $\mathcal{F}_{\mathsf{oprf}}$, Sim defines a set $\tilde{X}$ containing of all elements x such that $\mathsf{Per}(x)$ has been queried to $\mathcal{F}_{\mathsf{oprf}}$; Sim samples a uniformly random values as $\{F_B(x)\}_{x \in \tilde{X}}$ and give them to $\mathcal{A}$.

  This hybrid is indistinguishable from the previous hybrid since the the outputs of PRF are pseudorandom.
- *Hybrid 2.* Sim computes a set

$$X^* = \{\mathsf{Per}^{-1}(i, x') \mid \forall x' \in \mathcal{B}_X[i], i \in [1, N]\}$$

  Note that, $|X^*| \leq Nd$.
- *Hybrid 3.* On behalf of Bob, Sim gets the set E sending from Alice. Sim defines the set

$$X = \{x \in X^* \cap \tilde{X} \mid \mathsf{Per}(x) = (i, x') \wedge F_B(x) \oplus Z_{i,x'} \in E\}$$

  This hybrid will abort if there exist some $x_1, x_2 \in X$ such that

$$F_B(x_1) \oplus Z_{i,x'_1} = F_B(x_2) \oplus Z_{i,x'_2}$$

  Observe that $\{Z_{i,x'}\}_{i \leq N}$ is first fixed for elements in X^* and then the function F_B is sampled. Therefore, the probability of aborting is bounded to $(Nd)^2/2^v$ which is negligible $O(2^{-\kappa})$ when $v = \kappa + 2\log n$. This deduces that $|X| \leq n$.
- *Hybrid 4.* Sim inputs X to $\mathcal{F}_{\mathsf{psi}}$ functionality, receiving $X \cap Y$ and then outputs it as the output of honest Bob.

Bob is corrupted. Sim interacts with Bob as below:

- Similarly, Sim plays the role of $\mathcal{F}_{\mathsf{oprf}}$ to get $\tilde{Y}, \mathcal{B}_Y$ then Sim samples a uniformly random sequence $T = \{T_{i,y'}\}_{i \leq N}$ as the output PRF values of the set $\mathcal{B}_Y$, where $T_{i,y'} \in \{0,1\}^v$ corresponds to $y' \in \mathcal{B}_Y[i]$.
- From $\mathcal{B}_Y$, Sim extracts $Y^* = \{\mathsf{Per}^{-1}(i, y') \mid \forall y' \in \mathcal{B}_Y[i], i \in [1, N]\}$.
- Sim emulates $\mathcal{F}_{\mathsf{psi}}$ functionality with input set $Y := Y^* \cap \tilde{Y}$, receiving $I := X \cap Y$.
- Sim sends to Bob the set containing of
  - $F_A(y) \oplus T_{i,y'}$ for $y \in I$.
  - $n - |I|$ uniformly random values in $\{0,1\}^v \setminus T$.

This simulation is indistinguishable from the real protocol by the following hybrids:

- *Hybrid 0.* The same as real protocol, Alice is honest with the input set X, and $\mathcal{F}_{\mathsf{oprf}}$ is implemented honestly.
- *Hybrid 1.* Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ functionality then
  - When Bob is a sender in $\mathcal{F}_{\mathsf{oprf}}$, Sim extracts the set $\tilde{Y}$ such that for $y \in Y$, $\mathsf{Per}(y)$ has been queried to $\mathcal{F}_{\mathsf{oprf}}$; Sim give to Bob a uniformly random sequences defined as $\{F_A(y)\}_{y \in \tilde{Y}}$.
  - When Bob is a receiver in $\mathcal{F}_{\mathsf{oprf}}$, Sim learns $\mathcal{B}_Y$ which is the input set of Bob and then Sim samples a uniformly random sequence $T = \{T_{i,y'}\}_{i \leq N}$ as the output PRF values of the set $\mathcal{B}_Y$, where $T_{i,y'} \in \{0,1\}^v$ corresponds to $y' \in \mathcal{B}_Y[i]$.
- *Hybrid 2.* From $\mathcal{B}_Y$, Sim computes a set
  $$Y^* = \{\mathsf{Per}^{-1}(i, y') \mid \forall y' \in \mathcal{B}_Y[i], i \in [1, N]\}$$
  Sim emulates $\mathcal{F}_{\mathsf{psi}}$ functionality with input set $Y = \tilde{Y} \cap Y^*$, receiving $I := X \cap Y$.
- *Hybrid 3.* Sim sends to Bob the set E containing of
  - T = $\{F_A(y) \oplus T_{i,y'} \mid \forall y \in I\}$.
  - $n - |I|$ uniformly random values in $\{0,1\}^v \setminus T$.

  This hybrid is indistinguishable from the real protocol since in the view of Bob E consists of the XOR of pseudorandom values and arbitrary values. This concludes the proof.

□

3.5 Standard PSI from subfield-ring OLE

In this section, we describe a new PSI protocol, which builds upon a (simple variant of) a pseudorandom correlation generator for the ring-OLE correlation [BCG+20b]. Our protocol enjoys a number of important features: it is in the *standard model*, achieves *malicious security* at essentially no cost, has *low communication* (competitive even with the best maliciously secure PSI protocols in the random oracle model), and reasonable computation (albeit superlinear in n). Our protocol can also be generalized to a powerful notion of *batch non-interactive PSI*, where (after a small logarithmic-cost preprocessing step with each server) a client can broadcast a single encoding of his database, and then obtain the intersection with any of the server databases at any time after receiving a single message from this server. We believe that this functionality itself is of independent interest.

3.5.1 Semi-Honest Batch Non-Interactive PSI from Subfield Ring-OLE

We describe a new PSI scheme in the semi-honest model. Our protocol enjoys two interesting features: (1) it is in the standard model, and (2) it is a *batch non-interactive* protocol, a useful communication pattern that we describe afterward. The full construction is represented on Figure 3.6.

Theorem 3.5.1. *The PSI protocol on Figure 3.6 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{psi}}$ over the field $\mathbb{F}_{\mathsf{p}}$ with set size n and malicious set size $n' = n_X = n_Y = 2n$, with statistical security against augmented semi-honest adversaries in the $\mathcal{F}_{\mathsf{sOLE}}$ hybrid model.*

The proof is supported by the lemma below.

Lemma 3.5.1. *Let $F(X)$ be a degree-n polynomial, $q = p^t$ where p is a prime, $P(x) \in \mathcal{R}_{\mathsf{p}} = \mathbb{F}_{\mathsf{p}}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$ be an arbitrary polynomial of degree n and $R(x) \in \mathcal{R}_{\mathsf{q}} = \mathbb{F}_{\mathsf{p}^{\mathsf{t}}}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$ be a uniformly random polynomial of degree n. Then*

$$\Pr[\gcd(P(x), R(x)) \neq 1] \leq n^2/q.$$

Proof. $\gcd(P(x), R(x)) = 1$ iff $P(x)$ and $R(x)$ share no common root. A random polynomial over $\mathcal{R}_p$ of degree n has at most n roots, which are distributed uniformly; hence, each root of $R(x)$ is equal to a root of $P(x)$ with probability at most $1 - n/q$. Therefore:

$$\begin{aligned}\Pr[\gcd(P(x), R(x)) \neq 1] &= 1 - \Pr[\gcd(P(x), R(x)) = 1] \\ &= 1 - (1 - n/q)^n \leq n^2/q \text{ (union bound).}\end{aligned}$$

□

Proof. We first show that the protocol is correct with probability at least $1 - n^2/q = 1 - 2^{-\kappa}$ using $q = \kappa + 2\log n$. It follows from the description of the protocol that $u = p_A b_0 + p_B b_0'$, where b_0, b_0' are uniformly random degree-n polynomials over $\mathcal{R}_q$. Then $x \in A \cap B$ implies $p_A(x) = 0 \wedge p_B(x) = 0$, which implies $u(x) = 0$. In the other direction, it holds by Lemma 3.5.1 that the probability that p_A and b_0' share a common root (i.e. $\gcd(p_A, b_0') \neq 1$) is at most n^2/q. Then,

$$\begin{aligned} x \in I \quad &\Longrightarrow \quad p_A(x) = 0 \wedge U(x) = 0 \\ &\Longrightarrow \quad p_A(x) = 0 \wedge p_B(x) \cdot b_0'(x) = u(x) - p_A(x) \cdot b_0(x) = 0 \\ &\Longrightarrow \quad p_A(x) = 0 \wedge p_B(x) = 0 \text{ (since } \gcd(p_A, r_B^0) = 1 \text{ and } b_0' \neq 0 \text{ w.h.p}), \end{aligned}$$

hence $I \subseteq A \cap B$, which concludes the proof.

We now turn our attention to security. We use the following fact: given any set S, denoting by $p_S \in \mathcal{R}_p$ the polynomial whose set of roots is S, it holds that

$$u = p_A \cdot b_0 + p_B \cdot b_0' = p_{A\cap B} \cdot \left(p_{A\setminus B} \cdot b_0 + p_{B\setminus A} \cdot b_0'\right),$$

where $p_{A\setminus B}, p_{B\setminus A} \in \mathcal{R}_p^2$ are two polynomials of degree at most n and $\gcd(p_{A\setminus B}, p_{B\setminus A}) = 1$.

Alice is corrupted. We describe a simulator Sim which emulates Bob:

- (Setting up the correlation) Sim emulates the functionality $\mathcal{F}_{\mathsf{sOLE}}$: when Alice queries $\mathcal{F}_{\mathsf{sOLE}}$, Sim samples and sends two random polynomials $(a, s_A) \leftarrow\$ \mathcal{R}_p \times \mathcal{R}_q$.

- (Client set encoding) upon receiving t_A, Sim defines $p_{\tilde{A}} = a - t_A$ and obtains the roots $\tilde{A}$ of $p_{\tilde{A}}$ (note that Bob is 'augmented semi-honest', so $p_{\tilde{A}}$ is well-formed, but $\tilde{A}$ might differ from A). Sim queries the PSI functionality $\mathcal{F}_{\mathsf{psi}}$ on behalf of Alice with input set $\tilde{A}$ and obtains $\tilde{A} \cap B$.

- (Server-to-client message) Sim generates a random degree-n polynomial $b_1 \in \mathcal{R}_q$ and picks $v \leftarrow\!\!\$\ \mathcal{R}_q$. Sim sets $u \leftarrow p_{\tilde{A}\cap B} \cdot v$ and $t_B \leftarrow u + p_{\tilde{A}} b_1 \cdot X^n - s_A$. Sim sends (b_1, t_B) to Alice.

We prove that the simulated protocol is indistinguishable from an honest execution through a sequence of hybrids: in the first game, Sim simulates $\mathcal{F}_{\mathsf{sOLE}}$ honestly, and behaves as a honest Bob (using p_B) otherwise. Sim also extracts $p_{\tilde{A}}$ from $t_A = a - p_{\tilde{A}}$ and queries A to the PSI functionality on behalf of Alice in the ideal world, obtaining $\tilde{A} \cap B$. This game is perfectly indistinguishable from an honest execution of the protocol. Then, in the second game, Sim computes t_B as $p_{\tilde{A}\cap B} \cdot v + p_{\tilde{A}} b_1 \cdot X^n - s_A$, where v, b_1 are uniformly random degree-n polynomials over $\mathcal{R}_q$.

It remains to show that the second game is indistinguishable from the first game. Recall that $u = p_{\tilde{A}\cap B} \cdot \left(p_{\tilde{A}\setminus B} \cdot b_0 + p_{B\setminus\tilde{A}} \cdot b_0'\right)$ when the parties play honestly, but Alice uses input $\tilde{A}$. From the viewpoint of Alice, both b_0 and b_0' are distributed as uniformly random degree-n polynomials over $\mathcal{R}_q$. By Lemma 3.5.1, this implies that $p_{\tilde{A}\setminus B} \cdot b_0 + p_{B\setminus\tilde{A}} \cdot b_0'$ is distributed as a uniformly random degree-n polynomial $v \in \mathcal{R}_q$. By construction, $u = t_B - p_{\tilde{A}} b_1 \cdot X^n + s_A$, hence t_B is distributed as $p_{\tilde{A}\cap B} \cdot v + p_{\tilde{A}} b_1 \cdot X^n - s_A$ where v is a random degree-n polynomial over $\mathcal{R}_q$. This concludes the proof.

Bob is corrupted. We describe a simulator Sim which emulates Bob:

- (Setting up the correlation) Sim emulates the functionality $\mathcal{F}_{\mathsf{sOLE}}$: when Bob queries $\mathcal{F}_{\mathsf{sOLE}}$, Sim samples and sends two random polynomials $(b, s_B) \leftarrow\!\!\$\ \mathcal{R}_q^2$.

- (Client set encoding) Sim sends a uniformly random polynomial $t_A \leftarrow\!\!\$\ \mathcal{R}_q$.

- (Server-to-client message) Upon receiving (b_1, t_B), Sim computes $w \leftarrow t_B - s_B + t_A b$ and defines $\bar{B}$ to be the set of roots of w (since Bob is augmented semi-honest, $w = p_B b_0'$ has at least n roots). Sim queries $\bar{B}$ to $\mathcal{F}_{\mathsf{psi}}$ on behalf of Bob in the ideal world, and gets $\bar{B} \cap A$. Sim outputs $\bar{I} \leftarrow \bar{B} \cap A$.

t_A is distributed exactly as in the honest protocol by construction. It remains to show that Sim's simulated output $\bar{I}$ is the same as the honest output $\tilde{I} = A \cap \tilde{B}$ with overwhelming probability, where $\tilde{B}$ is the input used by Bob when computing $w = p_{\tilde{B}} b_0'$ (which can differ from Bob's real input B). This follows from Lemma 3.5.1: since Bob is (augmented) semi-honest, $w = p_{\tilde{B}} b_0'$ where b_0' is a uniformly random degree-n polynomial, hence by Lemma 3.5.1, $\gcd(p_A, b_0') = 1$ with probability at least $1 - n^2/q$. Therefore, with probability at least $1 - n^2/q = 1 - 2^{-\kappa}$ (using $q = \kappa + 2\log n$), it holds that $\bar{I} = A \cap \bar{B} = A \cap (\tilde{B} \cup \mathsf{roots}(b_0')) = A \cap \tilde{B} = \tilde{I}$. This concludes the proof. $\square$

Above, "augmented semi-honest security" refers to a strengthening of honest-but-curious corruption where the adversary is allowed to change the corrupted parties' inputs. This is a standard strengthening of semi-honest security, which has been argued to better capture real-world security [HL10]. It will also facilitate upgrading security to the malicious setting later on.

Batch non-interactivity. To securely realize the functionality $\mathcal{F}_{\mathsf{sOLE}}$, we rely on the PCG-based protocol of [BCG+20b] (using a straightforward adaptation to the subfield setting), which is secure under the ring-LPN assumption. Interestingly, instantiating the subfield ring OLE this way allows to import a powerful feature of the PCG of [BCG+20b], which is its *programmability*: when generating a ring-OLE correlation, the receiver can ensure that her output a remains *identical* across multiple instances of the protocol with different parties.

This feature enables the following communication structure: after a short (logarithmic-communication) interaction with N servers, a client, playing the role of Alice with input set A, can broadcast a single compact encoding of her dataset to all the servers (with input sets $B_1 \cdots B_N$). Afterwards, each server B_i can at any time send a single message m_i to Alice, from which she can recover $A \cap B_i$ without further interaction. To our knowledge, this batch non-interactive communication pattern was never achieved by any prior proposal; we believe that it can make our protocol appealing in realistic scenarios.

More concretely, after a logarithmic-communication preprocessing phase where Alice sets up PCG seeds with each of the servers, Alice broadcasts the value $t_A = a - p_A$ to everyone, which communicates $2n \log p \approx 2\ell n$ bits. This message can be seen as a compact public encoding of her dataset (it is only twice as large as Alice's set). Afterwards, each server can complete the protocol of Figure 3.6 by sending a single message (b_1, t_B) to the receiver, of length $3n \log q \approx 3(\kappa + 2\log n)n$, from which the receiver can locally recover $X \cap X_i$. Furthermore, using the encoding technique of [TLP+17], the $\kappa + 2\log n$ term can be reduced to $\kappa + \log n$ (the improvement is based on the observation that for an appropriate ordering, n random elements of a set of size $2^{\kappa+2\log n}$ are on average at distance $2^{\kappa+\log n}$ for each other, hence the cost of transmitting them can be reduced to essentially $\kappa + \log n$ per element by sending the distance between consecutive elements instead).

Efficiency. The communication cost of protocol (Figure 3.6) is $n \cdot (2\log p + 3\log q) + o(n)$ bits of communication. Here, the size of the subfield $\mathbb{F}_p$ depends only on the bitsize ℓ of the items in the sets A and B, hence we can set $\log p = \ell$. As we will see in the analysis, $\log q$ must be set to $\log q \approx \kappa + 2\log n$ to guarantee κ bits of statistical security. This leads to a total communication of $n \cdot (2\ell + 3\kappa + 6\log n) + o(n)$ bits, which is reduced to $n \cdot (2\ell + 3\kappa + 3\log n) + o(n)$ with the encoding of [TLP+17]. The $o(n)$ term above captures the cost of distributing the PCG seeds of the subfield ring-OLE (we discuss the concrete value of $o(n)$ later on, for our maliciously secure version of the protocol).

Regarding computation, the computational cost scales as $O(n \log^2 n)$ due to the fast polynomial interpolations, or as $O(n \log n)$ when using cyclotomic rings. We provide a concrete analysis of the computational cost of the maliciously secure version of our protocol in Section 3.5.2.

3.5.2 Maliciously Secure PSI in the Standard Model

In this section, we upgrade the security of our protocol to the malicious setting. Our upgrade introduces only a minimal communication overhead to the protocol, independent of the set sizes n. The full protocol is represented on Figure 3.7.

Theorem 3.5.2. *The PSI protocol on Figure 3.7 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{psi}}$ over the field $\mathbb{F}_\mathsf{p}$ with set size n and malicious set size $n' = n_X = n_Y = 2n$, with statistical security against malicious adversaries in the $\mathcal{F}_{\mathsf{sOLE}}$-hybrid model.*

Proof. We first consider the case where Alice is corrupted. The simulator Sim behaves as follows:

- He waits for the adversary to send (a, s_A) to $\mathcal{F}_{\mathsf{sOLE}}$ and receives t_A from Alice. Sim defines $p_A \leftarrow a - t_A$ and computes $\tilde{A} = \{x \in \mathbb{F}_{\mathsf{p}'} \text{s.t. } p_A(\mathsf{map}(x)) = 0\}$, and inputs $\tilde{A}$ to the PSI functionality on behalf of Alice, receiving $\tilde{A} \cap B$.
- He picks a uniformly random degree-n polynomial b_1 over $\mathcal{R}_q$ and sends b_1 to Alice.
- Upon receiving y from Alice, Sim computes $s'_A \leftarrow s_A - p_A b_1 \cdot X^n$. Then, if either (1) $p_A(0) \neq 1$ or (2) $y \neq s'_A(0)$, Sim aborts on behalf of Bob. Otherwise, Sim simulates t_B as in the augmented semi-honest model, picking $v \leftarrow\$ \mathcal{R}_q$ and setting $t_B \leftarrow p_{\tilde{A} \cap B} \cdot v + p_A b_1 \cdot X^n - s_A$. Sim sends t_B to Alice.

The analysis of this simulator is similar to the analysis in the augmented semi-honest model, up to two distinctions: first, the extracted polynomial p_A is not guaranteed to be of degree n anymore –

Figure 3.6: Augmented semi-honest PSI protocol based on ring-OLE

PARAMETERS:

- Two rings $\mathcal{R}_\mathsf{p} = \mathbb{F}_\mathsf{p}[\mathsf{X}]/\mathsf{F}(\mathsf{X}) \subseteq \mathcal{R}_\mathsf{q} = \mathbb{F}_{\mathsf{p}^\mathsf{t}}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$, where $F(X)$ has degree $2n+1$.
- The sender (Alice) and receiver (Bob) have respective input sets $A = \{a_1, a_2, \ldots, a_n\} \subset \mathbb{F}_p$ and $B = \{b_1, b_2, \ldots, b_n\} \subset \mathbb{F}_p$.
- A subfield ring-OLE in the ring $\mathcal{R}_q$ over the subring $\mathcal{R}_p$.

PROTOCOL:

1. **(Setting up the correlation)** Alice and Bob encode their sets to $p_A = \prod_{i=1}^n (X - a_i)$, $p_B = \prod_{i=1}^n (X - b_i)$ respectively, and invoke $\mathcal{F}_\mathsf{sOLE}$ to generate a subfield ring-OLE correlation over $\mathcal{R}_p, \mathcal{R}_q$: Alice receives $(a, s_A) \in \mathcal{R}_p \times \mathcal{R}_q$ and Bob receives $(b, s_B) \in \mathcal{R}_q^2$ such that $s_A + s_B = ab$.
2. **(Broadcasting the client set encoding)** Alice computes and sends $t_A = a - p_A$ to Bob.
3. **(Server-to-client message)** Bob sets $s'_B \leftarrow s_B - t_A b$. Then, Bob decomposes b as $b = b_0 + b_1 \cdot X^n$ (where b_0, b_1 are degree-n polynomials), sets $s'_B \leftarrow s_B - t_A b$, and picks a random degree-n polynomial b'_0 over $\mathcal{R}_q$. He sends b_1 and $t_B \leftarrow s'_B + p_B b'_0$ to Alice.
4. **(Output)** Alice sets $u \leftarrow t_B - p_A b_1 \cdot X^n + s_A$; note that $u = p_A b_0 + p_B b'_0$. Alice outputs the set $I = \{x \in A \mid u(x) = 0\}$.

but this corresponds to a malicious adversary using a set of larger size $n' \leq 2n$, which is allowed by the functionality. Second, we must show that Sim correctly emulates the behavior of an honest Bob when deciding to abort based on the checks (1) and (2). We show that the simulation is statistically close to the behavior of an honest Bob. To do so, we consider three cases:

Case 1: $y = s'_A(0)$ and $p_A(0) = 1$. In this case, Sim does not abort. We show that an honest Bob would not abort either:

$$\begin{aligned}
s'_A(0) + s'_B(0) &= s_A(0) + s_B(0) - (p_A(0)(b_1 \cdot X^n)(0) + t_A(0)b(0)) \\
&= s_A(0) + s_B(0) - (p_A(0)(b_1 \cdot X^n)(0) + (a(0) - p_A(0))b(0)) \\
&= a(0)b(0) - (p_A(0)(b_1 \cdot X^n)(0) + (a(0) - p_A(0))b(0)) \\
&= -(b_1 \cdot X^n)(0) + b(0) \\
&= b_0(0),
\end{aligned}$$

hence Bob does not abort.

Case 2: $p_A(0) \neq 1$. In this case, check (1) of Sim causes an abort. We show that when this is the case, Bob would abort as well with high probability, i.e., that $y \neq b_0(0) - s'_B(0)$. Indeed,

$$\begin{aligned}
b_0(0) - s'_B(0) &= b_0(0) - s_B(0) + t_A(0)b(0) \\
&= b_0(0) - a(0)b(0) + s_A(0) + t_A(0)b(0) \\
&= b_0(0) - a(0)b(0) + s_A(0) - p_A(0)b(0) + a(0)b(0) \\
&= b_0(0) \cdot (1 - p_A(0)) + s_A(0),
\end{aligned}$$

Hence if Alice manages to send $y = b_0(0) - s'_B(0)$, it implies that $(y - s_A(0)) \cdot (1 - p_A(0))^{-1} = b_0(0)$. The left-hand side is a value known to Alice, but the right-hand side is the constant coefficient of b_0, which is a uniformly random independent element of $\mathbb{F}_{\mathsf{q}}$. The probability that Alice does not cause an abort is therefore bounded by $1/q < 1/2^\kappa$, hence Bob aborts with probability at least $1 - 1/2^\kappa$.

Case 3: $p_A(0) = 1$ but $y \neq s'_A(0)$. In this case, check (2) of Sim causes an abort. As we saw in Case 1, it holds that $s'_A(0) = b_0(0) - s'_B(0)$, hence $y \neq b_0(0) - s'_B(0)$, hence Bob necessarily aborts.

Overall, Sim's emulation of Bob in the p_A check phase is $1/2^\kappa$-close to the honest game, which concludes the proof.

We now turn our attention to the case where Bob is corrupted. The simulator Sim behaves as follows:

- He waits for the adversary to send (b, s_B) to $\mathcal{F}_{\mathsf{sOLE}}$ and sends a uniformly random $t_A \leftarrow\!\!\$\ \mathcal{R}_p$ on behalf of Alice.
- Upon receiving b_1 from Bob, Sim defines $b_0 \leftarrow b - (b_1 \cdot X^n)$ (note that b_0 might not be a degree-n polynomial) and sets $s'_B \leftarrow s_B - t_A b$. He sends $y \leftarrow b_0(0) - s'_B(0)$ to Bob.
- Upon receiving t_B from Bob, Sim computes $w \leftarrow t_B - s_B + t_A b$ and defines the set $\bar{B} = \{x \in \mathbb{F}_{\mathsf{p}'} \text{s.t. } w(\mathsf{map}(x)) = 0\}$. Sim queries $\bar{B}$ to $\mathcal{F}_{\mathsf{psi}}$ on behalf of Bob in the ideal world, and gets $\bar{B} \cap A$. Sim checks whether $t_B(1) - s'_B(1) \neq 0$. It outputs $\bar{I} \leftarrow \bar{B} \cap A$ if this holds, and aborts otherwise.

The analysis is identical to that of the augmented semi-honest model (note that $|\bar{B}| \leq 2n$ by construction), up to Sim's check that $t_B(1) - s'_B(1) \neq 0$. We show that with overwhelming probability, Sim aborts if and only if the honest Bob aborts at this stage. Recall that Bob aborts if $u(1) = 0$. Then

$$\begin{aligned} u(1) &= t_B(1) - p_A(1)(b_1 \cdot X^n)(1) + s_A(1) \\ &= t_B(1) + s_A(1) \text{ since } p_A(1) = 0 \\ &= t_B(1) + a(1)b(1) - s_B(1) \\ &= t_B(1) + a(1)b(1) - s'_B(1) - t_A(1)b(1) \\ &= t_B(1) + p_A(1)b(1) - s'_B(1) \\ &= t_B(1) - s'_B(1) \text{ since } p_A(1) = 0, \end{aligned}$$

hence Sim aborts iff Bob aborts. This concludes the proof. □

Efficiency. Our malicious protocol has minimal communication overhead over our augmented semi-honest protocol. The main overhead stems from starting from a slightly larger field in which two elements can be "reserved elements". If p' is a prime power and $\ell \approx \log \mathsf{p}'$, the price to pay is therefore increasing ℓ to $\log \mathsf{p}$ where p is the smallest prime power above $\mathsf{p}' + 2$. While an exact expression would be rather tedious, for any reasonable input size this cost should be negligible (the simplest strategy is to pick $\mathsf{p}' = 2^\ell$ and $\mathsf{p} = 2^{\ell+1}$, in which case ℓ is increased by one bit, but much better encoding methods exist). Therefore, the communication remains $n \cdot (2\ell + 3\kappa + 6 \log n) + o(n)$ bits, or $n \cdot (2\ell + 3\kappa + 3 \log n) + o(n)$ with the encoding of [TLP+17].

Computation cost. Note that our standard model protocol shares with our other protocols the feature of having communication independent of λ. Our protocol requires more computation compared to the best ROM-based protocols, due to its use of polynomial interpolation. However, it still allows for very fast PSI computation (we estimate a few seconds to compute the intersection between databases of size 2^{20}, on one core of a standard laptop). Concretely, the protocol requires only

Figure 3.7: Maliciously secure PSI protocol in the $\mathcal{F}_{\mathsf{sOLE}}$-hybrid model

PARAMETERS:

- A field $\mathbb{F}_{\mathsf{p}'}$ and two rings $\mathcal{R}_\mathsf{p} = \mathbb{F}_\mathsf{p}[\mathsf{X}]/\mathsf{F}(\mathsf{X}) \subseteq \mathcal{R}_\mathsf{q} = \mathbb{F}_{\mathsf{p}^t}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$, where $F(X)$ has degree $2n+1$ and $|\mathbb{F}_{\mathsf{p}'}| \leq |\mathbb{F}_\mathsf{p}| - 2$. map is an efficient (and efficiently invertible) injective mapping, with $\mathsf{map}(\mathbb{F}_{\mathsf{p}'}) \subseteq \mathbb{F}_\mathsf{p} \setminus \{0,1\}$.
- The sender (Alice) and receiver (Bob) have respective input sets $A = \{a_1, a_2, \ldots, a_n\} \subset \mathbb{F}_{\mathsf{p}'}$ and $B = \{b_1, b_2, \ldots, b_n\} \subset \mathbb{F}_{\mathsf{p}'}$.
- A subfield ring-OLE in the ring $\mathcal{R}_q$ over the subring $\mathcal{R}_p$.

PROTOCOL:

1. **(Setting up the correlation)** Alice and Bob encode their sets to $p_A = c \cdot (X-1) \cdot \prod_{i=1}^n (X - \mathsf{map}(a_i))$ with $c = -(\prod_{i=1}^n(-\mathsf{map}(a_i)))^{-1}$ (note that this guarantees $p_A(0) = 1$ and $p_A(1) = 0$) and $p_B = \prod_{i=1}^n (X - \mathsf{map}(b_i))$ respectively. Alice and Bob invoke $\mathcal{F}_{\mathsf{sOLE}}$ to generate a subfield ring-OLE correlation over $\mathcal{R}_p, \mathcal{R}_q$: Alice receives $(a, s_A) \in \mathcal{R}_p \times \mathcal{R}_q$ and Bob receives $(b, s_B) \in \mathcal{R}_q^2$ such that $s_A + s_B = ab$.
2. **(Broadcasting the client set encoding)** Alice computes and sends $t_A = a - p_A$ to Bob.
3. **(Server-to-client message)** Bob sets $s'_B \leftarrow s_B - t_A b$. Then, Bob decomposes b as $b = b_0 + b_1 \cdot X^n$ (where b_0, b_1 are degree-n polynomials), and sets $s'_B \leftarrow s_B - t_A b$. He sends b_1 to Alice.
4. **(Checking p_A)** Alice computes $s'_A \leftarrow s_A - p_A b_1 \cdot X^n$. Alice sends $y \leftarrow s'_A(0)$ to Bob. If $y \neq b_0(0) - s'_B(0)$, Bob aborts. Else, Bob picks a random degree-n polynomial b'_0 over $\mathcal{R}_q$ and sends $t_B \leftarrow s'_B + p_B b'_0$ to Alice.
5. **(Output)** Alice sets $u \leftarrow t_B - p_A b_1 \cdot X^n + s_A$; note that $u = p_A b_0 + p_B b'_0$. If $u(1) = 0$, Alice aborts; otherwise, Alice computes the set $I = \{x \in A \mid u(\mathsf{map}(x)) = 0\}$ and outputs I.

- a single degree-n polynomial interpolation, one FFT over a polynomial ring with degree-$2n$ polynomials, and 3 multiplications of degree-n polynomials for the receiver, and
- a single degree-n polynomial interpolation, one FFT as above, 2 multiplications of degree-n polynomials, and a single n-multipoint polynomial evaluation for the sender.

Furthermore, both polynomial interpolations only have to be performed over a field $\mathbb{F}$, of size $|\mathbb{F}| \approx 2^\ell$ where ℓ is the bit size of the set items (e.g. 32 or 64 bits), and the multipoint evaluation is over a field of size $\kappa + 2\log n$ bits. This stands in stark contrasts with previous state of the art protocols [PRT+19] that relied on polynomial interpolation (*on top* of using the ROM), where the interpolations and multipoint evaluations had to be performed over a very large field $\mathbb{F}$ of size $|\mathbb{F}| \approx 2^{400}$. By using a cyclotomic ring, the FFTs and polynomial multiplications are much faster than the interpolations. On Section 3.2, we compare our protocol to the current fastest maliciously secure PSI protocols [PRT+20; RT21b; RS21].

Chapter 4

Efficient Designated-Verifier Zero-Knowledge Proofs

In this chapter, we present two additional contributions to the field of designated-verifier zero-knowledge proofs (ZKPs). First, we introduce a privately verifiable ZKP for circuit satisfiability that leverages vector OLE to achieve sublinear communication while maintaining competitive efficiency. This contribution is detailed in Section 4.1 and is presented in its entirety in [BCC+24]. Second, we propose a novel and efficient designated-verifier non-interactive ZKP (DV-NIZK) based on public-key PCF-based Oblivious Transfer (OT). This work is discussed in Section 4.2 and is fully elaborated in [BCM+24].

Contents

4.1 Sublinear PCG-based ZKP for General Circuits

In this section, we present our contribution to constructing a sublinear zero-knowledge proof (ZKP) for circuit satisfiability using PCG-based VOLE. In Section 4.1.4, we introduce our compiler, which transforms ZKPs for SIMD circuits into ZKPs for arbitrary circuits. Building on this, in Section 4.1.6, we provide an instantiation of our compiler using PCG-based VOLE ZKP, achieving sublinear communication while maintaining competitive efficiency. Finally, in Section 4.1.5, we present a scalable variant of our compiler, designed specifically for scenarios where the prover has limited memory.

4.1.1 Motivation and Related Works

Assume that the verification of a statement is represented as a public circuit $C : \{0,1\}^n \to \{0,1\}$. A zero-knowledge proof (ZKP) allows a prover to convince a verifier that it possesses a witness w such that $C(w) = 0$, without the verifier learning any information beyond the circuit output. The commit-and-prove zero-knowledge (CP-ZK) paradigm is among the most flexible and modular design mechanisms for constructing ZKP. For instance, a CP-SNARK allows a prover to commit to a batch of secrets via a commitment scheme (e.g. vector commitment or polynomial commitment), then prove relations between the committed values in ZK [CFQ19; CFF+21; Lip16]. A small communication footprint is achieved when the commitment is compressing and the proof is succinct. On the other hand, schemes like VOLE-based ZKPs [BMR+21; WYK+21; YSW+21; DIO20] rely on efficient interactive commitment scheme that separately commits to wire values in the circuit, then prove the consistency between committed wire values with constant overhead. Though general VOLE-ZKs incur communication complexity linear to the circuit size, they achieve high throughput owing to the lightweight operations.

Generally, CP-ZK proof systems with sublinear communication involve two components after the batch commitment of witnesses: (1) Hadamard product of committed vectors, (2) equality of individual wires across different committed vectors. The former is used to demonstrate the correct computation of multiplication gates and the latter is used to show that the committed wire values are consistent with the circuit topology.

From SIMD-ZK to general ZK. From another perspective, the above approach can be viewed as a conversion from commit-and-prove SIMD-ZK to general ZK. Define $(B, \mathcal{C})$-SIMD circuit which contains B identical components of the circuit $\mathcal{C}$. A SIMD-ZK proves that for input witnesses $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_B)$, $\mathcal{C}(\boldsymbol{w}_i) = 0$ for $i \in [B]$. By exploiting the fact that operations are identical across B components, SIMD-ZK schemes typically utilize vector commitments and batch proofs to achieve communication sublinear in $B \cdot |\mathcal{C}|$. In more detail, denote by $[\![\boldsymbol{w}]\!]$ a commitment to a vector $\boldsymbol{w}$. Define a witness matrix $\boldsymbol{W} = (\boldsymbol{w}_1 \| \ldots \| \boldsymbol{w}_B)$. Instead of viewing the ith column as the witness to the ith evaluation of $\mathcal{C}$, a prover commits to each row vector and lets the verifier obtain $([\![\boldsymbol{w}^1]\!], \ldots, [\![\boldsymbol{w}^{|\mathcal{C}|}]\!])$. In this way, for any gate $(\alpha, \beta, \gamma, \diamond)$ in $\mathcal{C}$ and $\diamond \in \{\mathsf{Add}, \mathsf{Mult}\}$, the prover only needs to prove that $\boldsymbol{w}^\gamma = \boldsymbol{w}^\alpha \diamond \boldsymbol{w}^\beta$. ZKP schemes achieve $\mathcal{O}(|\mathcal{C}|)$ proof size if both the vector commitment and batch proof of additions and multiplications incur constant size.

Most of priors work on different proof systems indeed take this approach by first implementing batch commitment and proof of multiplication gates, which are followed by a wiring consistency check [GWC19; CHM+20; CFQ19; CFF+21; AHI+17; WYY+22; YW22]. However, they take divergent paths to tackle the latter problem. A popular approach is to compile the circuit into an algebraic format via a constraint system, e.g. rank-1 constraint system (R1CS) [GGP+13]. Define $\boldsymbol{z} := (1, \boldsymbol{x}, \boldsymbol{w})$ in which $\boldsymbol{x}$ and $\boldsymbol{w}$ are the public and private inputs of the circuit. Denote $(\mathbf{L}, \mathbf{R}, \mathbf{O})$ as the matrices that represent the map from $\boldsymbol{z}$ to the vectors of the left, right and output wires of multiplication gates $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$. Then the relation $\boldsymbol{a} * \boldsymbol{b} - \boldsymbol{c} = \mathbf{0}$ can be expressed as $(\mathbf{L} \cdot \boldsymbol{z}) * (\mathbf{R} \cdot \boldsymbol{z}) - (\mathbf{O} \cdot \boldsymbol{z}) = \mathbf{0}$. In this way, the ZKP is reduced to proving matrix-vector products on committed values. On the other hand, some ZKPs like [WYY+22] and [YW22] proceed differently: they individually prove that $\boldsymbol{w}^\alpha[i] = \boldsymbol{w}^\beta[j]$ for any $i, j \in [B]$. Although this approach yields better scalability for the ZKP, it results in worse communication complexity, usually with a B^2 factor.

An interesting question is whether we can design a generic compiler that translates any commit-and-prove SIMD-ZK (CP-SIMD-ZK) into a general CP-ZK with sublinear communication. It would facilitate the design of communication-efficient ZKP because it allows the focus to be shifted to the design of SIMD-ZK primitives, which are generally easier than general-purpose ZKP.

From SIMD-ZK to scalable ZK. It is common for ZKPs to trade off scalability against succinctness. On the one hand, although zk-SNARKs generate proofs of constant size or size sublinear to $|\mathcal{C}|$, their memory overhead is at least $\mathcal{O}(|\mathcal{C}|)$. The constant factor is large when public-key operations are involved. This prevents them from being applied to large statements: prior benchmarks only focus on statements represented by less than 2^{25} constraints [CBB+23]. Efforts are made to distribute the zk-SNARK proof generation among a set of provers [WZC+18; OB22; SVV16; BG22], however, the overall computational and memory overhead is still prohibitive. They either need to disclose secret input to all provers, or only aim to delegate computation to more powerful workers but not to reduce the computational cost of them. Another line of work focuses on recursive SNARKs [KST22; KS22; BGH19] that allow a statement that can be divided into multiple steps to be proven step-by-step, but they require the statement to be structured, i.e., each step is represented by identical constraints. On the other hand, interactive ZKPs such as VOLE-ZK [BMR+21; WYK+21; YSW+21; DIO20] achieve high scalability by "streaming" the circuit evaluation. They evaluate the circuit gate-by-gate and only incur memory overhead linear in the current gates that are evaluated. Neither the witness nor the circuit structure for future gates are required to be known in advance. Hence these types of ZKPs scale to large circuits with billions of gates. However, their drawback is the $\mathcal{O}(|\mathcal{C}|)$ communication complexity and lack of public verifiability.

Naturally, it would be interesting to study how to achieve scalability and succinctness at the same time. Specifically, can we obtain efficient ZKPs with proof size sublinear to the circuit size, without the memory overhead being lower bounded by the circuit size?

4.1.2 Detailed Contributions

In this work, we start from SIMD-ZK schemes and aim to obtain efficient general ZK and scalable ZK. We first extend the SIMD-ZK functionality by adding a proof of linear map, which is easily realized by most SIMD-ZK schemes. Then we design two compilers. The first one converts a wide range of extended SIMD-ZK to general ZK, and the second one further converts it to scalable ZK for memory-constrained provers to prove large statements. For both constructions, we also demonstrate the generality of the compilers, i.e., our methods promote any SIMD-ZK to general and possibly scalable ZK so that attention can be paid only to the design of the efficient SIMD-ZK, instead of more complicated generic primitives.

Extended SIMD-ZK. We propose a functionality that extends the SIMD-ZK functionality $\mathcal{F}_{\mathsf{SIMDZK}}$ and denote it as $\mathcal{F}_{\mathsf{eSIMDZK}}$. In addition to the subroutines *commit*, *open* and *prove* that are commonly supported by SIMD-ZK schemes, it also contains a proof of linear map that checks the relation $\boldsymbol{x} = \mathbf{M}\boldsymbol{y}$ for committed vectors $(\boldsymbol{x}, \boldsymbol{y})$. The functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$ is the fundamental building block of our construction to compile SIMD-ZK to general ZK. Based on the extended SIMD-ZK, we design a SIMD compiler that allows a wide spectrum of SIMD-ZK to work for general circuits. To do so, it first converts the general circuit into a SIMD circuit by ignoring the circuit connectivity, and proves its satisfiability via a SIMD proof. This only utilizes the *commit*, *open* and *prove* thus can be handled by the underlying SIMD-ZK. Then the compiler represents the wiring as a linear mapping of committed wire values, and proves the wiring consistency by the proof of linear map from $\mathcal{F}_{\mathsf{eSIMDZK}}$.

ZKP for large statements. Except for VOLE-based ZKP, most practical ZKPs incur large RAM consumption, often linear to the circuit size. To relax the memory overhead, we propose a framework for memory bounded provers to prove the correctness of large statements. It also relies on $\mathcal{F}_{\mathsf{eSIMDZK}}$ and can easily achieve sublinear communication complexity for arbitrary large circuits by properly instantiating the underlying SIMD-ZK. Particularly, it utilizes the proving technique in our SIMD compiler to evaluate a circuit segment by segment and prove the connectivity of wires between these segments. Similar to the current scalable interactive ZK, it does not require the whole circuit structure or the witness to be known in advance, hence allowing streaming.

$\log_2 B$	Communication (MB)		Running time (s)	
	Setup	Online	Setup	Online
9	4.6	60.13	6.84	377.3
10	4.6	30.54	14.7	380.7
11	4.6	15.78	38.72	407.83
12	6.7	8.82	144.75	438.19
QS [YSW+21]	1087.23		185.43	

Table 4.1: Performance of AntMan++ with variable batch size. Benchmarked with 1 thread, 50 Mbps bandwidth and circuit size $C = 2^{27}$.

Scheme-Threads	Bandwidth (Mbps)			
	10	25	50	100
AM-1	461.71	449.75	446.55	444.17
AM-4	292.61	280.53	277.43	275.28
AM-8	263.55	249.89	248.24	246.07
QS-8 [YSW+21]	900.47	361.29	181.63	91.9

Table 4.2: Performance of AntMan++ with variable threads and bandwidth. Benchmarked with circuit size $C = 2^{27}$ and batch size $B = 2^{11}$. Numbers are in seconds.

Instantiation for VOLE-based proof systems. To demonstrate the generality of our compiler, we describe and analyze the detailed instantiation of our compiler with various CP-ZK that inherently work well for SIMD circuits, i.e., VOLE-based ZK [WYY+22]. We show how to adapt these work for general ZK and scalable ZK by merely satisfying the minimum requirement, that is, realizing the SIMD-ZK functionality. We emphasize that the transformation may affect the security guarantee of the underlying SIMD-ZK, and extra security analysis will be provided in that case.

Furthermore, we implement the SIMD compiler and evaluate the compilation of a VOLE-based ZK [WYY+22] that is previously designed for SIMD circuits. For a circuit of size $|\mathcal{C}| = 2^{27}$, it shows up to $83.6\times$ improvement on communication, compared to the general VOLE-ZK Quicksilver [YSW+21]. In terms of running time, it is 70% faster when bandwidth is 10Mbps and 30% faster when bandwidth increases to 25Mbps using the same set of parameters. Its running time can be further improved if sacrificing communication by reducing batch size as shown in Table 4.2,Table 4.1.

Related Work.

Previous work on complexity-preserving zero-knowledge proofs study efficient proof generation with constrained space or time budget [BHR+20; BHR+21; EFK+20; HR18; BC12; BCC+13]. Bootle et al. propose elastic SNARKs that can either achieve linear time and space complexity, or reduce the

RAM consumption to $\mathcal{O}(\log C)$ with $\mathcal{O}(C \log^2 C)$ computational complexity [BCH+22]. Assume an NP relation that can be verified in time T and space S by a RAM program, Bangalore et al. [BBH+22] propose a public-coin ZKP based on collision-resistant hash functions that allow the prover to run in time $\tilde{\mathcal{O}}(T)$ and space $\tilde{\mathcal{O}}(S)$, with proof size $\tilde{\mathcal{O}}(T/S)$. Their space-preserving ZKP is converted from Ligero [AHI+17].

Recent recursive zk-SNARK and incremental verifiable computation (IVC) propose succinct arguments for composed circuits, which can be evaluated step by step [KST22; KS22; STW23; BGH19; BCL+21]. These techniques increase the scalability of the prover, who separately generates proof for each step while simultaneously proves its consistency with all previous steps without going over the history data. They can potentially support streaming proofs in a way that the input and witness for future steps are not necessarily known until those steps are reached. However, many of them only support structured circuit which are divided into a sequence of components that share the same structure. More advanced IVCs cross this barrier, however, they reveal the output of each step and thus do not provide the zero-knowledge guarantee when they are treated as general ZK [KST22; KS22].

Notation and Functionalities.

For a vector $\mathbf{x} \in \mathbb{F}^B$ we define its i-th coordinate by x_i, and a vector $\mathbf{x}' := (f(0), \mathbf{x}) \in \mathbb{F}^{B+1}$ as the concatenation of a value $f(0) \in \mathbb{F}$ and the vector $\mathbf{x}$. Given distribution ensembles $\{X_n\}, \{Y_n\}$, we write $X_n \approx Y_n$ to denote that X_n is computationally indistinguishable to Y_n. $\mathsf{negl}(\lambda)$ is defined as a negligible function such that $\mathsf{negl}(\lambda) = o(\lambda^{-c})$ for any positive constant c. A circuit $\mathcal{C}$ over a field $\mathbb{F}$ consists of input, output, addition and multiplication gates, where input gates use circuit-input wires as their output wires and output gates use circuit-output wires as their input wires. $|\mathcal{C}| = C$ is the number of multiplication gates in the circuit $\mathcal{C}$. Define $(B, \mathcal{C})$-SIMD circuit as a circuit that contains B copies of $\mathcal{C}$.

We prove the security in the UC model using two ideal functionalities of interactive ZKP $\mathcal{F}_{\mathsf{ZK}}$ and Vector OLE $\mathcal{F}_{\mathsf{VOLE}}$ which are defined in Chapter 2.

Lemma 4.1.1 (Schwartz–Zippel)**.** *Let $P \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be a non-zero polynomial of total degree d over an field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$ and let $r_1, \dots, r_n$ be selected at random independently and uniformly from S. Then* $\Pr[P(r_1, r_2, \dots, r_n) = 0] \leq d/|S|$.

4.1.3 Technical Overview

From SIMD to General Circuit in ZK

Denote the prover as P and verifier as V. Define $(B, \mathcal{C})$-SIMD circuit as B identical repetitions of a circuit $\mathcal{C}$ with size $|\mathcal{C}| = C$. SIMD-ZK is designed for such circuits. First, we would like to focus on converting SIMD-ZK to general ZK that works for arbitrary circuits. The functionality of ZKP for SIMD circuits is shown in Figure 4.1. P first groups and commits to the vectors of witnesses. Then it uses the underlying ZKP to prove the relation of committed values by directly operating on commitments. Since elements in each vector are committed in a batch, the operations on the commitment apply to all of the committed elements.

The common framework to conduct the transformation from SIMD-ZK to general ZK is decomposing a circuit into a batch of SIMDs and using a wire consistency check on top of SIMD-ZK to check the consistency between SIMDs. As in AntMan [WYY+22], one can first arrange all gates in batches, commit to their input and output wire values, then utilize a SIMD-ZK to prove that all batches of gates are computed correctly. Then an extra protocol is invoked to prove the consistency of

Figure 4.1: Functionality of SIMD ZK $\mathcal{F}_{\mathsf{SIMDZK}}$

PUBLIC PARAMETER: Define B to be the batch size and τ_{max} to be the maximum time that a commitment can be used in the proof.

Commit: Upon receiving input $(\mathsf{Commit}, \boldsymbol{w} \in \mathbb{F}^B)$ from P and (Commit) from V, pick a tag $[\![\boldsymbol{w}]\!]$ and store $([\![\boldsymbol{w}]\!], \boldsymbol{w}, \mathsf{ctr}_{\boldsymbol{w}} = 0)$ in the memory. Return $[\![\boldsymbol{w}]\!]$ to both parties.

OPEN: Upon receiving $(\mathsf{Open}, [\![\boldsymbol{w}]\!])$, if a tuple $([\![\boldsymbol{w}]\!], \boldsymbol{w})$ was previously stored, output $([\![\boldsymbol{w}]\!], \boldsymbol{w})$ to V; otherwise abort.

PROVE: Upon receiving $(\mathsf{Prove}, \mathcal{C}, [\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_m]\!])$, where the circuit $\mathcal{C} : \{0,1\}^m \to \{0,1\}$, fetch $\boldsymbol{w}_i$ from the memory, for $i \in [m]$. If for any $\boldsymbol{w}_i$ that $[\![\boldsymbol{w}_i]\!]$ does not exist or its counter $\mathsf{ctr}_{\boldsymbol{w}_i} \geq \tau_{\mathsf{max}}$, abort. Check $\mathcal{C}(\boldsymbol{w}_1[i], \ldots, \boldsymbol{w}_m[i]) = 0$ for all $i \in [B]$. If any check fails, abort; otherwise, return Pass. For $i \in [m]$, set $\mathsf{ctr}_{\boldsymbol{w}_i} = \mathsf{ctr}_{\boldsymbol{w}_i} + 1$.

each individual wire value that is repeatedly packed in multiple commitments, e.g., for batched wire values $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{F}^B$ and wire indices $i, j \in [B]$, it aims to check whether they satisfy $\boldsymbol{w}_1[i] = \boldsymbol{w}_2[j]$. AntMan requires $\mathcal{O}(B^3)$ complexity for checking all combinations of $(i, j) \in [B] \times [B]$, which leads to a total communication complexity of $\mathcal{O}(B^3 + C/B)$. This translates to a $\mathcal{O}(C^{3/4})$ cost when setting $B = C^{1/4}$.

A better wire consistency check. We follow an idea similar to the above but manage to improve the complexity from $\mathcal{O}(C^{3/4})$ to $\mathcal{O}(C^{1/2})$. We ignore the wiring of the circuit and pack the multiplication gates in blocks of size B, which results in C/B batches. The SIMD proof is invoked to first commit to the input and output wires of the packed multiplication gates, then prove the SIMD circuit satisfiability. They totally incur communication complexity $\mathcal{O}(C/B)$. Then, we manage to perform the wire consistency check with cost $\mathcal{O}(B)$ rather than $\mathcal{O}(B^3)$.

Instead of considering the wire consistency among each pair of commitments that contain values from the same wire as done in AntMan, we consider how they are all consistent with a global vector $\boldsymbol{w}$ that contains all wire values in the circuit. Taking the left input wire of all multiplication gates as an example. Define a circuit $\mathcal{C}$ that has a total of Bm wire values and Bn multiplication gates. Assume global wire values $\boldsymbol{w} \in \mathbb{F}^{Bm}$ and the values of left input wires across all multiplication gates $\boldsymbol{l} \in \mathbb{F}^{Bn}$. For any $i \in [Bn]$, the left wire of the i-th multiplication gate must be associated a wire index $\alpha_i \in [Bm]$ such that $\boldsymbol{l}[i] = \boldsymbol{w}[\alpha_i]$. Alternatively, one can define a mapping matrix $\mathbf{L} \in \{0,1\}^{Bn \times Bm}$ such that the i-th row $\mathbf{L}_i$ is all-zero except at the entry $\mathbf{L}_i[\alpha_i]$. In this way, the wire consistency check boils down to check $\boldsymbol{l} = \mathbf{L}\boldsymbol{w}$, where $\mathbf{L}$ is public and parties have commitments $\{[\![\boldsymbol{l}_i]\!]\}_{i\in[n]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i\in[m]}$. In the context of SIMD-ZK protocols, values in $\boldsymbol{l}$ and $\boldsymbol{w}$ are batch-committed, meaning that operations on them are applied to every element in the vector. As a result, it is not straightforward to use SIMD-ZK to prove wire consistency which intuitively involves operations for separate elements.

We sketch our idea below. First, let V send a challenge vector $\hat{\boldsymbol{r}} \in \mathbb{F}^{Bn}$ and convert the check of $\boldsymbol{l} \stackrel{?}{=} \mathbf{L}\boldsymbol{w}$ to the check of $\hat{\boldsymbol{r}}^{\mathsf{T}}\boldsymbol{l} \stackrel{?}{=} \hat{\boldsymbol{r}}^{\mathsf{T}}\mathbf{L}\boldsymbol{w}$. This reduces the proof of a matrix-vector multiplication to a proof of two inner products, with an increase in soundness error depending on the distribution of $\hat{\boldsymbol{r}}$. To simplify the notation, we define a public vector $\boldsymbol{v}^{\mathsf{T}} = \hat{\boldsymbol{r}}^{\mathsf{T}}\mathbf{L}$, then rewrite the above relation as $\hat{\boldsymbol{r}}^{\mathsf{T}}\boldsymbol{l} \stackrel{?}{=} \boldsymbol{v}^{\mathsf{T}}\boldsymbol{w}$. If we define a circuit $\mathcal{C} : \mathbb{F}^{2n+2m+1} \to \mathbb{F}$ such that

$$\mathcal{C}(r_1, \ldots, r_n, l_1, \ldots, l_n, v_1, \ldots, v_m, w_1, \ldots, w_m, q) : \sum_{i\in[n]} r_i \cdot l_i - \sum_{j\in[m]} v_j \cdot w_j - q,$$

then P can prove the above statement by: 1) Divide each of the vectors in $(\hat{\boldsymbol{r}}, \boldsymbol{l}, \boldsymbol{v}, \boldsymbol{w})$ into length-B

segments. Compute and commit to $\boldsymbol{q} := \sum_{i\in[n]} \hat{r}_i * l_i - \sum_{j\in[m]} v_j * w_j \in \mathbb{F}^B$. Prove the consistency between $(\hat{\boldsymbol{r}}, \boldsymbol{l}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{q})$ by using a SIMD-ZK composed of B evaluations of the circuit $\mathcal{C}$. 2) prove that $\sum_i \boldsymbol{q}[i] = 0$. This is not obvious, as it involves the computation of the sum of values in one commitment. A naive way is for P to open the commitment to $\boldsymbol{q}$, but it compromises the zero-knowledge requirements because $\boldsymbol{q}$ is the linear combination of private circuit wire values. To tackle the problem, P instead commits to a uniform vector $\tilde{\boldsymbol{r}} \in \mathbb{F}^B$ under the constraint that $\sum_{i\in[B]} \tilde{\boldsymbol{r}}[i] = 0$. It should be done before V samples $\hat{\boldsymbol{r}}$ (else P can break soundness). After P commits to the mask vector, V sends the challenge $\hat{\boldsymbol{r}}$ and the new SIMD circuit is defined to be

$$\begin{aligned}&\mathcal{C}'(\hat{r}_1, \ldots, \hat{r}_n, l_1, \ldots, l_n, v_1, \ldots, v_m, w_1, \ldots, w_m, q, \tilde{r})\\ &= \sum_{i\in[n]} \hat{r}_i \cdot l_i - \sum_{j\in[m]} v_j \cdot w_j - q - \tilde{r}\end{aligned}$$

P computes and commits to $\boldsymbol{q} \in \mathbb{F}^B$ such that

$$\boldsymbol{q} = \sum_{i\in[n]} \hat{r} * l_i - \sum_{j\in[m]} v_j * w_j - \tilde{r}.$$

The parties can now use the SIMD-ZK to prove B number of instances of $\mathcal{C}'$ with committed inputs $[\![\hat{\boldsymbol{r}}_1]\!], \ldots, [\![\hat{\boldsymbol{r}}_n]\!]$, $[\![\boldsymbol{l}_1]\!], \ldots, [\![\boldsymbol{l}_n]\!]$, $[\![\boldsymbol{v}_1]\!], \ldots,$ $[\![\boldsymbol{v}_m]\!]$, $[\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_m]\!]$, $[\![\boldsymbol{q}]\!]$ and $[\![\tilde{\boldsymbol{r}}]\!]$. Finally, the proof of $\sum_i \boldsymbol{q}[i] = 0$ is specific to the underlying commitment schemes. The naive way is to let P fully open $\boldsymbol{q}$ to V who verifies its sum locally. This would generally require $\mathcal{O}(B)$ communication complexity.

Soundness comes from the randomness of the challenge vector $\boldsymbol{r}$ that is sampled after P commits to $\tilde{\boldsymbol{r}}$. Assume that $\mathbb{F}$ is an exponentially large field and a cheating prover commits to $(\boldsymbol{l}, \boldsymbol{w})$ such that $\boldsymbol{l} - \mathbf{L}\boldsymbol{w} \neq \mathbf{0}^{Bn}$. By Schwarz-Zippel, the probability that the erroneous values happen to be corrected by $\hat{\boldsymbol{r}}$ during the check of $\sum_i \boldsymbol{q}[i] \stackrel{?}{=} 0$ where $\boldsymbol{q} := \hat{\boldsymbol{r}}^\intercal \boldsymbol{l} - \hat{\boldsymbol{r}}^\intercal \mathbf{L}\boldsymbol{w}$ is $1/|\mathbb{F}|$, which is negligible.

Figure 4.2: Functionality of extended SIMD zero-knowledge $\mathcal{F}_{\mathsf{eSIMDZK}}$

Public parameter: batch size B.
$\mathcal{F}_{\mathsf{eSIMDZK}}$ supports all that $\mathcal{F}_{\mathsf{SIMDZK}}$ supports and the following instruction.

Linear map: Upon receiving input $(\mathsf{LinearMap}, [\![\boldsymbol{x}_1]\!], \ldots, [\![\boldsymbol{x}_n]\!], [\![\boldsymbol{y}_1]\!], \ldots, [\![\boldsymbol{y}_n]\!], \mathbf{M})$, check if tuple $([\![\boldsymbol{x}_i]\!], \boldsymbol{x}_i)$ and $([\![\boldsymbol{y}_i]\!], \boldsymbol{y}_i)$ exists for $i \in [n]$ and that $\boldsymbol{x} = \mathbf{M}\boldsymbol{y}$. If any check fails, abort; otherwise, return Pass.

Plugging in the protocol. For a general circuit with a total of $|\boldsymbol{w}| = Bm$ wire values and $C = Bn$ multiplication gates, the above approach leads to a zero-knowledge proof of linear map that can be instantiated by any SIMD-ZK. The actual communication complexity depends on the cost of proving the inner product argument by the underlying SIMD-ZK, plus the opening cost of the commitment scheme. Let $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o}) \in \mathbb{F}^{nB}$ be the batched wire values of left, right and output of multiplication gates in the circuit. The wire consistency can be proven by checking $(\boldsymbol{l} \stackrel{?}{=} \mathbf{L}\boldsymbol{w}, \boldsymbol{r} \stackrel{?}{=} \mathbf{R}\boldsymbol{w}, \boldsymbol{o} \stackrel{?}{=} \mathbf{O}\boldsymbol{w})$, where $(\mathbf{L}, \mathbf{R}, \mathbf{O}) \in \mathbb{F}^{nB\times mB}$ are public maps that describe the circuit connectivity. Furthermore, the SIMD-ZK protocol handles the rest of the multiplicative relation check $\boldsymbol{o} \stackrel{?}{=} \boldsymbol{l} * \boldsymbol{r}$. This scheme is captured in the extended SIMD-ZK functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$ shown in Figure 4.2. Compared to the common SIMD-ZK functionality shown in Figure 4.1, it supports the proof of linear map between committed vectors. Based on this extended SIMD-ZK, we propose a compiler that compiles any SIMD-ZK into general ZK. By plugging this compiler to AntMan [WYY+22], it improves its communication complexity from $\mathcal{O}(C^{3/4})$ to $\mathcal{O}(C^{1/2})$.

Memory constrained prover. The above construction can be viewed as a compiler that enables a SIMD-ZK to handle arbitrary circuits $\mathcal{C}$, where all wire values fit in a vector $\boldsymbol{w}$ of size $\mathcal{O}(C)$. Assume the linear mapping matrices use succinct representation, the proof requires memory overhead $\mathcal{O}(C)$, which upper bounds the largest circuit that the scheme can prove. We propose a second compiler that further extends the previous idea to the streaming setting, in which the memory overhead is proportional to the plaintext evaluation of the circuit. Furthermore, the whole circuit structure and the witnesses are not required to be known until they are reached. Instead, P proves the circuit segment-by-segment and only needs to evaluate the current and the previous one at a time: the circuit $\mathcal{C}$ is split into segments $\mathcal{C} = (\mathcal{C}_1, \ldots, \mathcal{C}_{n'})$. For any consecutive segments $\mathcal{C}_j$ and $\mathcal{C}_{j+1}$, let $(\boldsymbol{w}_j, \boldsymbol{l}_j, \boldsymbol{r}_j, \boldsymbol{o}_j)$ and $(\boldsymbol{w}_{j+1}, \boldsymbol{l}_{j+1}, \boldsymbol{r}_{j+1}, \boldsymbol{o}_{j+1})$ be the witness and the input and output wire values of multiplication gates for each segment. P first uses a commit-and-prove SIMD-ZK to prove the internal satisfiability of $\mathcal{C}_j$ including the linear and multiplicative relations of $(\boldsymbol{w}_j, \boldsymbol{l}_j, \boldsymbol{r}_j, \boldsymbol{o}_j)$. Then P proves that the output wires of $\mathcal{C}_j$ correctly link to some input wires of $\mathcal{C}_{j+1}$. Namely, it additionally invokes the check of linear map to prove $\mathbf{M}\boldsymbol{w}_j = \tilde{\boldsymbol{w}}_{j+1}$, in which $\mathbf{M}$ is a map that indicates the connectivity between $\mathcal{C}_j$ and $\mathcal{C}_{j+1}$ and $\tilde{\boldsymbol{w}}_{j+1}$ are the input wire values of $\mathcal{C}_{j+1}$. After this, P and V discard everything for segment $\mathcal{C}_j$ and carry on with the check of internal circuit satisfiability of $\mathcal{C}_{j+1}$. The above step incurs memory overhead $\mathcal{O}(|\boldsymbol{w}_j| + |\tilde{\boldsymbol{w}}_{j+1}|)$. Based on this framework, P is able to prove the satisfiability of a large circuit by separately evaluating a sequence of smaller circuits.

Compiling AntMan SIMD-ZK. The AntMan SIMD-ZK protocol consists of the following key components: 1) a constant-size additive-homomorphic polynomial commitment scheme, 2) a proof of multiplicative relation on committed polynomials, i.e. prove that $f_0(\cdot) = f_1(\cdot) \cdot f_2(\cdot)$. and 3) a proof of degree reduction, i.e. for two polynomials $(f(\cdot), \hat{f}(\cdot))$ with degrees $d_1 < d_2$, $f(i) = \hat{f}(i)$ for $i \in [d_1 + 1]$. We write $[\![f]\!]$ for a commitment to the polynomial $f(\cdot)$. The AntMan protocol realizes $\mathcal{F}_{\mathsf{SIMDZK}}$ as follows:

1. For each batch of B private inputs $\boldsymbol{w}_{\boldsymbol{\alpha}} \in \mathbb{F}^B$, P computes a degree-$(B-1)$ polynomial $f_{\boldsymbol{\alpha}}$ such that $f_{\boldsymbol{\alpha}}(i) = \boldsymbol{w}_{\boldsymbol{\alpha}}[i]$. P commits to $f_{\boldsymbol{\alpha}}$ so that P and V obtain $[\![f_{\boldsymbol{\alpha}}]\!]$.

2. The parties process the circuit in topological order. For any batch of k addition gates with commitments to input wires $([\![f_{\boldsymbol{\alpha}}]\!], [\![f_{\boldsymbol{\beta}}]\!])$, P and V locally computes the commitment to output wires by $[\![f_{\boldsymbol{\gamma}}]\!] = [\![f_{\boldsymbol{\alpha}}]\!] + [\![f_{\boldsymbol{\beta}}]\!]$. For multiplication gates with input commitments $[\![f_{\boldsymbol{\alpha}}]\!]$ and $[\![f_{\boldsymbol{\beta}}]\!]$, P computes $\boldsymbol{w}_{\boldsymbol{\gamma}} = \boldsymbol{w}_{\boldsymbol{\alpha}} * \boldsymbol{w}_{\boldsymbol{\alpha}}$ and a degree-$(B-1)$ polynomial $f_{\boldsymbol{\gamma}}$ such that $f_{\boldsymbol{\gamma}}(i) = \boldsymbol{w}_{\boldsymbol{\gamma}}[i], i \in [B]$. P also computes $\hat{f}_{\boldsymbol{\gamma}}(\cdot) = f_{\boldsymbol{\alpha}}(\cdot) \cdot f_{\boldsymbol{\beta}}(\cdot)$. P commits to them by generating $[\![f_{\boldsymbol{\gamma}}]\!]$ and $[\![\hat{f}_{\boldsymbol{\gamma}}]\!]$.

3. For each multiplication gates with input and output wires $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$, P proves that $([\![f_{\boldsymbol{\alpha}}]\!], [\![f_{\boldsymbol{\beta}}]\!], [\![\hat{f}_{\boldsymbol{\gamma}}]\!])$ is a multiplication triple and $\hat{f}_{\boldsymbol{\gamma}}(i) = f_{\boldsymbol{\gamma}}(i)$ for $i \in [B]$.

4. When a batch of k output wires $\boldsymbol{\alpha}$, P opens the commitment to $f_{\boldsymbol{\alpha}}$, from which V reconstructs $\boldsymbol{w}_{\boldsymbol{\alpha}}$.

The overhead of AntMan SIMD-ZK lies in the commitment of batch circuit intermediate wire values at Step 2, which takes $\mathcal{O}(C)$ for a $(B, \mathcal{C})$-SIMD circuit. The proof of multiplication and degree reduction only incurs $\mathcal{O}(B)$ with random linear combination.

When applying the SIMD compiler to the AntMan SIMD-ZK, it takes $\mathcal{O}(C/B)$ to prove all multiplicative relations for a general circuit of size C. Namely, it checks C multiplication triples $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o})$ via SIMD-ZK. Additionally, it invokes the proof of linear map to check the wire consistency between $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o})$ and $\boldsymbol{w}$, which contains intermediate wire values in the circuit. This procedure incurs $\mathcal{O}(B)$ communication overhead at the final commitment opening. Hence, it takes $\mathcal{O}(C/B + B) \geq$

Figure 4.3: The protocol for extended SIMD ZK from SIMD ZK Π_{eSIMDZK}

INPUTS: The prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold a public matrix $\mathbf{M} \in \mathbb{F}^{Bn \times Bk}$ for some integers n and k. Commitments $[\![\boldsymbol{x}]\!]$ and $[\![\boldsymbol{y}]\!]$ are public, where $\boldsymbol{x} \in \mathbb{F}^{Bn}$ and $\boldsymbol{y} \in \mathbb{F}^{Bk}$.
PROTOCOL:

1. $\mathcal{P}$ uniformly samples a vector $\tilde{\boldsymbol{r}} \in \mathbb{F}^B$ such that $\sum_{i=1}^{B} \tilde{\boldsymbol{r}}[i] = 0$. Then $\mathcal{F}_{\mathsf{SIMDZK}}$ is invoked to obtain its commitment $[\![\tilde{\boldsymbol{r}}]\!]$.
2. $\mathcal{V}$ uniformly samples a vector $\hat{\boldsymbol{r}} \in \mathbb{F}^{Bn}$ and sends it to $\mathcal{P}$. Everyone computes $\boldsymbol{v} = \hat{\boldsymbol{r}}^T \mathbf{M} \in \mathbb{F}^{Bk}$. Then for $i \in [k]$, $\mathcal{F}_{\mathsf{SIMDZK}}$ is invoked to construct $[\![\boldsymbol{v}_i]\!]$, where $\boldsymbol{v}_i$ is the i-th B-sized vector of $\boldsymbol{v}$. In the same way, everyone can have access to $\{[\![\hat{\boldsymbol{r}}_i]\!]\}_{i\in[n]}$.
3. $\mathcal{P}$ computes $\boldsymbol{q} \in \mathbb{F}^B$, such that $\boldsymbol{q}[i] = \sum_{j=1}^{n} \hat{\boldsymbol{r}}_j[i]\boldsymbol{x}_j[i] - \sum_{j=1}^{k} \boldsymbol{v}_j[i]\boldsymbol{y}_j[i] + \tilde{\boldsymbol{r}}[i]$. $\mathcal{P}$ invokes $\mathcal{F}_{\mathsf{SIMDZK}}$ to obtain $[\![\boldsymbol{q}]\!]$.
4. Define circuit
$$\begin{aligned}&\mathcal{C}_{\mathsf{Lin}}(a_1, \dots, a_n, b_1, \dots, b_n, c_1, \dots, c_k, d_1, \dots, d_k, e, f)\\ &:= \sum_{i=1}^{n} a_i \cdot b_i - \sum_{i=1}^{k} c_i \cdot d_i + e - f,\end{aligned}$$
then call $\mathcal{F}_{\mathsf{SIMDZK}}.\mathsf{Prove}(\mathcal{C}_{\mathsf{Lin}}, [\![\hat{\boldsymbol{r}}_1]\!], \dots, [\![\hat{\boldsymbol{r}}_n]\!], [\![\boldsymbol{x}_1]\!], \dots, [\![\boldsymbol{x}_n]\!],$
$[\![\boldsymbol{v}_1]\!], \dots, [\![\boldsymbol{v}_k]\!], [\![\boldsymbol{y}_1]\!], \dots, [\![\boldsymbol{y}_k]\!], [\![\tilde{\boldsymbol{r}}]\!], [\![\boldsymbol{q}]\!])$.
5. $\mathcal{V}$ sends $(\mathsf{Open}, [\![\boldsymbol{q}]\!])$ to $\mathcal{F}_{\mathsf{SIMDZK}}$, which returns $\boldsymbol{q}$ to V; V checks $\sum_{i=1}^{B} \boldsymbol{q}[i] = 0$ and aborts if the check fails.

$\mathcal{O}(C^{1/2})$ in total to prove the satisfiability of arbitrary circuits. This protocol is referred as AntMan++. We implemented the AntMan++ and evaluate its performance on proving general circuits of size up to $C = 2^{27}$. It is compared with the prior practical VOLE-based ZK QuickSilver [YSW+21], which requires $\mathcal{O}(C)$ communication overhead. More details are shown in Section 4.1.6.

4.1.4 Generic Compiler of ZK Proofs from SIMD Circuits to Arbitrary Circuits

In this section, we first present a construction for extended SIMD-ZK functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$ which supports the proof of linear map, in addition to the normal SIMD-ZK functionality $\mathcal{F}_{\mathsf{SIMDZK}}$. Based on the extended SIMD-ZK, we describe our compiler that enables a SIMD-ZK scheme to work for general circuits. At last, we present a framework that allows SIMD-ZK schemes to prove large statements with small memory footprints.

Extended SIMD-ZK

The protocol for extended SIMD-ZK is shown in Figure 4.3, which realizes the functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$. It is based on the $\mathcal{F}_{\mathsf{SIMDZK}}$ functionality to perform the committing and opening of batched wire values, as well as prove the element-wise multiplicative relations between these batches. It takes input a public matrix $\boldsymbol{M} \in \mathbb{F}^{Bn \times Bk}$ and two vectors $\boldsymbol{x} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n) \in \mathbb{F}^{Bn}$ and $\boldsymbol{y} = (\boldsymbol{y}_1, \dots, \boldsymbol{y}_k) \in \mathbb{F}^{Bk}$ from P, outputs 1-bit information to V indicating whether $\boldsymbol{x} = \boldsymbol{M}\boldsymbol{y}$. Essentially, it is a proof of linear map. The first step is to reduce the proof of linear map to a proof of inner products, which is achieved by a random linear combination: V uniformly samples $\hat{\boldsymbol{r}} \in \mathbb{F}^{Bn}$ and converts the check of

$\boldsymbol{x} \overset{?}{=} \boldsymbol{M}\boldsymbol{y}$ into $\hat{\boldsymbol{r}}^\intercal \boldsymbol{x} \overset{?}{=} \boldsymbol{v}^\intercal \boldsymbol{y}$, where $\boldsymbol{v}^\intercal = \hat{\boldsymbol{r}}^\intercal \boldsymbol{M}$. After dividing these vectors into length-B segments, P and V invoke the $\mathcal{F}_{\mathsf{SIMDZK}}$ functionality of batch size B. P inputs $\boldsymbol{q}$ and proves the correctness of $\boldsymbol{q} = \sum_{i=1}^{n} \hat{\boldsymbol{r}}_i * \boldsymbol{x}_i - \sum_{j=1}^{k} \boldsymbol{v}_i * \boldsymbol{y}_i \in \mathbb{F}^B$. Eventually it opens the commitment to $\boldsymbol{q}$ and let V check $\sum_{i=1}^{B} \boldsymbol{q}[i] \overset{?}{=} 0$. To ensure the privacy of P, it needs to make sure that only opened commitment to $\boldsymbol{q}$ does not reveal information of $\boldsymbol{x}$ and $\boldsymbol{y}$. It does so by the random mask $\tilde{\boldsymbol{r}}$. The impact of this mask on soundness is negligible since it is committed before $\hat{\boldsymbol{r}}$ is sampled.

In terms of the cost, the protocol Π_{eSIMDZK} takes input $k+n$ vector commitments. During the protocol execution, it additionally commits to $k+n+1$ size-B vectors. If element-wise product between a public vector and a committed vector is supported by the underlying $\mathcal{F}_{\mathsf{SIMDZK}}$, the number of commitments is reduced to 1 size-B vector commitment. Parties invoke the Prove procedure from $\mathcal{F}_{\mathsf{SIMDZK}}$ to prove a $(B, n+k)$-SIMD circuit. P also opens a size-B vector to V with cost at most $\mathcal{O}(B)$. The cost is reduced if the underlying SIMD-ZK protocol provides an easier way to prove $\sum_{i=1}^{B} \boldsymbol{q}[i] = 0$ for a committed vector $\boldsymbol{q}$ without opening the commitment.

Theorem 4.1.1. *Protocol Π_{eSIMDZK} (Figure 4.3) securely realizes the Functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$ (Figure 4.2) in the $\mathcal{F}_{\mathsf{SIMDZK}}$-hybrid model, with soundness error $|\mathbb{F}|^{-1}$.*

Proof. We first consider the case of a malicious prover and then the case of a malicious verifier. In each case, we construct a PPT simulator $\mathcal{S}$ given access to functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$, and running a PPT adversary $\mathcal{A}$ as a subroutine while emulating $\mathcal{F}_{\mathsf{SIMDZK}}$ for $\mathcal{A}$. We show that no PPT environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution.

Malicious prover. The simulator $\mathcal{S}$ simulates the view of adversary $\mathcal{A}$ for the protocol execution of Π_{eSIMDZK} as follows:

1. By emulating the (Commit) command of $\mathcal{F}_{\mathsf{SIMDZK}}$, $\mathcal{S}$ receives $\tilde{\boldsymbol{r}}$ from $\mathcal{A}$ and sends a handler $[\![\tilde{\boldsymbol{r}}]\!]$ to $\mathcal{A}$.

2. $\mathcal{S}$ uniformly samples $\hat{\boldsymbol{r}} \in \mathbb{F}^{Bn}$ and sends to $\mathcal{A}$. For $i \in [k]$, after receiving (Commit, $\boldsymbol{v}_i$) from $\mathcal{A}$, $\mathcal{S}$ sends a handler $[\![\boldsymbol{v}_i]\!]$ to $\mathcal{A}$. Similarly, $\mathcal{S}$ sends $[\![\hat{\boldsymbol{r}}_i]\!]$ to $\mathcal{A}$ for $i \in [n]$.

3. After receiving (Commit, $\boldsymbol{q}$) from $\mathcal{A}$, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{SIMDZK}}$ by sending $\mathcal{A}$ another handler $[\![\boldsymbol{q}]\!]$.

4. $\mathcal{S}$ receives (Prove, $\mathcal{C}, \tau_1, \ldots, \tau_{2n+2k+2}$) from $\mathcal{A}$, and then checks whether τ_i for all $i \in [2n+2k+2]$ match their corresponding tags. For $i \in [B]$, $\mathcal{S}$ checks whether $\sum_{j=1}^{n} \hat{\boldsymbol{r}}_j[i]\boldsymbol{x}_j[i] - \sum_{j=1}^{k} \boldsymbol{v}_j[i]\boldsymbol{y}_j[i] + \tilde{\boldsymbol{r}}[i] - \boldsymbol{q}[i]$ equals to 0 or not. If any check fails, $\mathcal{S}$ aborts; otherwise sends Pass to $\mathcal{A}$.

5. $\mathcal{S}$ emulates the (Open) command of $\mathcal{F}_{\mathsf{SIMDZK}}$ and receives a handler τ from $\mathcal{A}$. If τ does not match $[\![\boldsymbol{q}]\!]$ or the vector $\boldsymbol{q}$ previously sent by $\mathcal{A}$ does not satisfy $\sum_{i=1}^{B} \boldsymbol{q}[i] = 0$, $\mathcal{S}$ aborts.

Define E to be the event that a cheating prover $\mathcal{A}$ successfully convinces V in the real world. This happens when $\boldsymbol{r}$ accidentally corrects the wrong input of $\mathcal{A}$. Define $\boldsymbol{z} = \mathbf{M}\boldsymbol{y}$ and

$$f(x_1, \ldots, x_{Bn}) = \sum_{i=1}^{Bn} x_i(\boldsymbol{x}[i] - \boldsymbol{z}[i]) + \sum_{i=1}^{B} \tilde{\boldsymbol{r}}[i].$$

With fixed $\boldsymbol{x}, \boldsymbol{z}, \tilde{\boldsymbol{r}}$ and uniformly sampled $\hat{\boldsymbol{r}}$, we have

$$\Pr[E | \boldsymbol{x} \neq \mathbf{M}\boldsymbol{y}] = \Pr[f(\hat{\boldsymbol{r}}) = 0 | \boldsymbol{x} \neq \mathbf{M}\boldsymbol{y}] = |\mathbb{F}|^{-1}.$$

since $f(x_1, \ldots, x_{Bn})$ is a Bn-variate degree-1 polynomial. Hence we conclude that $\mathcal{A}$ cannot distinguish between the real and ideal world except with probability $|\mathbb{F}|^{-1}$.

Malicious verifier. Similarly in this case, $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. To emulate the (Commit) command, $\mathcal{S}$ sends a handler $[\![\tilde{\boldsymbol{r}}]\!]$ to $\mathcal{A}$.
2. $\mathcal{S}$ recieves $\hat{\boldsymbol{r}}$ and $(\mathsf{Commit}, \boldsymbol{v}_i)$ from $\mathcal{A}$ for $i \in [k]$. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{SIMDZK}}$ by sending $\mathcal{A}$ a handler $[\![\boldsymbol{v}_i]\!]$ for $i \in [k]$. In the same way, $\mathcal{S}$ sends $\mathcal{A}$ handlers $\{[\![\hat{\boldsymbol{r}}_i]\!]\}_{i\in[n]}$.
3. Then, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{SIMDZK}}$ and sends a handler $[\![\boldsymbol{q}]\!]$ to $\mathcal{A}$.
4. $\mathcal{S}$ receives $(\mathsf{Prove}, \mathcal{C}, \tau_1, \ldots, \tau_{2n+2k+2})$ from $\mathcal{A}$ and checks whether $\{\tau_i\}_{i\in[2n+2k+2]}$ match their corresponding tags. Then $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{eSIMDZK}}$. If check fails or $\mathcal{F}_{\mathsf{eSIMDZK}}$ aborts, $\mathcal{S}$ aborts; otherwise sends Pass to $\mathcal{A}$.
5. By emulating the (Open) command of $\mathcal{F}_{\mathsf{SIMDZK}}$, $\mathcal{S}$ uniformly samples a vector $\boldsymbol{q} \in \mathbb{F}^B$ such that $\sum_{i=1}^{B} \boldsymbol{q}[i] = 0$ and sends $\boldsymbol{q}$ to $\mathcal{A}$.

The only difference between reality and the ideal world is the method of calculating vector $\boldsymbol{q}$. Following the constraint $\sum_{i=1}^{B} \boldsymbol{q}[i] = 0$, $\mathcal{S}$ uniformly samples vector $\boldsymbol{q}$. While in reality, each entry of $\boldsymbol{q}$ is masked by vector $\tilde{\boldsymbol{r}}$ chosen by $\mathcal{P}$. As a result, in both worlds, all entries except one of $\boldsymbol{q}$ are information-theoretic secure, so no one can distinguish one from another.

Overall, any PPT environment $\mathcal{Z}$ cannot distinguish between the real-world execution and ideal-world execution, which completes the proof.

□

Compiling Extended SIMD-ZK

The general approach to compiling a SIMD protocol into a generic protocol is to supplement it with additional proof of wiring consistency. Namely, denote $\boldsymbol{w}$ as a vector that includes all the wire values in a circuit, then any input wire of a multiplication gate can be represented as the linear combination of a series of values in $\boldsymbol{w}$, who are the wire values that connect from the circuit inputs or the output of other gates. This relation can be generally represented as a linear map $\boldsymbol{M}$ between a vector of wire values $\boldsymbol{x}$, and $\boldsymbol{w}$, which should satisfy $\boldsymbol{x} = \boldsymbol{M}\boldsymbol{w}$. As shown in Figure 4.4, along with the vector $\boldsymbol{w}$, P also commits to $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o})$ which are the batches of input and output wire values of multiplication gates. Showing that $\boldsymbol{o} = \boldsymbol{l} * \boldsymbol{r}$ is enough to prove that all multiplication gates are computed correctly. Additionally, P also proves the correctness of $(\boldsymbol{l} = \boldsymbol{L}\boldsymbol{w}, \boldsymbol{r} = \boldsymbol{R}\boldsymbol{w}, \boldsymbol{o} = \boldsymbol{O}\boldsymbol{w})$, in which $(\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{O})$ are the linear maps that defines the routing of wires that connect to the input and output wires of multiplication gates. Additionally, the proof of $\boldsymbol{0} = \boldsymbol{A}\boldsymbol{w}$ shows the correct computation of all addition gates.

To handle a general circuit $\mathcal{C}$, our compiler fully depends on the extended SIMD-ZK functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$. Regarding the cost analysis, P commits to a total of $k + 3n_2$ size-B vectors to V. They invoke the proof of linear map for 4 times to prove the wiring consistency, and the proof of element-wise multiplication to prove the correctness of n_2 batches of multiplication gates. An optimization to reduce the cost for the proof of linear map is to combine the 4 of them into 1. Namely, define $\boldsymbol{w}'$ to be the wire values excluding the input and output wires of multiplication gates. Construct the witness vector $\boldsymbol{w} = (\boldsymbol{w}' \| \boldsymbol{l} \| \boldsymbol{r} \| \boldsymbol{o})$ and prove the wiring consistency by proving $\boldsymbol{0} = \mathbf{A}'\boldsymbol{w}$, in which $\mathbf{A}' \in \mathbb{F}^{K\times K}$ is a map that describes the circuit wire connectivity.

Figure 4.4: Generic ZK in the $\mathcal{F}_{\mathsf{eSIMDZK}}$ hybrid Π_{compiler}

INPUTS: The prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold an arbitrary circuit $\mathcal{C}$ over a large field $\mathbb{F}$, where $\mathcal{C}$ contains $N_1 = Bn_1$ addition gates, $N_2 = Bn_2$ multiplication gates and $K = Bk$ wires for some n_1, n_2 and k.

PROTOCOL:

1. Set $c = 1$. For each gate in the form $(i, \alpha, \beta, \gamma, T)$
   - If $T = ADD$, set $\mathbf{A}_i := \mathbf{I}_\alpha + \mathbf{I}_\beta - \mathbf{I}_\gamma$; P sets $\boldsymbol{w}[\gamma] := \boldsymbol{w}[\alpha] + \boldsymbol{w}[\beta]$
   - If $T = MULT$, set $(\mathbf{L}_c, \mathbf{R}_c, \mathbf{O}_c) := (\mathbf{I}_\alpha, \mathbf{I}_\beta, \mathbf{I}_\gamma)$; P sets $(\boldsymbol{l}[c], \boldsymbol{r}[c], \boldsymbol{o}[c]) =: (\boldsymbol{w}[\alpha], \boldsymbol{w}[\beta], \boldsymbol{w}[\alpha] \cdot \boldsymbol{w}[\beta])$. Increase c by 1.

   After the circuit is processed, matrix $\mathbf{L}, \mathbf{R}, \mathbf{O} \in \mathbb{F}^{N_2 \times K}$, and $\mathbf{A} \in \mathbb{F}^{N_1 \times K}$ are public; P has $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o}, \boldsymbol{w}) \in \mathbb{F}^{N_2} \times \mathbb{F}^{N_2} \times \mathbb{F}^{N_2} \times \mathbb{F}^{K}$.
2. P splits wire values $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o}, \boldsymbol{w})$ into chunks of size B, i.e., $\{\boldsymbol{l}_i, \boldsymbol{r}_i, \boldsymbol{o}_i\}_{i \in [n_2]}$ and $\{\boldsymbol{w}_i\}_{i \in [k]}$, such that each element is in $\mathbb{F}^B$. $\mathcal{F}_{\mathsf{eSIMDZK}}$ is invoked to obtain commitments $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}_{i \in [n_2]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i \in [k]}$.
3. Then, $(\mathsf{LinearMap}, \{[\![\boldsymbol{l}_i]\!]\}_{i \in [n_2]}, \{[\![\boldsymbol{w}_i]\!]\}_{i \in [k]}, \mathbf{L})$ is sent to $\mathcal{F}_{\mathsf{eSIMDZK}}$ to check that $\boldsymbol{l} = \mathbf{L}\boldsymbol{w}$; similarly check that $\boldsymbol{r} = \mathbf{R}\boldsymbol{w}$, $\boldsymbol{o} = \mathbf{O}\boldsymbol{w}$, and that $\mathbf{0} = \mathbf{A}\boldsymbol{w}$.
4. Let circuit $\mathcal{C}_{\mathsf{Mult}} : \mathbb{F}^3 \to \mathbb{F}$ such that $\mathcal{C}_{\mathsf{Mult}}(x, y, z) := xy - z$. For $i \in [n_2]$, send $(\mathsf{Prove}, \mathcal{C}_{\mathsf{Mult}}, [\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!])$ to $\mathcal{F}_{\mathsf{eSIMDZK}}$.

Theorem 4.1.2. *The Protocol Π_{compiler} (Figure 4.4) securely realizes the Functionality $\mathcal{F}_{\mathsf{ZK}}$ in the $\mathcal{F}_{\mathsf{eSIMDZK}}$-hybrid model, with 0 soundness error.*

Proof. Similarly, we construct a PPT simulator in two cases and argue that no PPT environment $\mathcal{Z}$ can distinguish reality and the ideal world.

Malicious prover. The simulator $\mathcal{S}$ simulates the view of adversary $\mathcal{A}$ for the protocol execution of Π_{compiler} as follows:

1. Following the protocol specification, $\mathcal{S}$ obtain matrix $\mathbf{L}$, $\mathbf{R}$, $\mathbf{O}$ and $\mathbf{A}$ from circuit $\mathcal{C}$.
2. By emulating the (Commit) command of $\mathcal{F}_{\mathsf{eSIMDZK}}$, $\mathcal{S}$ receives $\{\boldsymbol{l}_i, \boldsymbol{r}_i, \boldsymbol{o}_i\}_{i \in [n_2]}$ and $\{\boldsymbol{w}_i\}_{i \in [k]}$ from $\mathcal{A}$ and sends $\mathcal{A}$ handlers $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!],$

   $[\![\boldsymbol{o}_i]\!]\}_{i \in [n_2]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i \in [k]}$.
3. After receiving $(\mathsf{LinearMap}, \{\tau_i\}_{i \in [n_2 + k]}, \mathbf{L})$ from $\mathcal{A}$, $\mathcal{S}$ checks whether $\{\tau_i\}_{i \in [n_2]}$ match $\{[\![\boldsymbol{l}_i]\!]\}_{i \in [n_2]}$ and $\{\tau_i\}_{i \in [n_2+1, n_2+k]}$ matches $\{[\![\boldsymbol{w}_i]\!]\}_{i \in [k]}$. Then, $\mathcal{S}$ checks whether $\boldsymbol{l} = \mathbf{L}\boldsymbol{w}$. If any check fails, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ sends Pass to $\mathcal{A}$. Similarly, $\mathcal{S}$ handles other three $(\mathsf{LinearMap})$ commands from $\mathcal{A}$.
4. For $i \in [n_2]$, $\mathcal{S}$ receives $(\mathsf{Prove}, \mathcal{C}, \tau_1, \tau_2, \tau_3)$ from $\mathcal{A}$ and checks whether $\{\tau_1, \tau_2, \tau_3\}$ match the tags $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}$. In each round, $\mathcal{S}$ also checks that $\boldsymbol{l}_i[j] \cdot \boldsymbol{r}_i[j] = \boldsymbol{o}_i[j]$ for $j \in [B]$. If any check fails, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ sends Pass to $\mathcal{A}$.

It is trivial that $\mathcal{S}$ is perfect, since whenever an ideal functionality is called in the protocol, $\mathcal{S}$ acts exactly the same as the definition of the functionality. On the other hand, if the witness indeed satisfies linear as well as the multiplication constraints, we can conclude that it satisfies circuit $\mathcal{C}$. Given the perfectness of the ideal functionality, we can conclude that the soundness error is 0.

Malicious verifier. The simulator $\mathcal{S}$ simulates the view of adversary $\mathcal{A}$ for the protocol execution of Π_{compiler} as follows:

1. $\mathcal{S}$ follows the protocol specification and obtain matrix $\mathbf{L}, \mathbf{R}, \mathbf{O}$ and $\mathbf{A}$ from circuit $\mathcal{C}$.
2. By emulating the (Commit) command of functionality $\mathcal{F}_{\mathsf{eSIMDZK}}$, $\mathcal{S}$ sends $\mathcal{A}$ handlers $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}_{i\in[n_2]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i\in[k]}$.
3. After receiving $(\mathsf{LinearMap}, \{\tau_i\}_{i\in[n_2+k]}, \mathbf{L})$ from $\mathcal{A}$, $\mathcal{S}$ checks whether $\{\tau_i\}_{i\in[n_2]}$ match $\{[\![\boldsymbol{l}_i]\!]\}_{i\in[n_2]}$ and $\{\tau_i\}_{i\in[n_2+1,n_2+k]}$ matches $\{[\![\boldsymbol{w}_i]\!]\}_{i\in[k]}$. Then, $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{ZK}}$. If check fails or $\mathcal{F}_{\mathsf{ZK}}$ aborts, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ sends Pass to $\mathcal{A}$. Similarly, $\mathcal{S}$ handles other three $(\mathsf{LinearMap})$ commands from $\mathcal{A}$.
4. For $i \in [n_2]$, $\mathcal{S}$ receives $(\mathsf{Prove}, \mathcal{C}, \tau_1, \tau_2, \tau_3)$ from $\mathcal{A}$ and checks whether $\{\tau_1, \tau_2, \tau_3\}$ match the tags $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}$. In each round, $\mathcal{S}$ also queries $\mathcal{F}_{\mathsf{ZK}}$. If any check fails or $\mathcal{F}_{\mathsf{ZK}}$ aborts, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ sends Pass to $\mathcal{A}$.

Similarly, since $\mathcal{S}$ acts according to the definition of the ideal functionality and there is no commitment opening during the protocol, the simulation is perfect.

As a result, no PPT environment $\mathcal{Z}$ can distinguish between the real-world scenario and the ideal-world execution, which completes the proof. □

4.1.5 Generic ZK for Limited-Memory

Besides a basic-version compiler, we also present another compiler that can deal with a situation where the prover's memory is limited. Although a similar question has already been proposed before [BCC+13; GGP+13; Pat04; KST22], our construction does not rely on any complicated assumption other than the realization of $\mathcal{F}_{\mathsf{eSIMDZK}}$ with the parameter $\tau_{\mathsf{max}} > 1$. The protocol is shown in Figure 4.5. We take the advantage of the commit-and-prove paradigm: instead of proving the whole circuit at one time, circuit can be "partially" proved. The value of wires that connect between different parts of the circuit can be reserved as commitments and used for the proof of connectivity. Specifically, prover will clarify a space threshold parameter S before the proof, and the original circuit $\mathcal{C}$ will be divided into $\lceil|\mathcal{C}|/S\rceil$ parts (denoted as $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{\lceil|\mathcal{C}|/S\rceil}$), where each part contains at most S gates. In each round, S gates of $\mathcal{C}_i$ will be read and processed in the memory, and $\mathcal{P}$ generates the proof for $\mathcal{C}_i$. At the end of each round, $\mathcal{P}$ commits to a vector which contains all the wire values that are still active in $\mathcal{C}_{i+1}$, and discards those that won't be used in the remaining circuit.

To support this pruning operation, we add a DEL gate to the encoding of the circuit. $\mathcal{P}$ reads the circuit from a stream of $(\alpha, \beta, \gamma, T)$, where $T \in \{ADD, MULT, DEL\}$. If $T \in \{ADD, MULT\}$, $\mathcal{P}$ processes gates α, β, γ similarly as the previous compiler. If $T = DEL$, $\mathcal{P}$ adds gate α to the set $\mathcal{D}$, which contains all the wire values that no longer appear in the next segment of the circuit. After the proof of consistency inside $\mathcal{C}_i$, P forms a new commitment to wire values that are not in the set $\mathcal{D}$. By applying $\mathcal{F}_{\mathsf{eSIMDZK}}.\mathsf{LinearMap}$, $\mathcal{P}$ proves that the committed wire values belongs to the output wires of $\mathcal{C}_i$, which are also the input of $\mathcal{C}_{i+1}$. $\mathcal{P}$ and $\mathcal{V}$ repeat this procedure for the proof of each segment.

Now we claim that if the plaintext evaluation of circuit $\mathcal{C}$ requires memory space M, then in our protocol, the prover's space complexity is $\mathcal{O}(M)$. Denote $\boldsymbol{o}_i$ as the output of subcircuit $\mathcal{C}_i$, and circuit input $\boldsymbol{x}$ is denoted as $\boldsymbol{o}_0$. In each round, we call $\mathcal{F}_{\mathsf{eSIMDZK}}.\mathsf{Prove}$ to complete the proof for $\mathcal{C}_i$ and $\mathcal{F}_{\mathsf{eSIMDZK}}.\mathsf{LinearMap}$ to prove the transformation between $[\![\boldsymbol{o}_{i-1}]\!]$ and $[\![\boldsymbol{o}_i]\!]$. As each subcircuit contains at most S gates, proving $\mathcal{C}_i$ requires $\mathcal{O}(S)$ space. And also, using $\mathcal{F}_{\mathsf{eSIMDZK}}.\mathsf{LinearMap}$ to prove the consistency between $[\![\boldsymbol{o}_{i-1}]\!]$ and $[\![\boldsymbol{o}_i]\!]$ requires $\mathcal{O}(|\boldsymbol{o}_{i-1}| + |\boldsymbol{o}_i|)$ space, so the space

Figure 4.5: Generic ZK in limited-memory scenario $\Pi_{\mathsf{small-space}}$

INPUTS: The prover $\mathcal{P}$ and verifier $\mathcal{V}$ hold an arbitrary circuit $\mathcal{C}$ over a large field $\mathbb{F}$, and a space threshold parameter $S = sB$ for some integer s. $\mathcal{P}$ holds the secret input $\boldsymbol{x}$ such that $\mathcal{C}(\boldsymbol{x}) = 0$.

PROTOCOL:

1. Let $h() : \mathbb{Z} \to \mathbb{Z}$ be a function map wire indices to physical indices and $\boldsymbol{w}$ be a dynamic list storing wire value to be dealt with in the current round. Initially, set $h(i) = i$ and $\boldsymbol{w}[i] = \boldsymbol{x}[i]$ for all $i \in [|\boldsymbol{x}|]$. Define function $\mathsf{Im}() : f \to \mathbb{Z}$, returning the maximum index that f has the definition.
2. Let $\mathcal{D} = \varnothing$ and $W = \mathsf{Im}(h) + S$. Initialize **L, R, O, A** to be empty matrices. Read the next S gates to the memory (or until the last gate). For each in the form $(\alpha, \beta, \gamma, T)$:
   - If $T = ADD$, set $h(\gamma) := \mathsf{Im}(h) + 1$, compute $\boldsymbol{r} := \mathbf{I}_{h(\alpha)} + \mathbf{I}_{h(\beta)} - \mathbf{I}_{h(\gamma)} \in \mathbb{F}^W$ and append $\boldsymbol{r}$ to $\mathbf{A}$. P sets $\boldsymbol{w}[h(\gamma)] := \boldsymbol{w}[h(\alpha)] + \boldsymbol{w}[h(\beta)]$
   - If $T = MULT$, set $h(\gamma) := \mathsf{Im}(h) + 1$, append rows in $\mathbb{F}^W$ $\mathbf{I}_{h(\alpha)}, \mathbf{I}_{h(\beta)}, \mathbf{I}_{h(\gamma)}$ to matrices $\mathbf{L}, \mathbf{R}, \mathbf{O}$ respectively. P sets $\boldsymbol{w}[h(\gamma)] := \boldsymbol{w}[h(\alpha)] \cdot \boldsymbol{w}[h(\beta)]$ and append values $\boldsymbol{w}[h(\alpha)], \boldsymbol{w}[h(\beta)], \boldsymbol{w}[h(\gamma)]$ to vectors $\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o}$ respectively.
   - If $T = DEL$, add α to $\mathcal{D}$.

   Suppose that there are $S_1 = s_1 B$ addition gates and $S_2 = s_2 B$ multiplication gates ($S = S_1 + S_2$), and after processing S gates, $|\boldsymbol{w}| = kB$. $\mathbf{A} \in \mathbb{F}^{S_1 \times W}$ and $\mathbf{L}, \mathbf{R}, \mathbf{O} \in \mathbb{F}^{S_2 \times W}$ are public.
3. P splits wire values ($\boldsymbol{l}$, $\boldsymbol{r}$, $\boldsymbol{o}$, $\boldsymbol{w}$) into chunks of size B, i.e., $\{\boldsymbol{l}_i, \boldsymbol{r}_i, \boldsymbol{o}_i\}_{i \in [s_2]}$ and $\{\boldsymbol{w}_i\}_{i \in [k]}$, such that each element is in $\mathbb{F}^B$. $\mathcal{F}_{\mathsf{eSIMDZK}}$ is invoked to obtain commitments $\{[\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!]\}_{i \in [s_2]}$ and $\{[\![\boldsymbol{w}_i]\!]\}_{i \in [k]}$.
4. Then, $(\mathsf{LinearMap}, \{[\![\boldsymbol{l}_i]\!]\}_{i \in [n_2]}, \{[\![\boldsymbol{w}_i]\!]\}_{i \in [k]}, \mathbf{L})$ is sent to $\mathcal{F}_{\mathsf{eSIMDZK}}$ to check that $\boldsymbol{l} = \mathbf{L}\boldsymbol{w}$; similarly check that $\boldsymbol{r} = \mathbf{R}\boldsymbol{w}$, $\boldsymbol{o} = \mathbf{O}\boldsymbol{w}$, and that $\mathbf{0} = \mathbf{A}\boldsymbol{w}$.
5. Let circuit $\mathcal{C}_{\mathsf{Mult}} : \mathbb{F}^3 \to \mathbb{F}$ such that $\mathcal{C}_{\mathsf{Mult}}(x, y, z) := xy - z$. For $i \in [s_2]$, $(\mathsf{Prove}, \mathcal{C}_{\mathsf{Mult}}, [\![\boldsymbol{l}_i]\!], [\![\boldsymbol{r}_i]\!], [\![\boldsymbol{o}_i]\!])$ is sent to $\mathcal{F}_{\mathsf{eSIMDZK}}$.
6. Let $\mathcal{R} = \mathsf{Domain}(h) \setminus \mathcal{D}$. Suppose that $|\mathcal{R}| = k'$. For the i-th element in $\mathcal{R}$, let $h'(\mathcal{R}[i]) = i$, and set the i^{th} row of $\mathbf{H}$ as $\mathbf{I}_{h(\mathcal{R}[i])}$.
7. $\mathcal{P}$ computes $\boldsymbol{w}'$ such that for each $\boldsymbol{w}'[h'(i)] = \boldsymbol{w}[h(i)]$. Append 0 to $\boldsymbol{w}'$ and $\mathbf{0}$ to $\mathbf{H}$ until the size of $\boldsymbol{w}'$ becomes a multiple of B. Supppose that $|\boldsymbol{w}'| = k'B$, and then $\mathcal{P}$ calls Commit to obatin $\{[\![\boldsymbol{w}'_i]\!]\}_{i \in [k']}$. Update $(h, \boldsymbol{w}) := (h', \boldsymbol{w}')$.
8. Both parties call $(\mathsf{LinearMap}, \{\boldsymbol{w}'_i\}_{i \in [k']}, \{\boldsymbol{w}_i\}_{i \in [k]}, \mathbf{H})$ to check the consistency between $\boldsymbol{w}$ and $\boldsymbol{w}'$.
9. If more gates need to be processed, jump to step 2.

complexity of each round is $\mathcal{O}(S + |\boldsymbol{o}_{i-1}| + |\boldsymbol{o}_i|)$. As a result, the overall space complexity is $\mathcal{O}(S + \max\{|\boldsymbol{o}_{i-1}| + |\boldsymbol{o}_i|\}_{i \in [\lceil |\mathcal{C}|/S \rceil]})$. Since in the plaintext evaluation of $\mathcal{C}$, only active wire value needs to be read into the memory, memory upper bound $M \geq \max\{|\boldsymbol{o}_0|, |\boldsymbol{o}_1|, |\boldsymbol{o}_2|, \ldots, |\boldsymbol{o}_{\lceil |\mathcal{C}|/S \rceil}|\}$. By choosing $S < M$, we can conclude that the space complexity of the protocol is $\mathcal{O}(M)$.

4.1.6 Sublinear Designated-Verifier ZK

This section shows an instantiation of SIMD-ZK that benefits from our compiler. We leverage Antman [WYY+22] to achieve a sublinear ZKP based on vector OLE.

Figure 4.6: The protocol of SIMDZK from AntMan Π_{AntMan}

PUBLIC INPUT. The prover P and verifier V hold a general circuit $\mathcal{C}$ over a large field $\mathbb{F}$, where $\mathcal{C}$ contains $n = |\mathcal{C}|$ multiplication gates and m input gates. Let $\alpha_1, \ldots, \alpha_B \in \mathbb{F}$ be B distinct elements that are fixed for the whole protocol execution. Both parties invoke Initialize() in IT-PAC to obtain τ_1.

PRIVATE INPUT. P holds m witnesses $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m \in \mathbb{F}^B$ such that $\mathcal{C}(\boldsymbol{w}_1[i], \ldots, \boldsymbol{w}_m[i]) = 0$ for all $i \in [B]$.

COMMIT: On input $\boldsymbol{w} \in \mathbb{F}^B$, $\mathcal{P}$ computes polynomial $f(\cdot) = \sum_{i=0}^{B-1} f_i \cdot X^i$ such that for $i \in [B]$, $f(\alpha_i) = w_i$. Both parties invoke $(\langle b \rangle, \tau_2) \leftarrow \mathsf{PreGen}(f)$. $\mathcal{P}$ obtains $\langle b \rangle$ and $\mathcal{V}$ obtains τ_2. If Λ has been revealed, invoke $(\mathsf{M}, \mathsf{K}) \leftarrow \mathsf{Gen}(\tau_1, \tau_2)$. $\mathcal{P}$ holds M and $\mathcal{V}$ holds K.

OPEN: On input $(\llbracket f(\cdot) \rrbracket, f(\cdot), \Lambda)$, both parties compute that $\llbracket \mu \rrbracket := \llbracket f(\Lambda) \rrbracket - f(\Lambda)$. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a random oracle. $\mathcal{P}$ sends $\mathsf{H}(\mathsf{M}_\mu)$ to $\mathcal{V}$ who checks whether $\mathsf{H}(\mathsf{M}_\mu) = \mathsf{H}(\mathsf{K}_\mu)$.

AntMan: VOLE-based Designated-Verifier ZK. AntMan [WYY+22] is a sublinear VOLE-based ZK proof for SIMD circuits, which only requires communicating $\mathcal{O}(B + |\mathcal{C}|)$ field elements to prove a $(B, \mathcal{C})$-SIMD circuit. It also presents a construction for proving a single execution of an arbitrary circuit, by breaking down the circuits into individual gates and batching them as SIMD circuits. The proving of SIMD circuits requires sending $\mathcal{O}(|\mathcal{C}|/B + B)$ field elements, and the cost to check the wire-value consistency is $\mathcal{O}(B^3)$, which leads to $\mathcal{O}(|\mathcal{C}|^{3/4})$ communication complexity in optimal. It is the only sublinear-communication VOLE-ZK protocol for proving an arbitrary circuit. In AntMan [WYY+22], the information-theoretic polynomial authentication code $\Pi^k_{\mathsf{IT\text{-}PAC}}$ servers as a polynomial commitment scheme. For arbitrary degree-k polynomial $f(\cdot)$ known by P, an IT-PAC $\llbracket f(\cdot) \rrbracket$ consists of a MAC $M \in \mathbb{F}$ known by P and a tuple of keys $(K, \Delta, \Lambda) \in \mathbb{F}^3$ known by V, such that $M = K + f(\Lambda) \cdot \Delta$.

In the following, we first detail the commitment scheme used in the AntMan protocol, then discuss how to enable AntMan to prove arbitrary circuits.

Information-theoretic polynomial authentication code $\Pi_{\mathsf{IT-PAC}}$. As shown in Figure 4.7, the protocol is designed in the $(\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{Com}})$-hybrid model. It adopts additively homomorphic encryption (AHE) scheme to obliviously evaluate a polynomial, where the polynomial is known by P and the secret point Λ is known by V. Then VOLE correlations further transform such oblivious polynomial evaluation (OPE) into IT-PACs. A critical issue is to guarantee that the HE ciphertext which encodes the evaluation point Λ is correct. Instead of using the zero-knowledge proof of knowledge for the proof of validity (as done in several MPC protocols [KPR18; DPS+12]), AntMan utilizes a simple commit-and-open approach. Specifically, V first commits to the randomness that are used to generate the HE ciphertexts $\langle \Lambda^1 \rangle, \ldots, \langle \Lambda^k \rangle$. After receiving HE ciphertexts from V, P performs the homomorphic evaluation and commits to all of HE ciphertexts $\langle b \rangle$ that it should send to V for OPE. Then V opens the randomness and let P check the correctness of $\langle \Lambda^1 \rangle, \ldots, \langle \Lambda^k \rangle$. If they are valid, P opens $\langle b \rangle$ to continue with the execution of OPE. This allows the AntMan protocol to remove the possible leakage of secret polynomials, which is incurred by homomorphically performing polynomial evaluation upon incorrect ciphertexts.

AntMan++: Sublinear Designated-Verifier ZK. By applying our SIMD compiler to the original SIMD AntMan, we propose AntMan++, which is a more efficient VOLE-based ZK proof for arbitrary circuits. Similar to the original AntMan, we first batch arithmetic gates and prove their correctness. The generation of IT-PACs of all the wire values incurs $\mathcal{O}(|\mathcal{C}|/B)$ communication communication complexity. Additionally, checking the correctness of multiplication gates requires an opening of

Figure 4.7: Protocol for generating IT-PACs $\Pi^k_{\textsf{IT-PAC}}$

Let $\mathsf{AHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an additively homomorphic encryption scheme. Suppose that two parties $\mathcal{P}$ and $\mathcal{V}$ have already agreed a set of public parameters $\mathsf{par} = \mathsf{Setup}(1^\lambda)$ and global key $\Delta \in \mathbb{F}$. Let G be a PRG. Let k be the maximum degree of the polynomials committed in each IT-PAC.
INITIALIZE.

1. $\mathcal{V}$ samples $\mathsf{seed} \leftarrow \{0,1\}^\lambda$, and then $\mathcal{V}$ and $\mathcal{P}$ call the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$ with input seed, which returns a handle τ_1 to $\mathcal{P}$.
2. $\mathcal{V}$ samples $\Lambda \leftarrow \mathbb{F}$ and runs $\langle \Lambda^i \rangle \leftarrow \mathsf{Enc}(\mathsf{sk}, \Lambda^i; r_i)$ for all $i \in [1,k]$ where $(r_0, r_1, \ldots, r_k) = \mathsf{G}(\mathsf{seed})$ and $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{par}; r_0)$. Then, $\mathcal{V}$ sends the AHE ciphertexts $\langle \Lambda^1 \rangle, \ldots, \langle \Lambda^k \rangle$ to $\mathcal{P}$.

PRE-GEN. On input f,

3. $\mathcal{P}$ and $\mathcal{V}$ sends (extend) to $\mathcal{F}_{\mathsf{VOLE}}$, which returns u, w to $\mathcal{P}$ and v to $\mathcal{V}$, such that $w = v + u \cdot \Delta$.
4. On input polynomial $f(\cdot) = \sum_{i=0}^k f_i \cdot X^i \in \mathbb{F}[X]$, $\mathcal{P}$ computes a ciphertext $\langle b \rangle$ with $u + b = f(\Lambda)$ via $\langle b \rangle = \sum_{i=1}^k f_i \cdot \langle \Lambda^i \rangle + f_0 - u$.
5. P and V call the (Commit) command of $\mathcal{F}_{\mathsf{Com}}$ with inputs $\langle b \rangle$, which returns a handle τ_2 to V.

GEN. On input (τ_1, τ_2),

6. V and P call the (Open) command of $\mathcal{F}_{\mathsf{Com}}$ on input τ_1, which returns (seed, τ_1) to P. In parallel, $\mathcal{V}$ sends Λ to $\mathcal{P}$. Then, $\mathcal{P}$ computes $(r_0, r_1, \ldots, r_k) := \mathsf{G}(\mathsf{seed})$ and runs $\mathsf{sk} \leftarrow \mathsf{KeyGen}(\mathsf{par}; r_0)$. $\mathcal{P}$ checks that $\langle \Lambda^i \rangle = \mathsf{Enc}(\mathsf{sk}, \Lambda^i; r_i)$ for all $i \in [1,k]$, and aborts if the check fails. $\mathcal{P}$ sets $\mathsf{M} := w$.
7. $\mathcal{P}$ and $\mathcal{V}$ call the (Open) command of $\mathcal{F}_{\mathsf{Com}}$ on input τ_2, which returns $(\langle b_1 \rangle, \ldots, \langle b_\ell \rangle, \tau_2)$ to V. Then, $\mathcal{V}$ runs $b \leftarrow \mathsf{Dec}(\mathsf{sk}, \langle b \rangle)$, and then computes $\mathsf{K} := v - b \cdot \Lambda \in \mathbb{F}$.
8. Two parties obtain an IT-PAC $[f(\cdot)]$, where $\mathcal{P}$ holds $(f(\cdot), \mathsf{M})$ and $\mathcal{V}$ holds K.

size B.

The improvement of AntMan++ lies in the proof of wire consistency. As shown in Π_{compiler}, this problem is transferred into proof of linear map. And we use a random vector to further transfer linear-mapping proof into inner-product proof. In AntMan, we observe that the proof of the inner product between public and private vectors takes only $\mathcal{O}(B)$ communication overhead. Suppose the challenge vector $\boldsymbol{r}$ is public and witness $\boldsymbol{x}$ is private, and the IT-PACs of two vectors are known to both parties. After the secret evaluation point Λ is revealed, both parties can locally calculate $f_{\boldsymbol{r}}(\Lambda)$ because $\boldsymbol{r}$ is known. Via the additively homomorphic property of IT-PACs, both parties compute $f_{\boldsymbol{r}}(\Lambda) \cdot [\![\boldsymbol{x}]\!]$, which is also the IT-PAC of Hadamard product of $\boldsymbol{r}$ and $\boldsymbol{x}$. In this way, both parties compute $n + k$ IT-PACs and add them up to obtain $[\![\boldsymbol{q}]\!]$. In the end, according to the protocol in figure 4.3, both parties open the vector of size B and check whether their sum equals 0. As a result, the communication cost of AntMan++ is $\mathcal{O}(|\mathcal{C}|/B + B)$. When setting $B = |\mathcal{C}|^{1/2}$, it results in $\mathcal{O}(|\mathcal{C}|^{1/2})$.

The full description of SIMD AntMan is shown in Figure 4.6 and Figure 4.8.

Performance evaluation. We implement the AntMan++ protocol and benchmark its performance. Its homomorphic encryption (HE) is supported by the Microsoft SEAL [22] and other cryptographic building blocks are from EMP-toolkits [WMK16]. Two Amazon EC2 `m5.8xlarge` instances located in the same region are running as P and V. We manually throttle the network to simulate

Figure 4.8: The protocol of SIMDZK from AntMan (Cont.) Π_{AntMan} (Cont.)

PROVE: On input $(\mathcal{C}, [\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_m]\!])$, P and V do:

1. For each gate $(\alpha, \beta, \gamma, T)$ in $\mathcal{C}$, two parties holds IT-PAC of input wire vectors $[\![f]\!]$ and $[\![g]\!]$:
   - If $T = ADD$, both parties locally compute output IT-PAC $[\![h]\!] = [\![f]\!] + [\![g]\!]$.
   - If $T = MULT$, $\mathcal{P}$ computes a degree-$(2B-2)$ polynomial $\widetilde{h}(\cdot) := f(\cdot) \cdot g(\cdot) \in \mathbb{F}[X]$ and a degree-$(B-1)$ polynomial $h(\cdot)$ such that $h(\alpha_i) = \widetilde{h}(\alpha_i)$ for all $i \in [B]$. Then, P and V run sub-protocol $\Pi_{\mathsf{PAC}}^{(2B-2)}$ to generate two IT-PACs $[\![h(\cdot)]\!]$ and $[\![\widetilde{h}(\cdot)]\!]$.

   As there are n_2 multiplication gates, the commitments of their outputs are denoted as $[\![h_1]\!], \ldots, [\![h_{n_2}]\!]$. Consequently, their degree-$(2B-2)$ polynomials are denoted as $[\![\widetilde{h}_1]\!], \ldots, [\![\widetilde{h}_{n_2}]\!]$.
2. P samples two random polynomials $r(\cdot)$ and $s(\cdot)$ of respective degrees $B-1$ and $2B-2$ in $\mathbb{F}[X]$ such that $r(\alpha_i) = s(\alpha_i)$ for $i \in [1, t]$. Then, P and V generate the corresponding IT-PACs $[\![r(\cdot)]\!]$ and $[\![s(\cdot)]\!]$.
3. V samples seed $\leftarrow \{0,1\}^\lambda$ and sends it to P. Then, two parties compute $(\chi_1, \ldots, \chi_{n_2}) :=$ Hash(seed) $\in \mathbb{F}^{n_2}$.
4. P and V locally compute $[\![h(\cdot)]\!] := \sum_{j=1}^{n_2} \chi_j \cdot [\![h_j(\cdot)]\!] + [\![r(\cdot)]\!]$ and $[\![\widetilde{h}(\cdot)]\!] := \sum_{j=1}^{n_2} \chi_j \cdot [\![\widetilde{h}_j(\cdot)]\!] + [\![s(\cdot)]\!]$. Then, P sends the polynomial pair $(h(\cdot), \widetilde{h}(\cdot))$ to V, who checks that $h(\cdot), \widetilde{h}(\cdot)$ have the degrees $B-1$ and $2B-2$ respectively and $h(\alpha_i) = \widetilde{h}(\alpha_i)$ for all $i \in [1, t]$.
5. P and V run $\mathsf{Gen}(\tau_1, \tau_2)$ to open Λ to P, and then V can compute the local keys on all IT-PACs.
6. P and V run a VOLE-based zero-knowledge proof

$$\mathsf{DVZK}\left\{([\![f_j(\Lambda)]\!], [\![g_j(\Lambda)]\!], [\![\widetilde{h}_j(\Lambda)]\!])_{j\in[n_2]} \mid \forall j \in [n_2], \widetilde{h}_j(\Lambda) = f_j(\Lambda) \cdot g_j(\Lambda)\right\}.$$

7. P and V locally compute $[\mu] := [h(\Lambda)] - h(\Lambda)$ and $[\nu] := [\widetilde{h}(\Lambda)] - \widetilde{h}(\Lambda)$. Then, two parties run Open to check that $\mu = 0$ and $\nu = 0$.
8. Let $[\![v(\cdot)]\!]$ be the IT-PAC associated with the output values circuit $\mathcal{C}$. $\mathcal{P}$ and $\mathcal{V}$ run Open to check $v(\Lambda) = 0$.

If any check fails, $\mathcal{V}$ aborts.

low-bandwidth settings. We use the same 59-bit FFT-friendly field as the AntMan [WYY+22]. The performance of AntMan++ is not affected by the circuit structure and we benchmark with layered circuits for convenience. In all experiments, we randomly sample a circuit with 2^{16} input wires, 2^{27} addition gates and 2^{27} multiplication gates distributed at 2^{12} layers. We compare AntMan++ with the prior general VOLE-ZK Quicksilver [YSW+21] and use its default parameter setting in [WMK16]. We do not compare with AntMan [WYY+22] because it only proves SIMD circuits.

We first benchmark the running time and communication overhead with variable batch size $\log_2 B \in [9, 12]$. AntMan++ is split into the input-independent setup phase and online phase, and their performance is reported separately. As shown in Table 4.1, the increase of B leads to the significant reducing of the online communication overhead. The setup communication is dominated by HE ciphertexts and rotation keys. For the security of HE, the ciphertext size is fixed for all $\log_2 B \leq 11$ and start to increase when $B \geq 12$. The running time for both setup and online phases increase with B. The overhead mainly comes from the ciphertext rotation during the setup phase as well as the HE evaluation and polynomial multiplication during the online phase. Although its running time is $2.1\times \sim 3.2\times$ longer than Quicksilver, the bandwidth usage is $17.5\times \sim 83.6\times$ smaller.

Then we show the running time with the variable network bandwidth and the number of threads (Table 4.2). The batch size is fixed to be $B = 2^{11}$. AntMan++ is highly efficient in terms of network communication with asymptotically $\mathcal{O}(C/B)$ overhead. Its running time does not significantly deteriorate with the decreasing of bandwidth. On the other hand, AntMan++ is computationally heavy but fully parallelable, thus multi-threading is effective on increasing its throughput. When the number of threading is increased from 1 to 4, the running time is decreased by $36\% \sim 38\%$. Compared to Quicksilver, it requires 70% less running time when bandwidth is 10Mbps and 30% less when bandwidth is 25Mbps.

4.2 DV-NIZK from Public-Key PCF-based OT

In this section, we present our contribution to constructing an efficient designated-verifier non-interactive zero-knowledge (DV-NIZK) scheme based on public-key PCF-based OT. Our framework, described in Section 4.2.3, enables the direct construction of DV-NIZKs using public-key PCF-based OT in a black-box manner. To support the efficiency claims of our scheme, we outline our contribution of a new, efficient public-key PCF-based OT construction along with optimizations in Section 4.2.4. Finally, we provide concrete parameter choices for instantiating our PCF-based OT and DV-NIZK scheme in Section 4.2.5.

4.2.1 Motivations and Related Works

Generating pseudorandom correlations. Recently, a new paradigm has emerged which enables the *silent* generation of long correlated *pseudo*random strings [BCG+18; BCG+19b; BCG+19a], removing essentially all of the communication in the preprocessing phase. Concretely, this is made possible by the mean of cryptographic primitives, such as *pseudorandom correlation generators* (PCG) [BCG+19b] and *pseudorandom correlated functions* (PCFs) [BCG+20a].

A PCG is a pair of algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ where $\mathsf{PCG.Gen}$ produces two short keys $(\mathsf{k}_0, \mathsf{k}_1)$, and $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ produces a long string y_σ such that (y_0, y_1) form pseudorandom samples from the target correlation. PCGs enable silent secure computation as follows: using a small distributed protocol to securely generate the keys $(\mathsf{k}_0, \mathsf{k}_1)$, two parties can afterwards locally expand them into long correlated pseudorandom strings without any further communication. The online phase proceeds as before.

PCGs suffer from a considerable limitation: after distributing the keys, the parties are bound to generate *all at once* a priori *fixed amount* of correlated randomness. PCFs overcome this issue: a PCF is a pair of algorithms $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ where $\mathsf{PCF.Gen}$ produces two short keys $(\mathsf{k}_0, \mathsf{k}_1)$, and $\mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x)$ outputs y_σ^x where for each new input x, (y_0^x, y_1^x) appears like a fresh sample from the target correlation. Hence, after distributively generating the keys $(\mathsf{k}_0, \mathsf{k}_1)$ once and for all, two parties can generate on-the-fly any amount of target correlations in all their future secure computations.

The line of work on PCGs and PCFs has been fairly successful: modern PCG protocols for the oblivious transfer (OT) correlation (often called *silent OT extension*) can stretch up to 10M OT/s on one core of a standard laptop [CRR21; BCG+22; RRT23] from keys in the 10$\sim$20kB range, and the fastest PCFs for OT [BCG+22] can generate up to 100k OT/s on one core of a standard laptop.

Public-key silent OT. The silent generation of correlated randomness from PCGs or PCFs requires two parties to engage in an interactive protocol to securely generate the PCG/PCF keys. *Public-key* PCFs reduce this interactive phase to a bare minimum, by replacing it with a public-key setup. More precisely, after publishing their public keys online, any pair of parties on a network can start generating correlated randomness, without *any interaction* beyond the initial PKI. Public-key silent correlated randomness generation is somewhat of a holy grail in this line of work: it would represent a major step towards bridging the usability gap between secure communication (since PKI suffices to enable efficient pairwise secure communication) and secure computation, but public-key PCFs for OTs have so far proven considerably harder to achieve than standard PCG and PCFs. Until recently, we simply had no public key silent OT construction, beyond heavy-hammer constructions from obfuscation or threshold multikey FHE.

This changed recently with the result of [OSY21], which achieved the first practical public-key silent OT, assuming the quadratic residuosity assumption and the existence of correlation-robust hash functions. However, the efficiency of the new construction of [OSY21] still lags way behind that of state-of-the-art PCFs for OTs. Concretely, their construction relies on a new distributed discrete logarithm protocol that allows two parties, given multiplicative shares of a value G^x (where G generates a suitable DLog-easy group), to non-interactively compute additive shares of x. The public-key silent OT construction of [OSY21] has public keys of size around 1kB for one of the parties, and about 50kB for the other. In terms of computational efficiency, the cost of generating a single OT correlation is dominated by λ exponentiations with an exponent in $\mathbb{Z}_{N \cdot 2^\lambda}$, where N is an RSA modulus. Using $\lambda = 128$ and $\log N = 3072$, this translates to 128 exponentiations with 3200-bit exponents and takes about one second on one core of a standard laptop, which is between four and five orders of magnitude slower than the state-of-the-art PCF of [BCG+22]. In summary, as of today, the fundamental goal of obtaining concretely efficient and usable public-key silent OTs remains open.

Zero-knowledge proofs. ZKPs allow a prover to prove a statement is valid without revealing any information beyond its validity. NIZKs are ZKPs with a single flow from the prover to the verifier. It is a standard result [GO94] that NIZK proofs cannot exist in the plain model for all of NP. The standard way to circumvent this limitation is to let the prover and the verifier access a common random string which is generated by a trusted setup. There have been several works in realizing the notion of NIZK for general NP language from various assumptions such as trapdoor permutations which can be instantiated from factoring [BFM88] the Diffie-Hellman assumption over bilinear groups [GOS06] or Hidden-Bits Paradigm, indistinguishability obfuscation [SV14]. Recently, a recent line of work proposed several instantiations for the Fiat-Shamir transformation based on the hash function which satisfies a property called *correlation intractability*, then the Fiat–Shamir transform can be applied soundly to remove the interaction of proofs. Following this line of work, Peikert and Shiehian [PS19] gave a construction of NIZKs from plain Learning with Errors, and later Brakerski *et al.* [BKM20] recently showed that NIZKs can be constructed based on the hardness of both the learning parity with noise (LPN) assumption and the existence of trapdoor hash functions. Therefore, the remaining task would be how to construct efficient NIZKs from the minimal assumption. To investigate NIZKs based on minimal assumptions and to push NIZK schemes to the practical limit, several relaxations of NIZKs have been introduced, such as designated verifier NIZKs (DV-NIZKs) or designated prover NIZKs (DP-NIZKs), where a trusted third party additionally gives a secret verification key to the verifier or a secret proving key to the prover, or preprocessing NIZKs (PP-NIZKs), in which the trusted party generates both a secret verification key for the prover and a secret proving key for the prover.

DV-NIZK. A designated-verifier NIZK is a relaxed variant of NIZK where only the verifier who owns a secret key can verify the proof. In the DV-NIZK scheme, the trusted party generates a CRS together with a secret key which is given to the verifier and is used to verify whether a proof is accepted or rejected. Since whether the verifier accepts or rejects a proof depends on the secret verification key then two different DV-NIZKs settings have been proposed *one-time* DV-NIZKs and *reusable* DV-NIZKs; in *one-time* DV-NIZKs, the secret verification key can be only used for one proof of a statement while in *reusable* DV-NIZKs, the soundness still holds even the verifier uses a secret verification key for many proofs , i.e., the malicious prover learns nothing except the validity when producing many proofs for different statements and see whether they are accepted or rejected.

We explore an application of public-key PCF to *designated-verifier zero-knowledge proofs* (DV-NIZKs). A DV-NIZK allows any prover to demonstrate the truth of a statement using a single message, such that the proof can be verified using a secret verification key. DV-NIZKs are believed to be easier to obtain than standard NIZKs, in the following sense: they are known to exist under the plain CDH assumption in pairing-free groups [CH19; QRW19; KNY+19], while NIZKs are only known in pairing groups, or using subexponential hardness assumptions [JJ21; CJJ+23]. Yet, efficiency-wise, we do not know of any concretely efficient construction of DV-NIZKs in pairing-free groups (efficient NIZKs are known in pairing groups [GS08; KW15; CH20], and known DV-NIZKs in pairing-free groups rely on the hidden bit model, for which no concretely efficient instantiation is known). We show how, using a public-key PCF, one can compile any Σ-protocol with binary challenge into a DV-NIZK. Plugging our construction of public-key PCF, we obtain a new DV-NIZK from polynomial assumptions over pairing-free groups for all languages that admit a bit Σ-protocol, with communication comparable to that of the Σ-protocol. Conceptually, our result can be seen as observing that a public-key PCF suffices to upgrade *non-reusable* DV-NIZKs (which exist from public key encryption [CHH+07]) into *reusable* DV-NIZKs.

4.2.2 Detailed Contributions

In this section, taking advantage of *non-interactive* PK-PCF, we propose a new construction of *reusable* DV-NIZK argument of knowledge from a compiler that combines a sigma protocol for general NP language and a public-key pseudorandom correlation function (PK-PCF). Specifically, our *reusable* DV-NIZK comes from three ingredients:

- A Σ-protocol [CDS94] with 1-bit challenges for a language $\mathcal{L}$, for example Blum's protocol for graph Hamiltonicity [Blu86].
- A *strong* public key PCF for OT correlation where the key evaluation of each party can be *silently* obtained from their own secret key and public key of the other.
- A *non-reusable* DV-NIZK with computational adaptive soundness and adaptive zero-knowledge properties.

We also show how to enhance our construction to obtain a *reusable* DV-NIZK based on a *weak* public key PCF instead of the strong one. We state the formal result below.

Theorem 4.2.1 (informal)**.** *If there exists a weak public-key PCF then we can construct a reusable DV-NIZK argument of knowledge for any* NP *language* $\mathcal{L}$ *from a* Σ*-protocol for* $\mathcal{L}$ *and a non-reusable DV-NIZK scheme.*

The main idea behind our construction is the following. The designated verifier samples a PCF key pair $(\mathsf{sk}_V, \mathsf{pk}_V)$ and outputs a CRS containing their public key. A prover with statement x and witness w can then sample their own PCF key pair $(\mathsf{sk}_P, \mathsf{pk}_P)$ to produce a shared evaluation key with the designated prover. It then runs the Σ-protocol by computing a first message a. The challenge being binary, there are 2 possible third message for a transcript starting with a. We let z_b the third message for challenge $b \in \{0, 1\}$. Doing this λ-times lead to 2λ triplets $(a_i, b, z_{i,b})_{i\in[\lambda],b\in\{0,1\}}$. The prover then uses their PCF evaluation key to compute 2λ pseudorandom masks $r_{i,b}$ by evaluating the PCF on input $x|i|b$ (or $H(x|i|b)$ if the PCF is only weakly-secure). The prover finally outputs $\mathsf{pk}_P, (a_i, z_{i,0} \oplus r_{i,0}, z_{i,1} \oplus r_{i,1})$ as their proof.

The correctness of the PCF and Σ-protocol guarantee that the designated verifier can recover 1 mask out of each pair $(r_{i,0}, r_{i,1})$ and then can verify λ-transcripts, while security guarantees that the prover cannot predict which of the two is recovered by the verifier and that the non-revealed $r_{i,b}$ is pseudorandom, therefore providing soundness and zero-knowledge. A minor issue remains: one needs to prevent the prover to sample maliciously their PCF key pair such that it can predict the challenge bit. We show that it is sufficient to require the prover to additionally provide a proof (using a non-reusable DV-NIZK) that their PCF public key was generated from a honest execution of the PCF key generation algorithm (with possibly bad randomness).

Efficient Public-Key PCF for OT correlation. At the heart of our DV-NIZK construction is a new Public-Key PCF for OT correlations. It allows us to get good concrete efficiency for the DV-NIZK construction. At the time of writing, it is the state of art in term of efficiency and our PK-PCF is of independent interest.

4.2.3 Construction of Reusable DV-NIZK

Σ-protocol. A Σ-protocol is a 3-move, interactive proof (Figure 4.9) which consists of three algorithms $(\Sigma.P_1, \Sigma.P_2, \Sigma.V)$. While $(x, w) \in \mathcal{R}$, the prover inputs (x, w) to $\Sigma.P_1$ and produces a random message a to the verifier while a is considered as a commitment of x. The verifier then samples a random challenge bit b and sends it to the prover. The prover runs the algorithm $\Sigma.P_2(x, w, a, b)$ to produce a response z_b corresponding to bit challenge b and sends z_b to the verifier. Finally, the verifier runs $\Sigma.V(x, a, b, z_b)$ to decide whether accept or reject the proof.

Σ-protocol satisfies 3 properties: perfect completeness, 2-special soundness (given two accepting transcripts $(a, 0, z_0), (a, 1, z_1)$ of a statement x then there exists an efficient algorithm $\Sigma.\mathsf{Ext}(x, a, z_0, z_1)$ to extract the witness w) and special honest-verifier zero-knowledge (given b ahead of time, there exists a simulator $\mathsf{Sim}(x, b)$ simulating the transcript (a, b, z_b) without knowing a witness). The Blum's Hamiltonicity protocol [Blu86] is an instantiation of Σ-protocol with 1-bit challenges for NP language.

Figure 4.9: Σ-protocol with challenge space $\{0, 1\}$

$$\begin{array}{ccc} \mathsf{P} & & \mathsf{V} \\ (x,w)\in\mathcal{R} & & x \\ a \overset{\$}{\leftarrow} \Sigma.P_1(x,w) & \xrightarrow{\qquad a \qquad} & b \overset{\$}{\leftarrow} \{0,1\} \\ & \xleftarrow{\qquad b \qquad} & \\ z_b \overset{\$}{\leftarrow} \Sigma.P_2(x,w,a,b) & \xrightarrow{\qquad z_b \qquad} & \Sigma.V(x,a,b,z_b)\to\{0,1\} \end{array}$$

Public-key PCF. Informally, a PK-PCF is defined in the same way as a standard PCF except for the key generation Gen. In PK-PCF constructions, the Gen is instantiated by an algorithm which distributes *silently* the evaluation key to both parties. Particularly, each party σ first generates a secret key sk_σ then $\mathsf{PCF.Gen}(1^\lambda, \mathsf{crs}, \sigma, \mathsf{sk}_\sigma)$ gives each party σ a public key pk_σ for $\sigma \in [0, 1]$ such that after posting publicly the public keys $(\mathsf{pk}_0, \mathsf{pk}_1)$, the key derivation algorithm $\mathsf{PCF.KeyDer}$ is silently evaluated by each party to get a pair of evaluation keys $K_\sigma = \mathsf{PCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma})$

for $\sigma \in [0,1]$. We formula the general construction of PK-PCF for OT correlation in Figure 4.10. Note that in the DV-NIZK scheme, we take advantage of our PK-PCF construction for OT as a black-box for constructing our DV-NIZK.

Figure 4.10: PK-PCF for OT correlation

PARAMETERS. $\mathsf{PCF.Setup}(1^\lambda) \xrightarrow{\$} \mathsf{PCF.pp}$ which is accessed by both parties. Given a randomly public x is the point evaluation of PCF.

INTERACTION PHASE. For $\sigma \in [0,1]$:

- Compute $\mathsf{PCF.Gen}_\sigma(1^\lambda, \mathsf{crs}) \to (\mathsf{pk}_\sigma, \mathsf{sk}_\sigma)$ then party σ publishes pk_σ.
- Party σ derivates its evaluation key as $K_\sigma \leftarrow \mathsf{PCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma})$ for $\sigma \in [0,1]$.

EVALUATION PHASE. $\mathsf{PCF.Eval}(1^\lambda, \sigma, K_\sigma, x)$:

- If $\sigma = 0$, $\mathsf{PCF.Eval}(1^\lambda, \sigma, K_\sigma, x) = (r_0, r_1)$.
- If $\sigma = 1$, $\mathsf{PCF.Eval}(1^\lambda, \sigma, K_\sigma, x) = (b, r_b)$.

To apply PK-PCF to our reusable DVNIZK scheme, we provide a stronger security definition of PK-PCF. The formal definition is shown below:

Strong Security. Define $\mathsf{Supp}(\mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, \sigma))$ be a constraint relation such that for all $(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma) \in \mathsf{Supp}(\mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, \sigma))$ then there exists a public coin ρ such that $(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma) = \mathsf{PCF.Gen}'(1^\lambda, \mathsf{pp}, \sigma, \rho)$ ($\mathsf{PCF.Gen}'$ is deterministic algorithm defined by adding a public coin to $\mathsf{PCF.Gen}$ to make $\mathsf{PCF.Gen}$ deterministic).

For every $\sigma \in \{0,1\}$ and every non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$, it holds that for all sufficiently large λ,

$$|\Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda) = 1]| \leq \epsilon(\lambda)$$

where $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,b}$ ($b \in \{0,1\}$) is defined as in Figure 4.11. In particular, the adversary is given access to $N(\lambda)$ samples. Here, $\mathsf{RSample}$ is the algorithm for reverse sampling $\mathcal{Y}$ as in the Definition 2.7.1.

Non-reusable DV-NIZK arguments. Let $\mathcal{L}_P$ be an NP language associated with an NP relation $\mathcal{R}_P$. We take advantage of a (one-time) DV-NIZK scheme for $\mathcal{L}_P$ which consists of three algorithms $\mathsf{dv} = (\mathsf{dv.Setup}, \mathsf{dv.P}, \mathsf{dv.V})$. For convenience, we define the notion of a DV-NIZK as below.

- $\mathsf{dv.Setup}(1^\lambda) \to (\mathsf{dv.crs}, \mathsf{dv}.\mathcal{T})$.
- $\mathsf{dv.P}(\mathsf{dv.crs}, x, w) \to \mathsf{dv}.\pi$.
- $\mathsf{dv.V}(\mathsf{dv.crs}, x, \mathsf{dv}.\pi, \mathsf{dv}.\mathcal{T}) \to \{0,1\}$.

The dv scheme needs to satisfy perfect completeness, computational (bounded) adaptive knowledge soundness and computational zero-knowledge properties. The non-reusable DV-NIZK can be constructed from a public-key encryption scheme [PsV06] by λ invocations of Σ-protocol. In our construction, dv is used to prove that the prover uses the correct form of pk_P to generate the proof. Specifically, for a statement pk_P and a witness sk_P we consider $\mathsf{pk}_P \in \mathcal{L}_p$, $(\mathsf{pk}_P, \mathsf{sk}_P) \in \mathcal{R}_P$ if $(\mathsf{pk}_P, \mathsf{sk}_P) \in \mathsf{Supp}(\mathsf{PCF.Gen}_0(1^\lambda, \mathsf{PCF.pp}))$.

Figure 4.11: Strong security of a PK-PCF

```
$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda):}$
$\mathsf{pp} \leftarrow \mathsf{PCF.Setup}(1^\lambda)$
$(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma) \leftarrow \mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, \sigma)$
$(\mathsf{sk}_{1-\sigma}, \mathsf{pk}_{1-\sigma}) \leftarrow \mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, 1-\sigma)$
$k_{1-\sigma} \leftarrow \mathsf{PCF.KeyDer}(1-\sigma, \mathsf{sk}_{1-\sigma}, \mathsf{pk}_\sigma)$
**For** $i = 1, \ldots, N(\lambda)$ :
    $x^{(i)} \leftarrow\$ \{0,1\}^{n(\lambda)}$
    $y^{(i)}_{1-\sigma} \leftarrow \mathsf{PCF.Eval}(1-\sigma, k_{1-\sigma}, x^{(i)})$
For all $i \in [N(\lambda)]$:
$b \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}_0, \mathsf{pk}_1, \sigma, \mathsf{sk}_\sigma, (x^{(i)}, y^{(i)}_{1-\sigma}))$
**Output** $b$
```

```
$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda):}$
$\mathsf{pp} \leftarrow \mathsf{PCF.Setup}(1^\lambda)$
$\forall \mathsf{pk}_\sigma$ such that: $\exists \mathsf{sk}_\sigma$,
$(\mathsf{sk}_\sigma, \mathsf{pk}_\sigma) \in \mathsf{Supp}(\mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, \sigma))$
$(\mathsf{sk}_{1-\sigma}, \mathsf{pk}_{1-\sigma}) \leftarrow \mathsf{PCF.Gen}(1^\lambda, \mathsf{pp}, 1-\sigma)$
$k_\sigma \leftarrow \mathsf{PCF.KeyDer}(\sigma, \mathsf{sk}_\sigma, \mathsf{pk}_{1-\sigma})$
**For** $i = 1, \ldots, N(\lambda)$ :
    $x^{(i)} \leftarrow\$ \{0,1\}^{n(\lambda)}$
    $y^{(i)}_\sigma \leftarrow \mathsf{PCF.Eval}(\sigma, k_\sigma, x^{(i)})$
    $y^{(i)}_{1-\sigma} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y^{(i)}_\sigma)$
For all $i \in [N(\lambda)]$:
$b \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}_0, \mathsf{pk}_1, \sigma, \mathsf{sk}_\sigma, (x^{(i)}, y^{(i)}_{1-\sigma}))$
**Output** $b$
```

Reusable DV-NIZK from PK-PCF

We show how to combine a Σ-protocol (Figure 4.9), a *non-reusable* DV-NIZK, and a *strong* public-key PCF to obtain a *reusable* DV-NIZK scheme for all NP language. Let $\mathcal{R}$ be an NP language, $(x, w) \in \mathcal{R}$, then our re-usable DV-NIZK scheme $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ is defined as detailed in Figure 4.12.

The reusable knowledge soundness of our DV-NIZK construction (Figure 4.12) follows from the special soundness of Σ-protocol, knowledge soundness of DV-NIZK to extract a valid witness with high probability whenever prover outputs an accepted proof. In particular, we build a simulator can simulate the verification query without knowing the sk_V and an efficient extractor Ext can extract a valid witness by using $\mathsf{dv.Ext}$ to get the sk_P then later use $\Sigma.\mathsf{Ext}$ with accepted transcripts to extract witness w.

If the statement x is false, in each invocation i of Σ-protocol, then for each a_i there is only one challenge b that can have a valid response $z_{i,b}$. In our construction, $b \in [0, 1]$ is pseudorandomly generated by the output of PCF, and the prover learns nothing about the bit b if the pk_P is generated honestly corresponding with the secret sk_P, note that this condition is maintained by using the DV-NIZK scheme, so the prover can only cheat in Σ-protocol with a probability of $(1/2)^k$ after k invocations by correctly guessing the bit challenge b.

Our DV-NIZK scheme (Figure 4.12) is zero-knowledge because there exists a simulator knowing ahead of time the challenge $b \in [0, 1]$ that can simulate the view of the verifier $(a_i, b, z_{i,b})$ without knowing the witness. Therefore, the view of the verifier in Π can be simulated by a simulator that for each i-invocation of the proof defines $m_{i,b} := z_{i,b} \oplus r_{i,b}$ and picks a dummy item for $m_{i,1-b}$.

Theorem 4.2.2 (Completeness). *If* dv, Σ *are complete and* PCF *is correct then DV-NIZK scheme from Figure 4.12 is complete.*

Proof. Consider $(x, w) \in \mathcal{R}$, by completeness of the non-reusable DV-NIZK, $\mathsf{dv.V}(\mathsf{dv.crs}, \mathsf{pk}_P, \mathsf{dv.}\pi, \mathsf{dv.}\mathcal{T})$

Figure 4.12: Reusable DV-NIZK $\Pi(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$

- $\mathsf{Setup}(1^\lambda)$.
  - Compute $\mathsf{PCF.pp} \leftarrow \mathsf{PCF.Setup}(1^\lambda)$, $(\mathsf{pk}_V, \mathsf{sk}_V) \leftarrow \mathsf{PCF.Gen}_1(1^\lambda, \mathsf{PCF.pp})$ and $(\mathsf{dv.crs}, \mathsf{dv}.\mathcal{T}) \leftarrow \mathsf{dv.Setup}(1^\lambda)$.
  - Return $\mathsf{crs} := (\mathsf{PCF.pp}, \mathsf{dv.crs}, \mathsf{pk}_V)$ and $\mathcal{T} := (\mathsf{sk}_V, \mathsf{dv}.\mathcal{T})$.
- $\mathsf{P}(\mathsf{crs}, x, w)$.
  1. Compute $(\mathsf{pk}_P, \mathsf{sk}_P) \leftarrow \mathsf{PCF.Gen}_0(1^\lambda, \mathsf{PCF.pp})$, $k_0 \leftarrow \mathsf{PCF.KeyDer}(0, \mathsf{sk}_P, \mathsf{pk}_V)$ and generate $\mathsf{dv}.\pi = \mathsf{dv.P}(\mathsf{dv.crs}, \mathsf{pk}_P, \mathsf{sk}_P)$.
  2. For each $i \in [1, \lambda]$ generate all Σ-protocol transcripts for both challenges $b \in [0, 1]$:
  $$a_i = \Sigma.P_1(x, w)$$
  $$z_{i,b} = \Sigma.P_2(x, w, a_i, j) \text{ for } b \in [0, 1]$$
  3. Compute $(r_{i,0}, r_{i,1}) = \mathsf{PCF.Eval}(1^\lambda, 0, k_0, x)$ and for each $i \in [1, \lambda]$, $b \in [0, 1]$ define $m_{i,b} = z_{i,b} \oplus r_{i,b}$.
  4. Define the output $\pi := (\mathsf{pk}_P, \mathsf{dv}.\pi, \{a_i, m_{i,0}, m_{i,1}\}_{i \le \lambda})$.
- $\mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T})$.
  1. Check if $\mathsf{dv.V}(\mathsf{dv.crs}, \mathsf{pk}_P, \mathsf{dv}.\pi, \mathsf{dv}.\mathcal{T}) = 1$ then

     Compute $k_1 \leftarrow \mathsf{PCF.KeyDer}(1, \mathsf{sk}_V, \mathsf{pk}_P)$ and for each $i \in [1, \lambda]$, compute as follows:
     $$\mathsf{PCF.Eval}(1^\lambda, 1, k_1, x) = (b_i, r_{i,b_i}), \quad z_{i,b_i} = m_{i,b_i} \oplus r_{i,b_i}$$
     The output is defined $\mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T}) = 1$ if $\Sigma.V(x, a_i, b_i, z_{i,b_i}) = 1$ for all $i \in [1, \lambda]$.
  2. Otherwise, $\mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T}) \to 0$.

accepts for all pair $(\mathsf{pk}_P, \mathsf{sk}_P) \in \mathcal{R}_P$ which is correctly generated and by correction of the PCF and completeness of Σ-protocol, $\Sigma.V(x, a_i, b_i, z_{i,b_i})$ also accept for all $i \in [1, \lambda]$. □

Theorem 4.2.3 (Knowledge soundness). *If* dv *is adaptive knowledge sound,* Σ *is 2-special sound and* PCF *for OT correlation satisfies strong PK-PCF security property then DV-NIZK scheme from Figure 4.12 is reusable adaptive knowledge soundness.*

Proof. We describe an efficient simulator Sim that correctly emulates the verifier without knowing about sk_V. The simulator is done as follows:

- $\mathsf{Sim.Setup}(1^\lambda)$: compute $\mathsf{PCF.Setup}(1^\lambda) \to \mathsf{PCF.pp}$, $\mathsf{PCF.Gen}_1(1^\lambda, \mathsf{PCF.pp}) \to (\mathsf{pk}_V, \mathsf{sk}_V)$ and $\mathsf{dv.Setup}(1^\lambda) \to (\mathsf{dv.crs}, \mathsf{dv}.\mathcal{T})$, define the output $(\mathsf{PCF.pp}, \mathsf{dv.crs}, \mathsf{pk}_V) \to \mathsf{crs}$, a trapdoor $\mathcal{T} := (\mathsf{sk}_V, \mathsf{dv}.\mathcal{T})$ and erase sk_V.
- $\mathsf{Sim.V}(\mathsf{crs}, x, \pi, \mathsf{dv}.\mathcal{T})$: parse $\pi = (\mathsf{pk}_P, \mathsf{dv}.\pi, \{a_i, m_{i,0}, m_{i,1}\}_{i \le \lambda})$, use the $\mathsf{dv.Ext}$ to extract the sk_P , i.e., $\mathsf{sk}_P \leftarrow \mathsf{dv.Ext}(\mathsf{crs}, \mathsf{pk}_P, \mathsf{dv}.\mathcal{T})$. From sk_P, for each $i \in [1, \lambda]$, compute $k_0 \leftarrow \mathsf{PCF.KeyDer}(0, \mathsf{sk}_P, \mathsf{pk}_V)$, $(r_{0,i}, r_{1,i}) \leftarrow \mathsf{PCF.Eval}(1^\lambda, 0, k_0, x)$, $(b_i, r_{i,b_i}) \leftarrow \mathsf{RSample}(1^\lambda, 0, (r_{0,i}, r_{1,i}))$ then define $(z_{0,i}, z_{1,i}) = (m_{i,0} \oplus r_{0,i}, m_{i,1} \oplus r_{1,i})$.

  Firstly, check that if $(\mathsf{pk}_P, \mathsf{sk}_P) \notin \mathcal{R}_P$ then outputs 0. Otherwise, continue to check $\Sigma.V(x, a_i, b_i, z_{i,b_i}) = 1$ for all $i \in [1, \lambda]$. If all checks succeeded, accept (output 1). Otherwise, reject (output 0). To extract a witness, pick $i \leftarrow\$ [1, \lambda]$ then define $w \leftarrow \Sigma.\mathsf{Ext}(x, a_i, z_{i,0}, z_{i,1})$.

□

The simulator Sim first calls $\mathsf{Sim.Setup}(1^\lambda)$ to generate crs, and store $\mathsf{dv}.\mathcal{T}$. Each time the $\mathcal{A}$ sends a query (x, π) to the oracle $\mathcal{O}(\mathsf{crs}, ., ., \mathcal{T})$, Sim simulates $\mathcal{O}(\mathsf{crs}, ., ., \mathcal{T})$ (without knowing sk_V) by running $\mathsf{Sim.V}(\mathsf{crs}, x, \pi, \mathsf{dv}.\mathcal{T})$ and outputs whatever Sim.V outputs. When $\mathcal{A}$ outputs a final answer (x^*, π^*), Sim computes a witness w as in Sim.V.

We prove the following claim: for any input (x, π), it hold that

$$\Pr\left[\begin{array}{c}(\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Setup}(1^\lambda)\\ b \leftarrow \mathsf{Sim.V}(\mathsf{crs}, x, \pi, \mathsf{dv}.\mathcal{T})\\ b' \leftarrow \mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T})\end{array} : b = b'\right] \approx 1$$

We show that if $b = 1$, then $b' = 1$ with overwhelming probability. Indeed, if $b = 1$ it means

- $(\mathsf{pk}_P, \mathsf{sk}_P) \in \mathcal{R}_P$ then $\mathsf{dv.V}(\mathsf{dv.crs}, \mathsf{pk}_P, \mathsf{dv}.\pi, \mathsf{dv}.\mathcal{T}) = 1$.
- $\Sigma.V(x, a_i, b_i, z_{i,b_i}) = 1$ for all $i \in [1, \lambda]$ where $b_i \leftarrow\$ \{0, 1\}$ (output of Rsample) then by the soundness of Σ-protocol, we have $(x, w) \in \mathcal{R}$ with a probability of at least $1 - 1/2^\lambda$. This leads to $\Sigma.V(x, a_i, b_i, z_{i,b_i}) = 1$ for all $i \in [1, \lambda]$ and $b_i \in \{0, 1\}$ (output of PCF.Eval) with a probability of at least $1 - 1/2^\lambda$.

Therefore, $\mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T}) = 1$, i.e., if Sim's checks succeeding then the verifier's checks necessarily succeed with high probability. In particular, the probability that the Sim accepts the proof while the verifier rejects it is at most $\epsilon_\Sigma = 1/2^\lambda$.

We next prove that if $b' = 1$ then $b = 1$ with high probability. Assume the Sim rejects the proof while the verifier accepts it. Let denote ϵ as the probability of verifier that accepts the proof. Since Sim rejects the proof then at least one of checks must fail: either $(\mathsf{pk}_P, \mathsf{sk}_P) \notin \mathcal{R}_P$ or $\exists i \in [1, \lambda] :\ \Sigma.V(x, a_i, b_i, z_{i,b_i}) = 0$ (b_i is output of RSample). Because the verifier accepts the proof then $\mathsf{dv.V}(\mathsf{dv.crs},\ \mathsf{pk}_P, \mathsf{dv}.\pi, \mathsf{dv}.\mathcal{T}) = 1$ and $\Sigma.V(x, a_i, b_i, z_{i,b_i}) = 1$ for all $i \in [1, \lambda]$ and b_i is output of PCF.Eval (computed honestly).

- By the knowledge soundness of dv then

$$\Pr\left[\ (\mathsf{pk}_P, \mathsf{sk}_P) \notin \mathcal{R}_P \ \middle|\ \begin{array}{c}\mathsf{dv.V}(\mathsf{dv.crs}, \mathsf{pk}_P, \mathsf{dv}.\pi, \mathsf{dv}.\mathcal{T}) = 1\\ \mathsf{sk}_P \leftarrow \mathsf{dv.Ext}(\mathsf{crs}, \mathsf{pk}_P, \mathsf{dv}.\pi, \mathsf{dv}.\mathcal{T})\end{array}\right] = \epsilon_{\mathsf{dv}}$$

- By the security of PCF, $\Sigma.V(x, a_i, b_i, z_{i,b_i}) = 1$ for all $i \in [1, \lambda]$ and b_i is output of PCF.Eval (computed honestly) can be simulated indistinguishable by $\Sigma.V(x, a_i, b_i, z_{i,b_i}) = 1$ for all $i \in [1, \lambda]$ and b_i is output of RSample, i.e., a random bit. After that, since b_i is uniformly random then by the soundness of Σ- protocol, with high probability $(x, w) \in \mathcal{R}$.

  For convenience, we denote the event $\Sigma.V(x, a_i, b_i, z_{i,b_i}) = 1 \forall i \in [1, \lambda]$ where $b_i \leftarrow\$ \{0, 1\}$ as Ver and $\exists i \in [1, \lambda] :\ \Sigma.V(x, a_i, b_i, z_{i,b_i}) = 0$ where $b_i \leftarrow\$ \{0, 1\}$ as Bad. Then from the security of PCF and the soundness of sigma protocol, we have:

$$\Pr[\mathsf{Ver}] \geq \epsilon - \epsilon_{\mathsf{PCF}} \quad \text{and} \quad \Pr\left[\ \mathsf{Ver} \ \middle|\ \mathsf{Bad}\ \right] \leq \epsilon_\Sigma$$

  Observe that:

$$\begin{aligned}\Pr[\mathsf{Ver}] &= \Pr[\mathsf{Ver} \wedge \mathsf{Bad}] + \Pr[\mathsf{Ver} \wedge \overline{\mathsf{Bad}}]\\ &= \Pr[\mathsf{Ver} \mid \mathsf{Bad}] \cdot \Pr[\mathsf{Bad}] + \Pr[\mathsf{Ver} \wedge \overline{\mathsf{Bad}}] \leq \epsilon_\Sigma + \Pr[\mathsf{Ver} \wedge \overline{\mathsf{Bad}}]\end{aligned}$$

Then we obtain $\Pr[\mathsf{Ver} \wedge \overline{\mathsf{Bad}}] \geq \epsilon - \epsilon_{\mathsf{PCF}} - \epsilon_{\Sigma}$. Putting everything together, the verifier accepts a proof with probability of ϵ then the simulator also accept this proof with probability of at least $\epsilon - \epsilon_{\mathsf{PCF}} - \epsilon_{\Sigma} - \epsilon_{\mathsf{dv}}$.

In conclusion, we have:

$$\Pr\left[\begin{array}{c} (\mathsf{crs}, \mathcal{T}) \leftarrow \mathsf{Setup}(1^\lambda) \\ b \leftarrow \mathsf{Sim.V}(\mathsf{crs}, x, \pi, \mathsf{dv}.\mathcal{T}) \\ b' \leftarrow \mathsf{V}(\mathsf{crs}, x, \pi, \mathcal{T}) \end{array} : b = b' \right] \geq 1 - \mu$$

where $\mu = \epsilon_{\mathsf{dv}} + 2.\epsilon_{\Sigma} + \epsilon_{\mathsf{PCF}} = \mathsf{negl}(\lambda)(\lambda)$.

Now consider an $\mathcal{A}$ that outputs an accepting proof with probability of at least ϵ after Q polynomial times queries to oracle $\mathcal{O}(\mathsf{crs}, ., ., \mathcal{T})$. By the above claim, $\mathcal{A}$ outputs an accepting proof with probability of at least $\epsilon - Q.\mu$ after interacting Q times with $\mathsf{Sim.V}(\mathsf{crs}, x, \pi, \mathsf{dv}.\mathcal{T})$; moreover, with probability at least $1 - \mu'$ ($\mu' = \epsilon_{\mathsf{dv}} + \epsilon_{\Sigma} + \epsilon_{\mathsf{PCF}}$), this proof is also accepted by Sim 's verification algorithm. Overall, Sim obtains a proof accepted by his verification algorithm with probability at least $\approx \epsilon - (Q+1)\mu$. In particular, this implies that w extracted by Sim from π satisfies $(x, w) \in \mathcal{R}$ with probability at least $\epsilon - (Q+1)\mu$. Therefore, Sim extracts a valid witness with probability at least $\epsilon - (Q+1)\mu$. As $(Q+1)\mu = \mathsf{negl}(\lambda)(\lambda)$, we conclude that if $\mathcal{A}$ outputs an accepting proof with non-negligible probability, then Sim extracts a valid witness with non-negligible probability.

Theorem 4.2.4 (Zero Knowledge). *If* dv *is adaptive single-theorem ZK,* Σ *is special-honest verifier ZK and strong* PCF *satisfies pseudorandom* $\mathcal{Y}$*-output property then DV-NIZK scheme from Figure 4.12 is adaptive multi-theorem ZK.*

Proof. Denote Sim_{Σ} as the efficient simulator for Σ-protocol to prove HVZK property. Formally, given a challenger b_i ahead of time, Sim_{Σ} can output an accepting proof without knowing the witness , i.e., $\mathsf{Sim}_{\Sigma}(x, b_i) \to (a, b_i, z_{b_i})$ such that $\Sigma.V(x, a, b_i, z_{b_i}) = 1$. And for (one time) DV-NIZK scheme, there exists a simulator $\mathsf{Sim}_{\mathsf{dv}} = (S_1^{\mathsf{dv}}, S_2^{\mathsf{dv}})$ which is used to prove the zero-knowledge of scheme where $(\mathsf{dv.crs}, \mathsf{dv}.\mathcal{T}) \leftarrow S_1^{\mathsf{dv}}(1^\lambda)$ outputs a simulated common reference string and a simulator trapdoor and $\mathsf{dv}.\pi \leftarrow S_2^{\mathsf{dv}}(\mathsf{dv.crs}, \mathsf{dv}.\mathcal{T}, x)$ outputs an accepted proof.

We now construct a zero-knowledge simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ below:

- $\mathcal{S}_1(1^\lambda) \to (\mathsf{crs}, \mathcal{T})$. On inputs 1^λ and outputs $\mathsf{crs} \leftarrow (\mathsf{PCF.pp}, \mathsf{dv.crs}, \mathsf{pk}_V)$ and a trapdoor $\mathcal{T} := (\mathsf{sk}_V, \mathsf{dv}.\mathcal{T})$ such that
  - $\mathsf{PCF.Setup}(1^\lambda) \to \mathsf{PCF.pp}$, $\mathsf{PCF.Gen}_1(1^\lambda, \mathsf{PCF.pp}) \to (\mathsf{pk}_V, \mathsf{sk}_V)$
  - $\mathsf{Sim}^1_{\mathsf{dv}}(1^\lambda) \to (\mathsf{dv.crs}, \mathsf{dv}.\mathcal{T})$
- $\mathcal{S}_2(\mathsf{crs}, \mathcal{T}, x) \to \pi$. On inputs $(\mathsf{crs}, \mathcal{T}, x)$, $\mathcal{S}_2$ randomly chooses a pair $(\mathsf{pk}_P, \mathsf{sk}_P) \in \mathcal{R}_P$, computes $\mathsf{PCF.KeyDer}(1, \mathsf{sk}_V, \mathsf{pk}_P) \to k_1$ and for each invocation $i \in [1, \lambda]$, computes as below:

$$\mathsf{PCF.Eval}(1^\lambda, 1, k_1, x) = (b_i, r_{i,b_i})$$

  $\mathcal{S}_2$ uses the simulator $\mathsf{Sim}_{\Sigma}(x, b_i)$ to get an accepting transcript (a_i, b_i, z_{i,b_i}) for each $i \in [1, \lambda]$. $\mathcal{S}_2$ now defines the output:

$$\pi := (\mathsf{pk}_P, \mathsf{dv}.\pi, \{a_i, m_{i,0}, m_{i,1}\}_{i \leq \lambda})$$

  where $\mathsf{dv}.\pi = \mathsf{dv.P}(\mathsf{dv.crs}, \mathsf{pk}_P, \mathsf{sk}_P)$ and for each $i \in [1, \lambda]$, $m_{i,b_i} = z_{i,b_i} \oplus r_{i,b_i}$ otherwise $m_{i,1-b_i}$ is picked randomly.

To complete the proof, we consider a sequence of hybrid experiments:

- Hyb_0. Namely, at the beginning of the game, the challenger samples $\Pi.\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}, \mathcal{T})$ and gives crs to the adversary, where $\mathsf{crs} \leftarrow (\mathsf{PCF.pp}, \mathsf{dv.crs}, \mathsf{pk}_V)$ and $\mathcal{T} := (\mathsf{sk}_V, \mathsf{dv}.\mathcal{T})$. When the adversary makes a verification query on $(x, w) \in \mathcal{R}$, the challenger replies with $\Pi.\mathsf{P}(\mathsf{crs}, x, w) = (\mathsf{pk}_P, \mathsf{dv}.\pi, \{a_i, m_{i,0}, m_{i,1}\}_{i \le \lambda})$.

- Hyb_1. Same as Hyb_0, except we replace:

  - $(\mathsf{dv.crs}, \mathsf{dv}.\mathcal{T}) \in \Pi.\mathsf{Setup}(1^\lambda)$ by $(\mathsf{dv.crs}', \mathsf{dv}.\mathcal{T}') \leftarrow \mathcal{S}_1^{\mathsf{dv}}(1^\lambda)$
  - $\mathsf{dv}.\pi \in \pi$ by $\mathsf{dv}.\pi' \leftarrow S_2^{\mathsf{dv}}(\mathsf{dv.crs}', \mathsf{dv}.\mathcal{T}', \mathsf{pk}_P)$

  This is computational indistinguishable from Hyb_0 by the the zero-knowledge of dv.

- Hyb_2. Same as Hyb_1, except the challenger simulates $\{a_i', (m_{i,0}', m_{i,1}')_{i \le \lambda}\} \in \pi$ by the following way. For each $i \in [1, \lambda]$,

  - Compute $k_1 \leftarrow \mathsf{PCF.KeyDer}(1, \mathsf{sk}_V, \mathsf{pk}_P)$, $(b_i, r_{i,b_i}) \leftarrow \mathsf{PCF.Eval}(1^\lambda, 1, k_1, x)$
  - Use Sim_Σ for simulating (a_i', c_{i,b_i}') , i.e., compute $(a_i', b_i, c_{i,b_i}') \leftarrow \mathsf{Sim}_\Sigma(x, b_i)$
  - Define $m_{i,b_i}' := c_{i,b_i}' \oplus r_{i,b_i}$ and $m_{i,1-b_i}'$ is picked randomly

  This is computationally indistinguishable from Hyb_1 by the pseudorandom output of PCF and HVZK of Σ-protocol. Indeed, the adversary can only compute the value of (b_i, r_{i,b_i}) by the $\mathsf{PCF.Eval}$ and from the view of adversary $r_{i,1-b_i}$ is pseudorandom then $m_{i,1-b_i}$ also for all $i \in [1, \lambda]$. Moreover, for all $(x, w) \in \mathcal{R}$, with the $\Pi.\mathsf{Setup}(1^\lambda) := (\mathsf{crs}', \mathcal{T}')$ where $\mathsf{crs}' = \mathsf{PCF.pp}, \mathsf{dv.crs}', \mathsf{pk}_V)$, $\mathcal{T}' = (\mathsf{sk}_V, \mathsf{dv}.\mathcal{T}')$ and prove $\pi' = (\mathsf{pk}_P, \mathsf{dv}.\pi', \{a_i', m_{i,0}', m_{i,1}'\}_{i \le \lambda})$, the adversary always accepts it , i.e., $\Pi.\mathsf{V}(\mathsf{crs}', x, \mathcal{T}', \pi') = 1$.

□

4.2.4 Efficient Public-Key PCF-based OT

In this section, we describe our paradigm about constructing PCF for OT correlations from Pseudorandomly Constrained PRFs. Then, we explain how to modify the Naor-Reingold PRF in order to obtain a CPRF for the class of inner-product membership predicates, and show that two weak PRF constructions can be expressed as such predicates, leading to instantiations of pseudorandomly constrained PRFs and therefore of PCFs. Next, we provide several optimizations which benefit the most efficient instantiations of our paradigm and detail our optimized PCF construction.

We then describe how our PCF can be turned into an efficient public-key PCF by relying on ideas borrowed from [OSY21], and how to optimize the resulting construction. This section is a briefly description about our new construction of PK-PCFs for OT correlation. We refer the reader to the full version in [BCM+24] for a detailed explanation and the formal proofs of security.

A PCF for OT from Pseudorandomly Constrained PRFs

Let $F = (F.\mathsf{KeyGen}, F.\mathsf{Eval})$ be a (weak, strong) PRF with key space $\mathcal{K}$ and binary outputs. For a key $K \in \mathcal{K}$, let $F_K : x \mapsto F.\mathsf{Eval}(K, x)$. Also, let $\mathsf{CPRF} = (\mathsf{CPRF.KeyGen}, \mathsf{CPRF.Eval}, \mathsf{CPRF.Constrain}, \mathsf{CPRF.CEval})$ denote a constrained PRF for the class $\mathcal{F} = \{F_K\}_{K \in \mathcal{K}} \cup \{1 - F_K\}_{K \in \mathcal{K}}$, i.e., $\mathcal{F}$ contains all predicates "$F.\mathsf{Eval}(K, x)$ evaluates to b" for $b \in \{0, 1\}$ and $K \in \mathcal{K}$. Then, we construct a (weak, strong) pseudorandom correlation function for oblivious transfer correlation as follows:

- The sender gets two independent master secret keys $(\mathsf{msk}_0, \mathsf{msk}_1)$ of the CPRF. On an input x, this party evaluates the CPRF on x using both keys to obtain two pseudorandom outputs (y_0, y_1).
- The receiver gets a random (weak) PRF key $K \overset{\$}{\leftarrow} \mathcal{K}$, and two constrained keys: ck_0 that is msk_0 constrained at "$F_K(x) = 0$", and ck_1 that is msk_1 constrained at "$F_K(x) = 1$". On an input x, this party computes $b \leftarrow F_K(x)$, and sets $y_b \leftarrow \mathsf{CPRF.CEval}(\mathsf{ck}_b, x)$. It then outputs (b, y_b).

Correctness is straightforward: for any x, the predicate $F_K(x) = b$ is satisfied for some $b \in \{0,1\}$, hence the constrained key ck_b yields the correct output y_b by the correctness of the CPRF. *Sender security* follows from the fact that the two constrained keys are constrained at F_K and $1 - F_K$, respectively, hence both constrains can never be satisfied at the same time. Thus, by the security of the CPRF, when $F_K(x) = b$, the value y_{1-b} is indistinguishable from random for the receiver. *Receiver security* follows from the (weak) pseudorandomness of F, which entails that $b = F_K(x)$ is pseudorandom from the sender's perspective.

We sketch the full construction below (we omit the public parameters pp output by the CPRF for simplicity):

- $\mathsf{PCF.Gen}(1^\lambda)$:

  For $b \in \{0,1\}$, run $\mathsf{msk}_b \leftarrow \mathsf{CPRF.KeyGen}(1^\lambda)$. Then sample a secret key $K \leftarrow\$ F.\mathsf{KeyGen}(1^\lambda)$ for the PRF F. Compute $\mathsf{ck}_0 \leftarrow \mathsf{CPRF.Constrain}(\mathsf{msk}_0, f_K)$ and $\mathsf{ck}_1 \leftarrow \mathsf{CPRF.Constrain}(\mathsf{msk}_1, 1 - f_K)$. Output $k_0 \leftarrow (\mathsf{msk}_0, \mathsf{msk}_1)$ and $k_1 \leftarrow (K, \mathsf{ck}_0, \mathsf{ck}_1)$.

- $\mathsf{PCF.Eval}(\sigma, k_\sigma, x)$:
  - If $\sigma = 0$, parse $k_0 = (\mathsf{msk}_0, \mathsf{msk}_1)$, and compute $y_b \leftarrow \mathsf{CPRF.Eval}(\mathsf{msk}_0, x)$ for $b \in \{0,1\}$, and output (y_0, y_1).
  - If $\sigma = 1$, parse $k_1 = (K, \mathsf{ck}_0, \mathsf{ck}_1)$, and compute $b \leftarrow F_K(x)$. Set $y_b \leftarrow \mathsf{CPRF.CEval}(\mathsf{ck}_b, x)$, and output (b, y_b).

We further observe that the resulting PCF is *precomputable* as recently defined in [CMP+23]. Informally, it allows one of the parties to locally generate its own PCF key and compute its correlated randomness entirely, before even knowing the identity of the other party. In the above construction, the sender can precompute all pairs (y_0, y_1) ahead of time, and it is therefore precomputable.

We note that the fact that CPRFs for a class containing a PRF yield a PCF is not entirely new; for example, a similar observation was briefly mentioned in [BGM+20]. However, the few known constructions of sufficiently expressive CPRFs [BV15; AMN+18; CMP+23] are too expensive, and using them within the above transformation yields PCFs that are much less flexible than generic constructions based on homomorphic secret sharing or threshold FHE (that are not restricted to the OT correlation), and much less efficient than state-of-the-art PCFs [BCG+20a; BCG+22]. Our key contribution is identifying that a simple tweak to the Naor-Reingold PRF [NR97] yields an extremely efficient pseudorandomly constrained PRF.

A CPRF for Inner-Product Membership from the Naor-Reingold PRF

Let us first recall the Naor-Reingold PRF [NR97], whose input domain is $\mathcal{X} = \{0,1\}^n$. Let $\mathbb{G} = \mathbb{G}(\lambda)$ be a family of cyclic groups of prime order $p = p(\lambda)$.

- $F.\mathsf{KeyGen}(1^\lambda)$: Sample $g \leftarrow\$ \mathbb{G}$ and $a_1, a_2, \cdots, a_n \leftarrow\$ \mathbb{Z}_p^*$. Output $\mathsf{msk} \leftarrow (g, a_1, \cdots, a_n)$.
- $F.\mathsf{Eval}(\mathsf{msk}, x)$: On input $x = (x_1, \cdots, x_n) \in \{0,1\}^n$, output $g^{\prod_{i=1}^n a_i^{x_i}}$.

Evaluating the Naor-Reingold PRF requires a few multiplications, followed by a single exponentiation. Its security reduces to the Decisional Diffie-Hellman assumption over $\mathbb{G}$.

A no-evaluation-secure CPRF for inner-product. As a warm-up, we define the class of predicates

$$C_z : x \to \begin{cases} 0 & \text{if } \langle x, z\rangle = 0 \\ 1 & \text{otherwise.} \end{cases}$$

That is, a constrained key for z allows evaluating the PRF on all inputs x where $\langle x, z\rangle = 0$. Now, consider the following extension of the Naor-Reingold PRF:

- $F.\mathsf{Constrain}(\mathsf{msk}, z)$: Sample $r \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and define $(\alpha_1, \cdots, \alpha_n) \leftarrow (r^{z_1} \cdot a_1, \cdots, r^{z_n} \cdot a_n)$. Output $\mathsf{ck} = (g, \alpha_1, \cdots, \alpha_n)$.
- $F.\mathsf{CEval}(\mathsf{ck}, x)$: On input $x = (x_1, \cdots, x_n) \in \{0,1\}^n$, output $g^{\prod_{i=1}^n \alpha_i^{x_i}}$.

Here, each key a_i is *blinded* by a term r^{z_i}, and the outputs of the Eval and CEval algorithms coincide when the blinding terms cancel out which happens precisely when the inner product $\langle x, z\rangle$ is equal to 0 modulo the order of r. For a safe prime p with $p - 1 = 2q$, the order of r is q or $2q$ with overwhelming probability, so with $q \gg n$, $\langle x, z\rangle = 0 \bmod q$ iff $\langle x, z\rangle = 0$ over the integers. More precisely, we have:

$$\begin{aligned} F.\mathsf{CEval}(\mathsf{ck}, x) = g^{\prod_{i=1}^n \alpha_i^{x_i}} &= g^{\prod_{i=1}^n (r^{z_i} \cdot a_i)^{x_i}} = g^{r^{\sum_{i=1}^n x_i z_i} \prod_{i=1}^n a_i^{x_i}} \\ &= (g^{\prod_{i=1}^n a_i^{x_i}})^{r^{\langle x,z\rangle}} = (F.\mathsf{Eval}(\mathsf{msk}, x))^{r^{\langle x,z\rangle}} \\ &= F.\mathsf{Eval}(\mathsf{msk}, x) \text{ iff } \langle x, z\rangle = 0 \ . \end{aligned}$$

Furthermore, when the adversary makes no query to the evaluation oracle, it can be shown that the pseudorandomness of the above construction on a challenge input x where $\langle x, z\rangle \neq 0$ holds as long as $g^{r^{\langle x,z\rangle}}$ looks random for a uniformly random $r \in \mathbb{Z}_p^*$. Indeed, the constrained key owner can compute $r^{\langle x,z\rangle} \prod_{i=1}^n a_i^{x_i}$ and knows g, x, z. The actual evaluation is $g^{\prod_{i=1}^n a_i^{x_i}}$ and the constrained key reveals no information about r since a_i are uniformly random in $\mathbb{Z}_p^*$.

Before we move on, we make three observations:

- The algorithm $F.\mathsf{CEval}$ does not need to know z. Hence, our CPRF for inner products is also constraint-hiding.
- We described the construction for an input and a constrain $x, z \in \{0,1\}^n$ for simplicity, and to match with the original construction of Naor and Reingold. However, the construction extends immediately to the setting where $x, z \in [\pm B]^n$, where B is some polynomial-size bound (the security of the original Naor-Reingold construction for inputs of this form was shown in [ABP15] to reduce to a variant of the Diffie-Hellman assumption). We then have $|\langle x, z\rangle| \leq n \cdot B^2$ and assuming $n \cdot B^2 \ll q$, the inner product is computed over the integers.
- Eventually, we also consider a straightforward modification of the construction where both parties apply an arbitrary public preprocessing function $\mathsf{p}(\cdot)$ on the input x before feeding it into Eval or CEval. This allows to force the input x to have a specific format.

From no-evaluation security to full security. While the above construction can be attacked if the adversary makes an evaluation query, we recall that any no-evaluation secure CPRF can be turned into an adaptively secure CPRF (with any number of evaluation queries) in the random oracle model by hashing the output, as in [AMN+18]. Hence, proving no-evaluation security suffices for our goal.

From inner product to inner product membership. We just turned the Naor-Reingold PRF into a CPRF, but it is restricted so far to a small class of functions (inner product predicates) which is of course too limited to instantiate our template PCF construction from Section 4.2.4. Our next observation is that this class can be significantly expanded by adding elements of the form $g^{r^{-t}}$ to the constrained key to help the evaluator cancel out some r^t terms. Indeed, if the evaluator uses $g^{r^{-t}}$ instead of g as the basis for exponentiation, the computation of $F.\mathsf{CEval}(\mathsf{ck}, x)$ becomes

$$\begin{aligned} F.\mathsf{CEval}(\mathsf{ck}, x) &= (g^{r^{-t}})^{r^{\langle x,z\rangle}\cdot\prod_{i=1}^n (r^{z_i}\cdot a_i)^{x_i}} \\ &= (g^{r^{\langle x,z\rangle - t}})^{\prod_{i=1}^n (r^{z_i}\cdot a_i)^{x_i}} \\ &= (F.\mathsf{Eval}(\mathsf{msk}, x))^{r^{\langle x,z\rangle - t}} \quad , \end{aligned}$$

which is the same as $F.\mathsf{Eval}(\mathsf{msk}, x)$ iff $t = \langle x, z\rangle$. What makes this observation particularly powerful is that the evaluator can be given terms of the form $g^{r^{-t}}$ *for multiple values of* t, and choose upon evaluation the term $g^{r^{-t}}$ for $t = \langle x, z\rangle$. This yields a CPRF for the class of predicates

$$C_{z,S} : x \to \begin{cases} 0 & \text{if } \langle x, z\rangle \in S \\ 1 & \text{otherwise} \end{cases}$$

where $S \subseteq \mathbb{Z}_{p-1}$ is a polynomial size subset. The full construction is given below:

- $F.\mathsf{Constrain}(\mathsf{msk}, z, S)$: sample $r \leftarrow\$ \mathbb{Z}_p^*$. For every $t \in S$, define $g_t \leftarrow g^{r^{-t}}$ and define $(\alpha_1, \cdots, \alpha_n) \leftarrow (r^{z_1} \cdot a_1, \cdots, r^{z_n} \cdot a_n)$. Output $\mathsf{ck} = ((g_t)_{t\in S}, \alpha_1, \cdots, \alpha_n)$.
- $F.\mathsf{CEval}(\mathsf{ck}, x)$: on input $x = (x_1, \cdots, x_n) \in [\pm B]^n$, set $t \leftarrow \langle x, z\rangle$ and output $g_t^{\prod_{i=1}^n \alpha_i^{x_i}}$.

Note that our prior claim that the constrained key contains no information about r does not longer hold as it now contains $g^{r^{-t}}$ for all $t \in S$, hence no-evaluation security is no longer unconditional. We show that this construction is (no-evaluation) secure under a variant of the Diffie-Hellman assumption which we call *sparse power-DDH* assumption and which states that given $g^{r^{-t}}$ for various $t \in S$, it is infeasible to distinguish $g^{r^{-t}}$ for $t \notin S$ from uniformly random group elements.[1] The sparse power-DDH assumption is a static falsifiable assumption. It generalizes in a natural way the power-DDH assumption (which states that given g^{r^i} for $i = 1$ to n, it is infeasible to distinguish $g^{r^{n+1}}$ from random), and is easily proven to hold in the generic group model since all exponents are distinct univariate monomials (e.g., using [BBG05, Corollary A.3] , and observing that it is a special case of the uber-assumption family).

Inner-Product Membership Weak Pseudorandom Functions

We observe that several known candidate weak PRFs can be expressed as IPM predicates. In this section, we provide two candidates for IPM predicates that are BIPSW [BIP+18] and the XOR-MAJ [Gol00; AL16]. We write IPM - wPRF to denote a weak PRF expressed as an IPM predicate.

The BIPSW wPRF. In [BIP+18], the authors introduced several new low-complexity wPRF candidates, together with some preliminary analysis to back up the security claims. Five years later, these candidates have received some attention, both by cryptanalysts [CCK+21; JMN23] and in the context of a range of applications, from secure computation to side-channel security [DGH+21; ADD+23; DMM+21]. As the authors observed, one of their candidates (that we denote as BIPSW) can be rephrased as an LWR-style wPRF: $F_z(x) = \lfloor \langle x, z\rangle \bmod 6 \rceil$, with $x, z \in \{0,1\}^n$, and with the

[1] By $t \notin S$, we mean $t \notin S$ but still $t \in \mathcal{R}$, for some small support $\mathcal{R}$ denoting the (polynomial-sized) range of possible values for the inner-product $\langle z, x\rangle$, e.g., $R = \{-n \cdot B^2, \ldots, n \cdot B^2\}$ for $x, z \in [\pm B]^n$

rounding function defined as $\lfloor s \rceil = 0$ if $(s \bmod 6) \in \{0,1,2\}$, and $\lfloor s \rceil = 1$ if $(s \bmod 6) \in \{3,4,5\}$. The authors initially suggested a key length $n = 384$ as a conservative choice for security. Several attacks were later shown, in [CCK+21] and very recently in [JMN23], suggesting that the key length should be increased to $n = 770$. We note that the BIPSW candidate fits particularly well in our framework: it can be written as an IPM - wPRF by defining $S = \{s \leq n \ : \ \exists k \leq n/6, i \in \{0,1,2\}, s = 6k+i\}$. The size of S is $n/2$, which is as low as $|S| = 385$ for $n = 770$.

The XOR-MAJ wPRF. The previous construction works for arbitrary predicates P, provided that P takes at most $O(\log n)$ bits as input. In this section, we observe that when P is of the form $P(x_0, x_1) = \mathsf{SYM}_0(x_0) \oplus \mathsf{SYM}_1(x_1)$, where $\mathsf{SYM}_0, \mathsf{SYM}_1$ are arbitrary symmetric functions, then there exists an improved construction that handles predicates of *arbitrary* locality. This capture in particular the XOR - MAJ predicate, which computes the XOR between the parity of the x_0 input and the majority of the x_1 input. XOR - MAJ is probably the most common choice of predicate for the GAR wPRF, and its properties have been studied extensively [AL16; CDM+18; Méa; YGJ+21; Méa22; Üna23b].

We briefly outline how to express the GAR wPRF with the XOR - MAJ predicate as an IPM - wPRF (the generalization to other symmetric functions is immediate). Assume that the predicate is $P = \mathsf{XOR}_\mathsf{k}$ - MAJ_ℓ, which takes as input a $(k+\ell)$-bit subset z of the bits of the secret key, and outputs $\mathsf{XOR}(z_1, \cdots, z_k) \oplus \mathsf{MAJ}(z_{k+1}, \cdots, z_{k+\ell})$. Similarly to before, we parse a random input x as an encoding of two random disjoint subsets $(S_{0,x}, S_{1,x})$ of $[n]$, of size k and ℓ respectively. Then, we let $\mathsf{p}(x)$ denote the length-n vector with 1's at all entries indexed by $S_{1,x}$, value $\ell+1$ at all entries indexed by $S_{0,x}$, and 0's everywhere else. Observe that this encodings yields

$$\langle \mathsf{p}(x), z \rangle = \mathsf{HW}((z_i)_{i \in S_{1,x}}) + (\ell+1) \cdot \mathsf{HW}((z_i)_{i \in S_{0,x}}),$$

where $\mathsf{HW}(\cdot)$ denotes the Hamming weight. Furthermore, since $|S_{1,x}| = \ell$, every integer $\langle \mathsf{p}(x), z \rangle$ computed as above uniquely determines the pair $(\mathsf{HW}((z_i)_{i \in S_{1,x}}), \mathsf{HW}((z_i)_{i \in S_{0,x}})))$. In turn, symmetric functions such as XOR and MAJ are uniquely determined by the Hamming weight of their inputs (in particular, $\mathsf{XOR}(z)$ returns $\mathsf{HW}(z) \bmod 2$ and $\mathsf{MAJ}(z)$ returns 1 iff $\mathsf{HW}(z) > \ell/2$). Then, using the fact that $A \oplus B = 0$ iff $A = B$, we define S as follows:

$$S = \{s = s_1 + (\ell+1)s_0 \in [\ell + (\ell+1) \cdot k] \ : \ [\mathsf{HW}(s_1) \bmod 2] = [\mathsf{HW}(s_0) > \ell/2]\}.$$

Compared to the previous construction, this new construction is tailored to XOR - MAJ (or more generally to predicates of the form $P(x_0, x_1) = \mathsf{SYM}_0(x_0) \oplus \mathsf{SYM}_1(x_1)$[2].). However, for a predicate of locality $\ell + k$, the size of S scales as $O(\ell \cdot k)$, which is an exponential improvement over the $2^{\ell+k}$ cost of the generic construction. While the GAR wPRF is typically considered in the low-locality setting, our construction allows simultaneously relying on a particularly conservative parameter setting, using XOR - MAJ with locality $O(\sqrt{n})$ (in this parameter regime, the GAR wPRF is generally conjectured to provide subexponential security $2^{O(\sqrt{n})}$), and keeping S to a small size $|S| = O(n)$. Together with the BIPSW wPRF candidate, this instantiation yields the most efficient concrete instantiations of our framework. To give a single data point, using the state-of-the-art cryptanalysis on Goldreich-style local wPRFs, we can set the key length n to 256 and use the XOR_{10} - MAJ_{64} predicate to achieve 128 bits of security for up to 2^{40} queries to the wPRF, and have $|S| = 357$.

Optimizations

The above framework can be largely improved by various optimizations. In this section, we sketch several of them that allow improving the performance of our PCF. We provide a detailed description

[2]even more generally, the construction can be adapted to handle the XOR of any number N of symmetric predicates with respective locality $\ell_1, \cdots, \ell_N$, with $|S| = O(\prod_{i=1}^{N} \ell_i)$

of the optimizations in the full version.

Halving the key size. When we instantiate the framework of Section 4.2.4 using our Naor-Reingold CPRF, the sender key consists of two master secret keys $(g, a_1, \cdots, a_n)$ and $(h, a'_1, \cdots, a'_n)$, and the receiver key for the predicate $F_{z,S} : x \mapsto \langle x, z\rangle \in_? S$ consists of $(r^{z_i} \cdot a_i)_{i\in[n]}, ((r')^{z_i} \cdot a'_i)_{i\in[n]}, (g^{r^{-t}})_{t\in S}$, and $(h^{(r')^{-t}})_{t\in[\pm n\cdot B^2]\setminus S}$ (where r, r' are random elements of $\mathbb{Z}_p^*$ and B is a bound on the entries of x, z). Thanks to the random self-reducibility of DDH, the two master secret keys can use the same elements $(a_1, \cdots, a_n)$ provided that they use different bases g, h. For the same reason, we can also set $r = r'$ without any security loss. This reduces the sender key size by a factor two, and significantly compresses the receiver key size as well. Concretely, we have:

- Sender key: $(g, h, a_1, \cdots, a_n)$,
- Receiver key: $(r^{z_i} \cdot a_i)_{i\le n}, (g_t)_{t\in[\pm n\cdot B^2]}$,

where $g_t \leftarrow g^{r^{-t}}$ if $t \in S$, and $g_t \leftarrow h^{r^{-t}}$ if $t \in [\pm n \cdot B^2] \setminus S$. The resulting construction is secure under the same assumptions as the basic construction.

Reusing the g_t's. We observe that the value z (which relates to underlying PRF key used by the receiver) is known only to the receiver, while the set S is public (and relates to the definition of the PRF). In a multiparty setting where the sender wants to compute PCF keys with multiple receivers, we can exploit this observation to define the g_t's once for all, and pass them as common parameters to be used by all receivers. This requires adding two additional terms $(a_{0,j}, a'_{0,j})$ in the sender key for each receiver R_j, to re-randomize the bases g, h. That is, the sender now computes its pseudorandom OT messages as

$$s^x_{0,j} \leftarrow g^{a_{0,j}\cdot\prod_{i=1}^n a_i^{x_i}} \qquad\qquad s^x_{1,j} \leftarrow h^{a'_{0,j}\cdot\prod_{i=1}^n (a'_i)^{x_i}},$$

where $g^{a_{0,j}}$ and $h^{a'_{0,j}}$ play the role of fresh new bases for each receiver R_j (the receiver CEval has to be adapted accordingly). With this change, the g_t's can be viewed as public parameters (or as a "public key" associated to the sender).

Compressing the a_i's. When instantiating the group with a suitable elliptic curve, the size of the a_i's is typically 2λ bits (to achieve λ bits of security against generic discrete log attacks). To further reduce the key size, the a_i's can be generated from a pseudorandom generator in a two-step fashion: first, the sender receives a λ-bit seed seed and computes $(g, h, \mathsf{seed}_1, \cdots, \mathsf{seed}_n) \leftarrow \mathsf{PRG}(\mathsf{seed})$, where $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^{(4+n)\cdot\lambda}$ (each seed_i is in $\{0,1\}^\lambda$). Second, define $a_i \leftarrow \mathsf{PRG}'(\mathsf{seed}_i)$, where $\mathsf{PRG}' : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda}$. This approach enables compressing both the sender key and the receiver key:

- The sender key is now simply the λ-bit seed seed.
- The receiver key is still $(r^{z_i} \cdot a_i)_{i\le n}$ (together with the public g_t terms), except that whenever $z_i = 0$, we have $r^{z_i} \cdot a_i = a_i$, which we can send in compressed form by replacing it with seed_i, which is twice smaller. When z_i is a bitstring (which is the case for the BIPSW and XOR-MAJ wPRFs), this reduces the size of about half of the $r^{z_i} \cdot a_i$ to that of seed_i, resulting in a 25% reduction of the key length.

Exploiting the structure in S. Assume that S contains all integers s (from some bounded range $\{0, \cdots, m \cdot R\}$) such that $(s \bmod m) < m/2$, where m is some fixed value; we say that S is m-antiperiodic. Then we *almost* have:

$$s \notin S \iff (s - m) \in S \ ,$$

where the *almost* stems from the fact that the equivalence breaks down at the extremities: for example, $s = m/2 + 1 \notin S$, yet $s - m \notin S$ because $s - m$ is outside of the bounded range $\{0, \cdots, m \cdot R\}$. Nevertheless, we can recover the equivalence by slightly extending S into $S' = S \cup \{-m/2, \cdots, -1\}$ (the equivalence becomes: for every $s \in \{0, \cdots, R \cdot m\}$, $s \notin S' \iff (s - m) \in S'$).

In this case, we observe that it is not necessary to include in the receiver key both $(g_t)_{t \in S}$ and $(g_t)_{t \notin S}$. Indeed, as we are constraining before a key msk_0 with respect to the predicate "$\langle x, z \rangle \in S$", and a second key msk_1 with respect to the predicate "$\langle x, z \rangle \notin S$", we can then rewrite the second constraint as "$\langle x, z \rangle - m \in S$". Concretely, we now deal a single key msk to the sender, but we add an $(n + 1)$-th element a_{n+1} to act as a *shift*. The keys become:

- Sender key: $\mathsf{msk} = (g, a_1, \cdots, a_n, a_{n+1})$
- Receiver key: $\mathsf{ck} = (g, r^{z_1} \cdot a_1, \cdots, r^{z_n} \cdot a_n, r^{-m} \cdot a_{n+1}), (g_t)_{t \in S'}$.

Given msk, on input x the sender computes their two OT inputs as $s_b \leftarrow F.\mathsf{Eval}(\mathsf{msk}, x|b)$ for $b = 0, 1$. That is, we have:

$$s_b \leftarrow g^{\prod_{i=1}^{n} a_i^{x_i} \cdot a_{n+1}^{b}} \text{ for } b = 0, 1.$$

Now, because of the term $r^{-m} \cdot a_{n+1}$ in the key of the receiver, for every string x, there is only a single $b \in \{0, 1\}$ such that $\langle x|b, z| - m \rangle \in S$, i.e. such that $\langle x, z \rangle - b \cdot m \in S'$. Compared to the previous construction, this (almost) halves the number of group elements g_t in the receiver key, going from $m \cdot R$ to $(m/2) \cdot (R + 1)$.

The BIPSW wPRF and the XOR-MAJ wPRF satisfy this property: the set S is m-antiperiodic with $m = 6$ for BIPSW, and $m = 49$ for our parameter choice with XOR-MAJ. Hence, they can benefit from this optimization.

Final PCF Construction

We are now fully equipped to describe our final PCF construction. Concrete parameters for both instantiations based on the BIPSW and the XOR-MAJ are provided in the next section. Let the input domain be $\{0, 1\}^\ell$. Let F be an IPM - wPRF with preprocessing function $\mathsf{p} : \{0, 1\}^\ell \mapsto [0, B]^n$ (for some polynomial bound B), key space $\{0, 1\}^n$ and associated set S; that is, given a key $z \leftarrow\!\!\$ \{0, 1\}^n$ and an input $x \in \{0, 1\}^\ell$, $F_z(x)$ outputs 1 iff $\langle \mathsf{p}(x), z \rangle \in S$. We assume that S is m-antiperiodic for some integer m (*i.e.* $S = \{s \in \{0, \cdots, R\} : s \bmod m < m/2\}$ for some polynomial bound R). Define $S' \leftarrow S \cup \{-m/2, \cdots, -1\}$. Fix a family of cyclic groups $\mathbb{G} = \mathbb{G}(\lambda)$ of order $p = p(\lambda)$. Let $G_0 : \{0, 1\}^\lambda \mapsto \mathbb{Z}_p^* \times \{0, 1\}^{n \cdot \lambda}$, $G_1 : \{0, 1\}^\lambda \mapsto \mathbb{Z}_p^*$, and $G_2 : \{0, 1\}^\lambda \mapsto \{0, 1\}^n$ be three pseudorandom generators. Let $H : \mathbb{G} \mapsto \{0, 1\}^\lambda$ be a hash function.

- $\mathsf{PCF.Gen}(1^\lambda)$: sample $\mathsf{seed} \leftarrow\!\!\$ \{0, 1\}^\lambda$. let $(g, \mathsf{seed}_1, \cdots, \mathsf{seed}_{n+1}) \leftarrow G_0(\mathsf{seed})$ and $a_i \leftarrow \mathbb{G}_1(\mathsf{seed}_i)$ for $i = 1$ to $n + 1$. Sample $\mathsf{seed}_z \in \{0, 1\}^\lambda$, $r \leftarrow\!\!\$ \mathbb{Z}_p^*$, and let $z \leftarrow G_2(\mathsf{seed}_z)$. For $i = 1$ to n, set $v_i \leftarrow \mathsf{seed}_i$ if $z_i = 0$, and $v_i \leftarrow r \cdot a_i$ otherwise. Set $v_{n+1} \leftarrow\!\!\$ r^{-m} \cdot a_{n+1}$. Define $g_t \leftarrow g^{r^{-t}}$ for every $t \in S'$. Output $k_0 \leftarrow \mathsf{seed}$ and $k_1 \leftarrow (\mathsf{seed}_z, v_1, \cdots, v_{n+1}, (g_t)_{t \in S'})$.

- $\mathsf{PCF.Eval}(0, k_0, x)$: recompute $(g, \mathsf{seed}_1, \cdots, \mathsf{seed}_{n+1}) \leftarrow G_0(k_0)$ and $a_i \leftarrow \mathbb{G}_1(\mathsf{seed}_i)$ for $i = 1$ to $n + 1$. Let $(y_1, \cdots, y_n) \leftarrow \mathsf{p}(x)$. Define

$$s_b \leftarrow H\left(g^{\prod_{i=1}^{n} a_i^{y_i} \cdot a_{n+1}^{b}}\right) \text{ for } b = 0, 1.$$

  Output (s_0, s_1).

- $\mathsf{PCF.Eval}(1, k_1, x)$: parse k_1 as $(\mathsf{seed}_z, v_1, \cdots, v_{n+1}, (g_t)_{t \in S'})$. Let $(y_1, \cdots, y_n) \leftarrow \mathsf{p}(x)$. Recompute $z \leftarrow G_2(\mathsf{seed}_z)$. For $i = 1$ to $n + 1$, set $\alpha_i \leftarrow G_1(v_i)$ if $z_i = 0$ or $i = n + 1$, and

$\alpha_i \leftarrow v_i$ else. Let $b \leftarrow F_z(x)$ and $t \leftarrow \langle \mathsf{p}(x), z\rangle - b \cdot m$. Note that by definition, this means that $t \in S'$. Define

$$s_b \leftarrow H\left(g_t^{\prod_{i=1}^n \alpha_i^{y_i} \cdot \alpha_{n+1}^b}\right).$$

Output (b, s_b).

To state our main theorem, we define the sparse power-DDH assumption with respect to S'. For a group $\mathbb{G}_\lambda = \langle g\rangle$ of prime order p, the sparse power-DDH assumption with respect to S' over a support $[0, R]$ states that

$$\left(g, (g^{r\cdot a^i})_{i\in S'}, (g^{r\cdot a^i})_{i\in[0,R]\setminus S'}\right) \approx \left(g, (g^{r\cdot a^i})_{i\in S'}, (g^{t_i})_{i\in[0,R]\setminus S'}\right),$$

where $\approx$ denotes computational indistinguishability, $a, r \overset{\$}{\leftarrow} \mathbb{Z}_p^*$, and $t_i \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ for all $i \in [0, R] \setminus S'$. Note that the bound R and the set S' are fixed parameters of the construction; hence, this assumption is a static, falsifiable variant of the power-DDH assumption used in several previous works (e.g. [AMN+18]). It can be shown to hold in the generic group model. We obtain the following theorem:

Theorem 4.2.5 (informal). *Assuming that the sparse power-DDH assumption with respect to S' holds, that (G_0, G_1, G_2) are pseudorandom generators, that F is a secure* IPM - wPRF, *and modeling H as a random oracle, then the above construction is a weak pseudorandom correlation function for the oblivious transfer correlation.*

Note that, in the random oracle model, the construction can be upgraded to a strong PCF by first hashing the inputs [BCG+20a] and can also be proven secure under a weaker *search* version of the sparse power-DH assumption.

Distributed key generation. A useful feature of our PCF is that it admits a very efficient two-round distributed key generation algorithm. Concretely, and borrowing the notations from the construction above, the OT sender can simply generate seed and r themself, and send $(g_t)_{t\in S'}$ to the OT receiver directly, together with $v_{n+1} = r^{-m} \cdot a_i$. Then, the OT receiver samples seed_z. Eventually, to obtain the missing v_i's, observe that $v_i = \mathsf{seed}_i$ if $z_i = 0$, and $v_i = r \cdot a_i$ otherwise. Therefore, the sender and the receiver simply run n parallel instances of an oblivious transfer protocol, where the sender input pairs $(\mathsf{seed}_i, r \cdot a_i)$, and the receiver uses selection bits z_i. Security follows immediately from the security of the oblivious transfer protocol. Using a two-round OT protocol, the entire distributed key generation can be done in two rounds, and the communication boils down to n parallel OTs plus sending $|S'|$ group elements.

4.2.5 Concrete Instantiation of DV-NIZK

Concrete Parameters for PK-PCF based OT

With all the above optimizations in mind, we provide two concrete instantiations of PCF for the OT correlation, using either the BIPSW wPRF candidate, or the XOR-MAJ wPRF candidate.

Curve and exponentiations. To estimate the runtime of our constructions, we rely mainly on the website zka.lc, which provides an extensive list of benchmarks for standard operations on various curves and over various platforms. According to the benchmarks of zka.lc, computing one exponentiation represents about 50μs of computation on one core of an AWS platform using curve25519 [Ber06]. Note that in our construction, the sender must compute two exponentiations (to compute (y_0, y_1)) while the receiver computes a single exponentiation. However, the two sender exponentiations use a fixed basis g. Hence, the exponentiations can be significantly sped up with precomputation (in contrast, the receiver does an exponentiation with a basis g_t which is chosen based on the input). In our instantiations, exponentiations will generally dominate the runtime. Using more efficient curves, such as Microsoft's Four$\mathbb{Q}$ curve [CL15], the exponentiation time can be reduced to about 15μs on a Haswell architecture (note that the curve offers slightly less security compared to curve25519, about 122 bits instead of 128).

Parameters with BIPSW. For BIPSW, we used the state-of-the-art cryptanalysis from the works of [CCK+21; JMN23], and set the key length to $n = 770$, which achieves 128 bits of security according to these attacks. We note that this parameter choice ignores some significant polynomial factors in the cost estimation (that come from a nearest neighbor search), hence our parameter choice takes a bit of margin. Furthermore, the recent attack of [JMN23] has a much higher memory requirement compared to previous attack. On the other hand, we warn the reader that the BIPSW candidate is a relatively young wPRF and while a total break would be surprising at this point, the state of cryptanalysis is likely to improve over the years. With $n = 770$, S is 6-antiperiodic and we have $|S'| = 388$ With this parameter choice, the precomputable PCF has the following efficiency features:

- Key size: the receiver key size is 30.2kB (and the sender key size is 16 Bytes). Out of that, 12.1kB are public parameters $(g_t)_{t \in S'}$, which the sender can reuse with other receivers.
- Computation: computing s_b involves 385 multiplications over $\mathbb{Z}_p^*$, one exponentiation, and one hash. This translates to about 10k OT/s per core using curve25519 on an AWS platform, or about 15k OT/s per core using a curve such as Four$\mathbb{Q}$ on a Haswell architecture.

Parameters with XOR-MAJ. For the GAR wPRF instantiated with the XOR-MAJ predicate, we rely on the state-of-the-art cryptanalysis results from [AL16; CDM+18; YGJ+21; Üna23a]. Specifically, according to Table 1 of [Üna23a], for a candidate to achieve λ bits of security with a key of length $n = \lambda^\delta$ and a bound n^{1+e} on the number of queries, the underlying predicate P must have

- rational degree at least $\frac{\delta}{\delta-1} \cdot e + 1$, and
- resiliency at least $2e + 1$.

A k-variable Boolean function P has rational degree d if it is the smallest integer for which there exist degree d polynomials g and h, not both zero, such that $P \cdot g = h$. [3] A k-variate boolean function is t-resilient if it has no nontrivial correlation with any linear combination of at most t of its inputs. We note that Table 1 of [Üna23a] ignores the guess-and-decode attack of [YGJ+21], because their attack does not have a closed-form formula. However, Both the guess-and-determine attack of [CDM+18] and the guess-and-decode attack of [YGJ+21] are specifically targeted at predicates with a very small locality (the papers consider localities from 5 to 8), and their complexity scales very poorly for predicates with a larger locality. As we will see shortly, our candidates have considerably higher locality (e.g. 74 in our main instantiation) and after selecting them, we verified individually that they yield concrete instances which are (way) out of reach of the guess-and-determine and the guess-and-decode attacks. In the following, we therefore use the two criteria above to select our candidates.

The algebraic immunity and resiliency of the XOR-MAJ predicate have been studied in several papers. To match the above two constraints, it suffices to use the XOR_{ℓ_1} - MAJ_{ℓ_2} predicate with $\ell_1 = 2 \cdot (e + 1)$ and $\ell_2 = 2\delta e/(\delta - 1)$. We outline below a concrete choice of parameters for

[3]Table 1 of [Üna23a] mentions only the degree of the predicate, but strengthening the requirement to the rational degree is known to be necessary [AL16; DMR23].

illustration: set $\delta = 1.143$. This yields $\delta/(\delta - 1) = 8$ and $n = \lambda^\delta = 256$ using $\lambda = 128$. We get $\ell_2 = 16e$, and $|S'| = 16e^2 + 33e + 2$. Setting $e = 4$, the parties can generate up to $n^{1+e} = 2^{40}$ pseudorandom OTs and $|S'| = 390$. With this parameter choice, the precomputable PCF has the following efficiency features:

- Key size: the receiver key size is 18kB (and the sender key size is 16 Bytes). Out of that, 12.2kB are public parameters $(g_t)_{t\in S'}$, which the sender can reuse with other receivers.
- Computation: computing s_b involves 74 multiplications over $\mathbb{Z}_p^*$, one exponentiation, and one hash. This translates to about 15k OT/s per core using curve25519 on an AWS platform, or about 40k OT/s per core using a curve such as Four$\mathbb{Q}$ on a Haswell architecture.

Note that other choices of parameters can yield different trade-offs, such as achieving slightly smaller key size, slightly more OTs, or slightly less computation. For example, using $\delta = 1.2858$ yields $n = 512$, $|S'| = 222$, a slightly larger key size 18.9kB, 46 multiplications instead of 74, and a bound of 2^{45} on the target number of OTs.

Public Key PCF

We finally describe a *public key PCF*. Informally, a public key PCF allows users to generate a pair of public/secret keys, and then to broadcast their public key using a single message, while storing their secret key locally. Then, any pair of users can *non-interactively* obtain a PCF key pair (k_0, k_1) by combining their secret key and the other party's public key.

While the distributed key generation protocol described in Section 4.2.4 is particularly efficient, it requires two rounds of interaction. The protocol we now describe uses a *single* round of interaction. A major advantage of such protocol is that they enable n parties over a network to execute $\Omega(n^2)$ pairwise PCF key generations (to set up an OT channel between each pair of parties) using only $O(n)$ communication in total (this is similar to how non-interactive key exchange enable n^2 pairs of parties to agree on shared keys using $O(n)$ communication).

A simple construction. In our interactive protocol, sending $(g_t)_{t\in S'}$ does not require interaction: the interaction stems entirely from the OTs. We start with a protocol that replaces the two-round OT with the non-interactive OT protocol of Bellare and Micali [BM90]. The objective is, for the sender with input r, and the receiver with input z_i, to distributively generate keys $a_i \in \mathbb{Z}_p^*$ and $\alpha_i = r^{-z_i} \cdot a_i \in \mathbb{Z}_p^*$ respectively. These values can be viewed as multiplicative shares over $\mathbb{Z}_p^*$ of r^{-z_i} (up to inverting a_i locally). We observe that if DDH holds over (a suitable subgroup of) $\mathbb{Z}_p^*$, such multiplicative shares can be directly obtained via the Bellare-Micali protocol. Concretely, let $\mathbb{G}'$ be a suitable cyclic subgroup of $\mathbb{Z}_p^*$ where DDH is conjectured to hold, and let (G, H) be two random generators of $\mathbb{G}'$, and let $r \in \mathbb{G}'$. The protocol simply consists in having the sender send an ElGamal encryption of r, while the receiver sends a Pedersen commitment to z_i:

- **Sender to receiver:** pick a random coin ρ, and sends the ElGamal ciphertext $(C_0, C_1) \leftarrow (G^\rho, H^\rho \cdot r)$.
- **Receiver to sender:** pick n random coins $(\theta_1, \cdots, \theta_n)$ and send the Pedersen commitments $(H_1, \cdots, H_n) \leftarrow (H^{-z_i} \cdot G^{\theta_i})_{i\le n}$.
- **Output:** for $i = 1$ to n, the sender outputs $a_i \leftarrow H_i^\rho$, and the receiver outputs $\alpha_i \leftarrow C_1^{-z_i} \cdot C_0^{\theta_i}$.

Observe that

$$\alpha_i = C_1^{-z_i} \cdot C_0^{\theta_i} = G^{\rho\theta_i} \cdot H^{-\rho z_i} r^{-z_i} = a_i \cdot r^{-z_i}.$$

Furthermore, r is computationally indistinguishable from a random element of $\mathbb{G}'$ under the DDH assumption over $\mathbb{G}'$, and the protocol statistically hides z_i.

A first downside of this protocol is that we cannot set $\mathbb{Z}_p^* = \mathbb{G}'$, since DDH is easy over $\mathbb{Z}_p^*$ (it can be broken by computing the Legendre symbol). However, assuming that $p = 2q + 1$ is a safe prime (q is prime), we can set $\mathbb{G}'$ to be the subgroup QR_p of quadratic residues modulo p, where DDH is widely conjectured to hold (for a sufficiently large p). This implies that the protocol generates a (pseudo)random *square* r, instead of a random element of $\mathbb{Z}_p^*$. This does not harm the security of the CPRF but changes slightly the underlying sparse power-DDH variant: using r of the form w^2 for a (pseudo)random element w when computing $g_t \leftarrow g^{r^t} = g^{w^{2t}}$ for $t \in S'$ amounts exactly to relying on the sparse power-DDH assumption with respect to the set $2 \cdot S' = \{2 \cdot t \ : \ t \in S'\}$.

A more concerning downside is the size of p: due to subexponential-time algorithms for discrete logarithm over finite fields, p should be taken much larger than 256 bits, at the very least 1024 bits. This forces the group $\mathbb{G}$, over which we instantiate our PCF, to have order $p \geq 2^{1024}$, which considerably harms efficiency (both for key size and computation), and prevents us in particular to rely on efficient 256-bit elliptic curves. We circumvent this issue by setting p to a smaller value (e.g. a 256-bit prime), and relying on Paillier encryption.

A more efficient variant. Assume for simplicity that $p = 2q+1$ for a prime q (the construction also works fine with any large prime factor of $p - 1$). At a high level, we perform the Bellare-Micali-style non-interactive protocol over a Paillier group (similarly as in [OSY21]) followed by a post-processing operation which:

- converts the multiplicative shares over the Paillier group to subtractive shares modulo N (where N is an RSA modulus) using a distributed discrete log algorithm,
- converts the shares modulo N to shares modulo q using the fact that subtractive shares modulo N are with very high probability shares over $\mathbb{Z}$ when the shared value is sufficiently smaller than the modulus,
- converts the additive shares modulo q into multiplicative shares over $\mathbb{Z}_p^*$ via exponentiation.

Let QR_p denote the set of quadratic residues modulo p, which has order q. Let G be a basis of QR_p. Instead of sampling $r \leftarrow\!\!\$\ \mathsf{QR}_p$ directly, Alice samples $\Delta \leftarrow\!\!\$\ \mathbb{Z}_q$ and sets $r \leftarrow G^\Delta \bmod p$ (this yields the same distribution). Let N be a public RSA modulus, whose factorization is unknown to both parties. The protocol proceeds almost as the previous protocol, except that Alice sends a Paillier-ElGamal encryption of Δ (viewed as an integer in $\{0, \cdots, q-1\}$) instead of an ElGamal encryption of r. Let $(\mathbf{G}, \mathbf{H})$ be two random elements of $\mathbb{Z}_{N^2}$. Our protocol borrows ideas from [OSY21]. It builds upon a *distributed discrete logarithm* algorithm DDLOG over $\mathbb{Z}_{N^2}$, which has the following features: given respective multiplicative shares $(S^{\mathsf{send}}, S^{\mathsf{rec}})$ of a value $(1+N)^m$ modulo N^2, the sender and the receiver can locally compute $v^{\mathsf{send}} \leftarrow \mathsf{DDLOG}(\mathsf{send}, S^{\mathsf{send}})$ and $v^{\mathsf{rec}} \leftarrow \mathsf{DDLOG}(\mathsf{rec}, S^{\mathsf{rec}})$ which form subtractive shares of m over $\mathbb{Z}_N$ (*i.e.* $v^{\mathsf{send}} - v^{\mathsf{rec}} = m \bmod N$). Furthermore, if $m < N/2^\lambda$ (when viewed as an integer in $\{0, \cdots, N-1\}$), it holds with probability at least $1 - 2^{-\lambda}$ that $v^{\mathsf{send}} - v^{\mathsf{rec}} = m$ *over the integers.* The work of [OSY21] described an efficient implementation of DDLOG, whose cost boils down to one inversion and one multiplication over $\mathbb{Z}_N$. Given this procedure, our protocol proceeds as follows:

- **Sender to receiver:** pick a random coin ρ, and sends the Paillier-ElGamal ciphertext $(C_0, C_1) \leftarrow (\mathbf{G}^\rho \bmod N, \mathbf{H}^\rho \cdot (1+N)^\Delta \bmod N^2)$.
- **Receiver to sender:** pick n random coins $(\theta_1, \cdots, \theta_n)$ and send the Pedersen commitments $(H_1, \cdots, H_n) \leftarrow (\mathbf{H}^{z_i} \cdot \mathbf{G}^{\theta_i} \bmod N^2)_{i \leq n}$.

- **Output:** for $i = 1$ to n, the sender computes $\mathbf{G}_i^{\mathsf{send}} \leftarrow H_i^{\rho}$, and the receiver computes $\mathbf{G}_i^{\mathsf{rec}} \leftarrow C_1^{z_i} \cdot C_0^{\theta_i}$. Observe that
$$\mathbf{G}_i^{\mathsf{rec}} = C_1^{z_i} \cdot C_0^{\theta_i} = G^{\rho\theta_i} \cdot H^{\rho z_i}(1+N)^{\Delta \cdot z_i} = \mathbf{G}_i^{\mathsf{send}} \cdot (1+N)^{\Delta \cdot z_i} \bmod N^2.$$
Using DDLOG, both parties locally compute values $(v_i^{\mathsf{send}}, v_i^{\mathsf{rec}})$ such that $v_i^{\mathsf{send}} - v_i^{\mathsf{rec}} = b \cdot \Delta \bmod N$. Assuming that $q < N/2^{\lambda}$,[4] it holds that $v_i^{\mathsf{send}} - v_i^{\mathsf{rec}} = b \cdot \Delta$ over $\mathbb{Z}$ with probability at least $1 - 1/2^{\lambda}$. Eventually, the sender outputs $a_i \leftarrow G^{v_i^{\mathsf{send}}}$ and the receiver outputs $\alpha_i \leftarrow G^{v_i^{\mathsf{rec}}}$. Observe that
$$a_i = G^{v_i^{\mathsf{send}}} = G^{v_i^{\mathsf{rec}} + b\cdot\Delta} = \alpha_i \cdot r^b \bmod p.$$

A balancing optimization. In the above protocol, the size of the public keys is quite unbalanced: the sender public key contains a single Paillier-ElGamal ciphertext (in addition to $(g_t)_{t \in S'}$), while the receiver public key contains n Pedersen commitments over $\mathbb{Z}_{N^2}$ (where n is the wPRF key length, e.g. $n = 256$ for our XOR-MAJ candidate, or $n = 770$ for our BIPSW candidate). We now describe an optimization which reduces the receiver key size by a factor k, at the cost of increasing the Paillier-ElGamal ciphertext by a factor k^2. Taking $k = O(n^{1/3})$, this yields a variant in which both public keys contain $O(n^{2/3})$ elements of $\mathbb{Z}_{N^2}$. We note that our balancing optimization also applies to the public key PCF of [OSY21], thus enables reducing their public key size to $O(n^{2/3})$.

The main idea of the optimization is to compress the receiver public key by replacing the Pedersen commitments with a multi-Pedersen commitment. Fix a compression parameter k (which we assume to divide n for simplicity) and public random elements $(\mathbf{G}, \mathbf{H}_1, \cdots, \mathbf{H}_k) \in \mathbb{Z}_{N^2}^{k+1}$. We let the sender commits to z by batches of k values z_i at once, as follows:

- **Receiver to sender:** pick n/k random coins $(\theta_1, \cdots, \theta_{n/k})$ and send the Pedersen commitments $(H_1, \cdots, H_{n/k}) \leftarrow (\mathbf{G}^{\theta_{i+1}} \cdot \prod_{j=1}^{k} \mathbf{H}_j^{z_{j+k\cdot i}} \bmod N^2)_{0 \le i < n/k}$.

Suppose the parties want to retrieve multiplicative shares of $(1+N)^{\Delta \cdot z_i} \bmod N^2$. The main observation is that this can be done using the randomness-reuse variant of Paillier-ElGamal, putting $(1+N)^{\Delta}$ in the first "slot": the sender picks a random coin ρ_1 and computes
$$(C_0, (C_1^j)_{j \le k}) \leftarrow (\mathbf{G}^{\rho_1} \bmod N, \mathbf{H}_1^{\rho_1} \cdot (1+N)^{\Delta}, \mathbf{H}_2^{\rho_1}, \cdots, \mathbf{H}_k^{\rho_1} \bmod N^2).$$
Then, given this extended ciphertext and $H_1 = \mathbf{G}^{\theta_1} \cdot \prod_{j=1}^{k} \mathbf{H}_j^{z_j} \bmod N^2$, the parties retrieve multiplicative shares of $(1+N)^{\Delta \cdot z_1}$ by computing
$$\mathbf{G}_1^{\mathsf{send}} \leftarrow H_1^{\rho_1}, \qquad \mathbf{G}_1^{\mathsf{rec}} \leftarrow C_0^{\theta_1} \cdot \prod_{j \le k} (C_1^j)^{z_j}.$$
The above only yields shares of $(1+N)^{\Delta \cdot z_1}$. To extract shares of $(1+N)^{\Delta \cdot z_j}$ for $j = 2, \cdots, k$, the sender must proceed similarly as above, using extended Paillier-ElGamal ciphertexts, but this time placing $(1+N)^{\Delta}$ in the j-th slot. In total, the sender computes k length-$(k+1)$ extended ciphertexts, for a total of k elements of $\mathbb{Z}_N$ and k^2 elements of $\mathbb{Z}_{N^2}$ (these $k + k^2$ elements can be reused across all n/k batches). The full sender public key is given below:

- **Sender to receiver:** pick random coins ρ_j for $j = 1$ to k, and constructs the extended Paillier-ElGamal ciphertexts as follows for $j = 1$ to k:
$$C_0^j \leftarrow \mathbf{G}^{\rho_j} \bmod N$$
$$(C_1^{j,1} \cdots, C_1^{j,k}) \leftarrow (\mathbf{H}_1^{\rho_j}, \cdots, \mathbf{H}_j^{\rho_j} \cdot (1+N)^{\Delta}, \cdots, \mathbf{H}_k^{\rho_j}) \bmod N^2$$

[4]In practice, we take $\log q = 256$ and $\log N = 3072$.

Efficiency. Without the balancing optimization, sender's public key consists of $|S'|$ elements of $\mathbb{G}$, one element of $\mathbb{Z}_N$, and one element of $\mathbb{Z}_{N^2}$, and receiver's public key consists of n elements of $\mathbb{Z}_{N^2}$. To provide concrete estimates, we use our XOR-MAJ parameter set with $n = 256$ and $|S'| = 390$. We set $\lambda = 128$, $\log |\mathbb{G}| = 2\lambda$, and $\log N = 3072$. With these parameters, the sender public key size is 13.3kB, and the receiver public key size is 192kB. Using the balancing optimization, the public key of the sender consists in $|S'|$ elements of $\mathbb{G}$, k element of $\mathbb{Z}_N$, and k^2 element of $\mathbb{Z}_{N^2}$, and the public key of the receiver consists in n/k elements of $\mathbb{Z}_{N^2}$. With the XOR-MAJ parameter set and using $k = 5$, the sender key increases to 32.8kB while the receiver key is reduced to 38.4kB.

Regarding computation, the cost of deriving the PCF keys from the public and secret keys is dominated by $n+1$ exponentiations modulo N^2 and n exponentiations modulo p for the sender, and $2n$ exponentiations modulo N^2 and n exponentiations modulo p for the receiver. Using the balancing optimization, the number of exponentiations modulo N^2 increases to $n + k^2$ for the sender, and decreases to $n \cdot (1 + 1/k)$ for the receiver. Using $n = 256$ and $k = 5$, this translates to respectively 281 and 307 exponentiations over $\mathbb{Z}_{N^2}$.

Using $\log N = 3072$, an exponentiation modulo N^2 takes of the order of 5ms on one core a standard laptop, which translates to $1 \sim 2$ seconds of computation (note that this is a rough back-of-the-envelope estimation, true estimates may vary). Observe that this can be easily sped up using multiple cores, and that this is a one-time preprocessing phase to generate the shared PCF keys. After generating the PCF keys once, the parties can directly start generating OT correlations. Also, the computational efficiency can be significantly improved by sampling ρ and the θ_i's as 256-bit integers. This improves computation by one to two orders of magnitude, at the (reasonable) cost of having to assume the security of the small-exponent indistinguishability assumption (see e.g. [CC18; CKL+21] for discussions on this assumption and relations to other assumptions).

Concrete Instantiation of DV-NIZK

Following our PK-PCF construction, we define the language $\mathcal{L}_p$ for non-reusable DV-NIZK argument as below

$$\mathcal{L}_p := \left(G^t, H^t(1+N)^{r'}; \left(h^{r^s}, (h')^{r^s}\right)_{s \in S}\right)$$

where $\mathsf{sk}_p := (h, h', r, t)$ and $r := g_q^{r'} \bmod p$.

To build a non-reusable DV-NIZK argument of knowledge, firstly we build a Σ-protocol for language $\mathcal{L}_p$ then later using the construction of [PsV06] with the trapdoor $\mathsf{dv}.\mathcal{T}$ as the list of sk which are used in public-key encryption to obtain a (non-reusable) DV-NIZK scheme that satisfies adaptive knowledge soundness, zero-knowledge.

For instantiation of *strong* PK-PCF, we use our PK-PCF construction plus the techinique [BCE+23] that can convert weak PCF to strong PCF using PRF. We highlight our PK-PCF satisfies the strong security about Supp which we showed directly from the proof security in full version(theorem 3 of [BCM+24]).

Chapter 5

FOLEAGE: $\mathbb{F}_4$OLEAGE-based MPC for Boolean Circuits

In this chapter, we focus on secure computation of general Boolean circuits with multiple parties in the semi-honest setting. Our main contribution is $\mathbb{F}_4$OLEAGE, a novel $\mathbb{F}_4$-OLE-based protocol for secure computation in the preprocessing model that significantly outperforms the state-of-the-art approach in both the two-party and multi-party setting. We provide a detailed technical overview of our results in Section 5.3. We describe our optimized PCG for OLEs over $\mathbb{F}_4$ in Section 5.4. In Section 5.5, we describe our distributed seed generation protocol for PCG-based OLE over $\mathbb{F}_4$. In Section 5.6, we show our parameter choice, report on our implementation and evaluate the performance of our scheme. We present our addtional contribution about N-party MPC with preprocessing from $\mathbb{F}_4$-OLEs in Section 5.7. An optimization for seed expansion is presented in Section 5.8.

Contents

5.1 Motivation and Related Works

A secure multiparty computation (MPC) protocol for a public functionality f allows N parties with private inputs $(x_1, \cdots, x_N)$ to securely compute $f(x_1, \cdots, x_N)$, while concealing all other information about their private inputs to coalitions of corrupted parties. MPC was introduced in the seminal work of Goldreich, Micali, and Wigderson [GMW87] (GMW), and has since led to a rich body of work developing the foundations of MPC, and even practical open-source libraries [Kel20].

Two of the leading paradigms in secure computation are garbled circuits [Yao86] and secret-sharing-based secure computation [GMW87]. The seminal GMW protocol is of the latter type. In a secret-sharing-based MPC protocol, the parties hold shares of the inputs and iteratively compute the circuit representing the function, gate-by-gate. Because addition gates can be computed locally by the parties holding the input shares, only multiplication gates require interaction between the parties to evaluate. As such, the major bottleneck of MPC protocols is due to the communication required to evaluate the multiplication gates in a circuit. (Note that this is also true of the garbled circuit approach where addition gates are "free" and only multiplication gates need to be garbled [KS08].)

However, a core advantage of secret-sharing-based MPC, first identified in the work of Beaver [Bea92], is that secure multiplications can be *preprocessed* in an *input-independent* precomputation phase. In particular, the parties can securely generate additive shares of many "Beaver triples" $(a, b, a \cdot b) \in \mathbb{F}^3$. Then, for each multiplication gate that needs to be computed in the online phase, the parties can run a fast information-theoretically secure multiplication protocol that consumes one Beaver triple and involves communicating just two elements of $\mathbb{F}$ per party. This model of secure computation with preprocessing forms the basis for modern MPC protocols due to the efficiency of the online phase. However, this preprocessing paradigm only serves to push the inefficiency bottleneck of MPC to the offline phase that consists of generating many Beaver triples. We briefly survey the different techniques that have been developed in the last couple of decades for the efficient generation of Beaver triples in an MPC setting.

Modern secure computation protocols. The traditional approach for securely generating Beaver triples relies on Oblivious Transfers (OT) [Rab81; EGL82]: an N-party Beaver triple over $\mathbb{F}$ is generated by letting each *pair* of parties execute $\log |\mathbb{F}|$ oblivious transfers [Gil99], and thanks to OT *extension* protocols [Bea96; IKN+03], generating a large number of OTs requires only cheap symmetric-key operations. This OT-based approach is very competitive with a small number of parties, but becomes very inefficient with many parties. Specifically, because *each pair* of parties needs to perform OTs, the communication and computation costs are on the order of $\Omega(N^2)$, which quickly becomes impractical as N grows large.

Over the past decade, the practicality of secure computation has increased tremendously [DPS+12; KOS16; Kel20; DNN+17; HOS+18; KPR18]. This is especially true in the setting of secure computation of *arithmetic circuits over large fields*. Starting with the celebrated SPDZ protocol [DPS+12], a sequence of works has developed fast protocols that use Ring-LWE-based somewhat homomorphic encryption, or even linearly homomorphic encryption, to generate m Beaver triples with only

$O(m \cdot N)$ communication and computation per triple. These approaches significantly improve over the "naïve" $\Omega(m \cdot N^2)$ cost of the OT-based approach. Over sufficiently large fields (e.g., larger than 2^λ), when generating many triples, state-of-the-art protocols such as Overdrive [KPR18] achieve very good concrete efficiency.

More recently, following the line of work on silent secure computation initiated in [BCG+18; BCG+19b; BCG+19a], Boyle et al. [BCG+20b] have shown how to generate a large number m of *pseudorandom* (as opposed to truly random) Beaver triples under the Ring-LPN assumption. Their approach uses $O(\log m \cdot N^2)$ communication, followed solely by *local* computation, with good concrete efficiency (the authors estimated a throughput of around 10^5 triples per second on one core of a standard laptop). For sufficiently large values of m, this is highly competitive with Overdrive. However, both Overdrive and the existing PCG-based approach share a common restriction: they are only usable over large fields.

Secure computation of Boolean circuits. In contrast to the secure computation of arithmetic circuits over large fields, the fastest way to run N-party MPC protocols for Boolean circuits remains the "naïve" method of generating many pairwise OTs, at a cost of $\Omega(m \cdot N^2)$ bits for m Beaver triples. This is in contrast to the *two-party* setting, where two-party Beaver triples can be generated very efficiently thanks to a recent line of work [BCG+18; BCG+19b; BCG+19a] on silent OT extension. In silent OT extension, two parties can generate m Beaver triples using only $O(\log m)$ communication. The state-of-the-art protocols in this area [CRR21; BCG+22; RRT23] achieve impressive throughputs of several million Beaver triples per second on one core of a standard laptop. Furthermore, the recent SoftSpoken OT extension protocol [Roy22] yields even faster OTs at the cost of increasing communication. For example, SoftSpoken can generate nearly 30M OT/s on `localhost` at the cost of increasing the communication to $64m$ bits to generate m Beaver triples; other communication/computation tradeoffs are possible [Roy22, Table 1].[1]

The situation, however, is much less satisfying for the setting of secure computation of Boolean circuits with a larger number of parties. Protocols such as SPDZ [DPS+12] and Overdrive [KPR18] do not perform well when generating Beaver triples for Boolean circuits, even in the passive setting. This is due to the high overhead of embedding $\mathbb{F}_2$ in an extension field compatible with the number theoretic-transform used in efficient instantiations of the BGV encryption scheme [BGV14]. Furthermore, silent OT extension techniques build on Pseudorandom Correlation Generators (PCGs), which typically work only in the two-party setting [BCG+19b]. To handle more parties, one needs the stronger notion of *programmable* PCG [BCG+20b], which, informally, allows partially specifying parts of the generated correlation. Unfortunately, while efficient programmable PCGs over large fields were introduced in [BCG+20b], building *concretely efficient*, programmable PCGs over $\mathbb{F}_2$ has remained elusive thus far, making N-party PCGs for $\mathbb{F}_2$ primarily of theoretical interest. The state-of-the-art is the recent work of Bombar et al. [BCC+23], which generates Beaver triples over any field $\mathbb{F}_q$ with $q \geq 3$. However, Bombar et al. [BCC+23] leave analyzing the concrete efficiency for future work.

In light of this state of affairs, to the best of our knowledge, the current most efficient approach for N-party secure computation of Boolean circuits remains the classical OT-based approach. In a little more detail, to generate each Beaver triple, each party P_i samples a random pair (a_i, b_i) of bits, and each pair (P_i, P_j) of parties executes two oblivious transfer protocols to generate additive shares of $a_i b_j$ and $a_j b_i$. Then, all parties aggregate their shares to obtain shares of $\sum_{i,j} a_i b_j = (\sum_i a_i) \cdot (\sum_j b_j)$. When generating m Beaver triples, this approach requires $N \cdot (N-1) \cdot m$ oblivious transfers in total (to be compared with the $O(N^2 \cdot \log m)$ communication of [BCG+20b], or the $O(N \cdot m)$ communication

[1]Note that we need two calls to the OT functionality to generate one Beaver triple.

of Overdrive [KPR18], for the case of arithmetic circuits over large fields). While there has been tremendous progress in constructing efficient OT protocols [IKN+03; Roy22], even using silent OT extension (which has the lower communication overhead) requires $3N \cdot (N-1) \cdot m$ bits of communication (ignoring some $o(m)$ terms). Using SoftSpoken OT [Roy22] instead, which appears to be the most computationally efficient solution, and setting the "communication/computation tradeoff" parameter k to $k = 5$, the communication increases to $32N \cdot (N-1) \cdot m$ bits. When the number of parties grows, this soon becomes very inefficient.[2]

5.2 Detailed Contributions

In this chatpter, we focus on secure computation of general Boolean circuits with multiple parties in the semi-honest setting. Our main contribution is $\mathbb{F}_4$OLEAGE, a novel $\mathbb{F}_4$-OLE-based protocol for secure computation *in the preprocessing model* that significantly outperforms the state-of-the-art approach in both the *two-party* and *multi-party* setting. In particular, $\mathbb{F}_4$OLEAGE enjoys much lower communication in the preprocessing phase than all known alternatives and has a very low computational overhead. We expect $\mathbb{F}_4$OLEAGE to be the fastest alternative for large enough circuits on almost any realistic network setting, for any number of parties between two and several hundred. $\mathbb{F}_4$OLEAGE builds upon recent results constructing efficient PCGs and introduces several protocol-level, algorithmic-level, and implementation-level optimizations to make these PCG constructions significant fast (see Section 5.6 for a performance evaluation).

In the two-party setting. ($N = 2$), $\mathbb{F}_4$OLEAGE enjoys a silent preprocessing (generating m multiplication triples requires $O(\log m)$ communication), and significantly outperforms all previous silent protocols. In particular, our implementation generates around 12.3 million Beaver triples *per second* on one core of an Amazon `c5.metal` server. Compare this to the state-of-the-art silent OT protocol RRT [RRT23] which generates 3.4 million Beaver triples per second with the same setup. This makes RRT more than 3.5 times slower compared to $\mathbb{F}_4$OLEAGE. The fastest *non-silent* OT protocol, SoftSpoken OT, generates around 26 million multiplication triples per second on `localhost` in its fastest regime (using $k = 2$ [Roy22, Table 1]), while requiring around $128 \cdot m$ bits of total communication. However, while our approach does achieve a blazing-fast throughput, it has some limitations. In particular, the preprocessing phase of $\mathbb{F}_4$OLEAGE requires more rounds (16 rounds instead of 3 for generating 26M triples compared to [Roy22]). Additionally, our seed size is roughly $130\times$ larger compared to [RRT23], and $2\times$ larger compared to [BCG+20b]. This makes $\mathbb{F}_4$OLEAGE less suitable for generating a small number of triples. Eventually, our protocols are tailored to the generation of multiplication triples over $\mathbb{F}_2$ in the semi-honest setting: their efficiency scales less favorably in other settings, such as generating string OTs or authenticated triples.

In the multi-party setting. ($N > 2$), $\mathbb{F}_4$OLEAGE achieves *almost-silent* preprocessing: to securely compute a circuit with m AND gates, following a silent phase with $O(N^2 \cdot \log m)$ communication, our preprocessing phase requires a single broadcast of $N \cdot m$ bits (one bit per AND gate and per party), and the online phase is the standard GMW protocol. As N grows, this represents a drastic reduction in communication compared to the $\sim 3 \cdot N^2 m$ communication obtained when using silent OT extension, or the $\sim 32 \cdot N^2 m$ communication obtained with SoftSpoken OT, while remaining highly competitive in terms of computation.

[2]For a very large number of parties, the linear scaling in N of Overdrive should become favorable. However, after private communication with the authors of Overdrive, the break-even point for communication seems to happen only for values of N in the range of 400+, due to the high overhead of using BGV and embedding $\mathbb{F}_2$ elements.

	Communication	`localhost`	**LAN**	**WAN**
Multi-party setting ($N = 10$)				
SoftSpoken ($k = 2$)	134 GB	342s	1192s	12207s
SoftSpoken ($k = 4$)	67 GB	405s	596s	6104s
SoftSpoken ($k = 8$)	34 GB	1900s	**1900s**	3052s
			*298s	
RRT	6.3 GB	2619s	**2619s**	**2619s**
			*50.3s	*515s
$\mathbb{F}_4$OLEAGE	0.7 GB	1463s	**1463s**	**1463s**
			*5.6s	*57.9s
Two-party setting ($N = 2$)				
SoftSpoken ($k = 2$)	15 GB	38s	119s	1221s
SoftSpoken ($k = 4$)	7.5 GB	45s	60s	610s
SoftSpoken ($k = 8$)	3.7 GB	211s	**211s**	**211s**
RRT	258 KB	292s	**292s**	**292s**
$\mathbb{F}_4$OLEAGE	33.5 MB	81s	**81s**	**81s**

Table 5.1: Comparison of state-of-the-art protocols to generate N-party Beaver triples over $\mathbb{F}_2$ for $N = 10$ and $N = 2$ parties. The `localhost` column reports the runtimes (ignoring communication) for generating 10^9 triples. All protocols run on one core of AWS `c5.metal` (3.4GHz CPU); all runtimes averaged across ten trials.

Comparison with the state of the art. In Table 5.1, we provide a comparison between $\mathbb{F}_4$OLEAGE, SoftSpoken, and RRT, for $N = 10$ and $N = 2$ parties. In the multiparty setting, due to the very low bandwidth requirement of $\mathbb{F}_4$OLEAGE, we observe that computation is systematically the bottleneck when evaluated on one core of a commodity server. This indicates that $\mathbb{F}_4$OLEAGE is likely to stand out even more whenever more computational power is available, e.g., when evaluated in parallel on multiple cores.

The numbers in Table 5.1 have been computed using the running time T measured for generating 3^{16} OLEs (using the noise parameter $t = 27$ and $c = 3$), and estimating the per-party cost to generate 10^9 N-party Beaver triples as $2 \cdot (N-1) \cdot T \cdot (10^9/3^{16})$. When $N = 2$, the cost is estimated as $T \cdot (10^9/3^{16})$, accounting for the factor-2 saving tailored to the 2-party setting. For communication, we computed an estimate of $C = 13$MB of communication for our distributed protocol for generating a seed for 3^{18} OLEs. While one could in principle directly generate a seed that stretches to 10^9 OLEs, this would significantly slow down the computation as the 10^9 OLEs must be expanded all at once, and would not fit in memory. Hence, we estimate the communication as $2 \cdot (N-1) \cdot (3^{18}/10^9) \cdot C$ for generating 10^9 N-party Beaver triples (as 3^{18} OLEs is the maximum expansion size we could fit in the memory), and an additional 10^9 bits of communication per party (in the setting $N > 2$).

5.3 Technical Overview

In this chapter, we provide a detailed description of our results and the main technical ideas underlying them. In Section 5.3.1, we provide background on secure multi-party computation realized from PCGs for OLE correlations. In Section 5.3.2 we describe the PCG construction of [BCC+23], which forms the basis for our preprocessing protocol. In Section 5.3.3, we describe our idea for converting

$\mathbb{F}_4$ triples into $\mathbb{F}_2$ triples, which we tailor to the two-party case in Section 5.3.4. In Section 5.3.5, we describe our optimized PCG construction. In Section 5.3.6, we explain how we can obtain an efficient distributed seed generation protocol for our PCG construction.

Notations. Unless otherwise stated, an N-party linear secret shares of a value v is denoted $[\![v]\!] = ([\![v]\!]_1, \ldots, [\![v]\!]_N)$, where the i-th party obtains share $[\![v]\!]_i$. To disambiguate shares over $\mathbb{F}_4$ and shares over $\mathbb{F}_2$, we denote the field size with a superscript, i.e., $[\![\cdot]\!]^4$ and $[\![\cdot]\!]^2$, respectively. We identify $\mathbb{F}_4$ with $\mathbb{F}_2[X]/(X^2+X+1)$ and let θ denote a primitive root of X^2+X+1. Given an element $x \in \mathbb{F}_4$, we write $x(0)$ and $x(1)$ to denote the $\mathbb{F}_2$-coefficients of x viewed as a polynomial over $\mathbb{F}_2[X]/(X^2+X+1)$; that is, $x = x(0) + \theta \cdot x(1)$.

5.3.1 Background: Secure MPC from PCGs

We start by describing prior approaches to realizing MPC in the preprocessing model from PCGs for OLE correlations.

PCGs for the OLE correlation. Our starting point is the template for generating N-party pseudorandom Beaver triples put forth by Boyle et al. [BCG+20b]. At the heart of their framework is the use of a programmable PCG [BCG+20b] for the OLE correlation. Concretely, a PCG for a target correlation C (i.e., a distribution over pairs of strings) is a pair of algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Eval})$ such that

- $\mathsf{PCG.Gen}$ generates a pair of *succinct* keys $(\mathsf{k}_0, \mathsf{k}_1)$ jointly encoding the target correlation, and
- $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ produces a string R_σ corresponding to party σ's secret share of the target correlation.

At a high level, a PCG must satisfy two properties: (1) *pseudorandomness* (or correctness) which states that (R_0, R_1) must be indistinguishable from a random sample from C, and (2) *security* which states that R_σ should appear random conditioned on satisfying the target correlation with $R_{1-\sigma} = \mathsf{PCG.Expand}(1-\sigma, \mathsf{k}_{1-\sigma})$ even given $\mathsf{k}_{1-\sigma}$, for $\sigma \in \{0,1\}$.

We focus on the OLE correlation over a finite field $\mathbb{F}$. For a length-m OLE correlation, the string R_0 (which we call the *sender* output) is a list of m tuples $(u_i, v_i)_{i \leq m} \in (\mathbb{F}^2)^m$, and the string R_1 (which we call the *receiver* output) is a list of m pairs $(x_i, w_i)_{i \leq m} \in (\mathbb{F}^2)^m$ such that $w_i = u_i \cdot x_i + v_i$ for every i. Observe that, we can equivalently view v_i and $-w_i$ as additive shares of $u_i \cdot x_i$, which we will denote as $[\![u_i \cdot x_i]\!]$. Informally, security for the OLE correlation amounts to showing that the following two properties hold:

- **Sender security:** from the viewpoint of the receiver (who has k_1 and generates (x_i, w_i)), the distribution of (u_i, v_i) is computationally indistinguishable from the distribution of $(u_i, w_i - u_i \cdot x_i)$, for a uniformly random $u_i \leftarrow\$ \mathbb{F}$.
- **Receiver security:** from the viewpoint of the sender (who has k_0), the distribution of each x_i is computationally indistinguishable from a random field element.

Going from OLE to Beaver triples. As shown in [BCG+19b], given a PCG for the OLE correlation (or a PCG for OLE for short), two parties can generate many pseudorandom Beaver triples over $\mathbb{F}$ as follows. First, the parties compute $\mathsf{PCG.Gen}$ via a two-party secure computation protocol to obtain PCG keys k_0 and k_1, respectively. Then, using $\mathsf{PCG.Expand}$, the two parties locally obtain many correlations of the form $(u_i, [\![u_i x_i]\!]_0)$ and $(x_i, [\![u_i x_i]\!]_1)$, respectively. Given two such OLE correlations, where one party has $(u_0, u_1, [\![u_0 x_0]\!]_0, [\![u_1 x_1]\!]_0)$ and the other party has $(x_0, x_1, [\![u_0 x_0]\!]_1, [\![u_1 x_1]\!]_1)$, the two parties can *locally* derive one Beaver triple of the form $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket ab \rrbracket)$ by computing:

$$(\underbrace{\llbracket u_0 + x_1 \rrbracket}_{\llbracket a \rrbracket},\ \underbrace{\llbracket u_1 + x_0 \rrbracket}_{\llbracket b \rrbracket},\ \llbracket u_0x_0 + u_1x_1 \rrbracket + u_0u_1 + x_0x_1 = \underbrace{\llbracket (u_0 + x_1) \cdot (u_1 + x_0) \rrbracket}_{\llbracket ab \rrbracket}).$$

In a little more detail, the sender computes their share of the Beaver triple as $(u_0, u_1, \llbracket u_0x_0 \rrbracket_0 + \llbracket u_1x_1 \rrbracket_0 + u_0u_1)$ and the receiver computes their share as $(x_1, x_0, \llbracket u_0x_0 \rrbracket_1 + \llbracket u_1x_1 \rrbracket_1 + x_0x_1)$. While this technique works well in the two-party setting, in the *multi*-party setting, things are not so simple.

Going from two parties to many parties. As first discussed by Boyle et al.[BCG+20b], to generate N-party Beaver triples using a PCG for OLE, the parties need to ensure *consistency* among the OLE correlations generated by *each pair of parties*. That is, to generate one multiplication triple $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket ab \rrbracket)$, we need each pair of parties (P_i, P_j) to hold respective values (a_i, b_i) and (a_j, b_j) (viewed as an individual share of a and b), together with two-party shares $\llbracket a_ib_j \rrbracket$ and $\llbracket a_jb_i \rrbracket$. Then, all parties can combine their shares to get

$$\llbracket (\textstyle\sum_i a_i) \cdot (\sum_j b_j) \rrbracket = \sum_{i \neq j} \llbracket a_ib_j \rrbracket + \sum_i a_ib_i.$$

Observe that this requires party P_i to have OLEs of the form $(a_i, \llbracket a_ia_j \rrbracket_i)$, with every other party P_j (who in turn has share $(a_j, \llbracket a_ia_j \rrbracket_j)$), *where P_j's value a_i remains the same across all OLEs.* This is precisely what the notion of a *programmable* PCG for OLE achieves: it allows the parties to specify seeds (ρ_0, ρ_1) such that $\mathsf{PCG.Gen}(\rho_0, \rho_1)$ outputs keys $\mathsf{k}_0, \mathsf{k}_1$ that, informally speaking, have all the pseudorandom (a_i, b_i) deterministically generated from the seeds ρ_0 and ρ_1 respectively (while still maintaining the required security properties). By reusing the same seeds across executions with multiple parties, the parties can ensure the required consistency across their outputs.

5.3.2 Constructing Programmable PCGs

In addition to defining the notion of programmable PCGs, the work of Boyle et al. [BCG+19b; BCG+20b] introduced a construction from a variant of the LPN assumption over rings. At a high level, the ring-LPN assumption they introduce states that $(a, as + e)$ is hard to distinguish from (a, b), where a, b are random polynomials from a suitable ring $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$, where P splits into $\deg(P)$ linear factors and s, e are random *sparse* polynomials from $\mathcal{R}$. The construction of Boyle et al. proceeds by generating a single large pseudorandom OLE correlation over a polynomial ring $\mathcal{R} = \mathbb{F}_q[X]/(P(X))$, assuming the hardness of the ring-LPN assumption over $\mathcal{R}$. When P splits into $D = \deg(P)$ linear factors, the Chinese Remainder Theorem makes it possible to convert this large OLE correlation over $\mathcal{R}$ into D OLE correlations over $\mathbb{F}_q$ (by reducing it modulo each of the factors of P). Unfortunately, the condition that P splits requires $|\mathbb{F}_q| \geq D$, which restricts the construction to only work over large fields. This makes the resulting OLE correlations only suitable for generating Beaver triples over $\mathbb{F}_q$, which limits their applications. Moreover, other existing efficient (non-PCG-based) protocols for generating Beaver triples are also restricted to large fields [DPS+12; KPR18]. However, for the Boolean circuit case, the state-of-the-art remains the basic OT-based approach originally proposed in the GMW protocol.

A programmable PCG for $\mathbb{F}_4$-OLE. The large-field restriction of the Boyle et al.'s PCG construction was recently overcome by Bombar et al. [BCC+23]. At a high-level, the authors of [BCC+23] manage to replace the polynomial ring $\mathcal{R}$ by a suitable *Abelian group algebra* $\mathbb{F}[\mathbb{G}]$ (that is, the set of formal sums $\sum_{g \in \mathbb{G}} a_g g$ for $a_g \in \mathbb{F}$, where $\mathbb{G}$ is an Abelian group; endowed with the convolution product), which identifies to some ring of multivariate polynomials. Moreover, they show that an appropriate choice of Abelian group algebra can simultaneously satisfy the following properties, for almost every choice of finite field $\mathbb{F}$:

1. $\mathbb{F}[\mathbb{G}]$ is isomorphic to many copies of $\mathbb{F}$ (note that this property is necessary to convert an OLE correlation over $\mathbb{F}[\mathbb{G}]$ into many OLEs over $\mathbb{F}$),

2. The assumption that $(a, as + e)$ is indistinguishable from random over $\mathbb{F}[\mathbb{G}] \times \mathbb{F}[\mathbb{G}]$, with $a \leftarrow\$ \mathbb{F}[\mathbb{G}]$ and (s, e) two random *sparse* elements of $\mathbb{F}[\mathbb{G}]$ (with respect to the canonical notion of sparsity over the group algebra, i.e., sparse formal sums $\sum_{g \in \mathbb{G}} a_g g$) is a plausible assumption,

3. Operations over $\mathbb{F}[\mathbb{G}]$ can be computed efficiently using a Fast Fourier Transform (FFT) algorithm[Obe07; BCC+23].

The second property is a new variant of the syndrome decoding (or LPN) assumption which the authors called *Quasi-Abelian Syndrome Decoding*. It naturally extends to a "module"-variant, i.e. the indistinguishability of pairs $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$ where $\mathbf{s}$ and e are drawn from a sparse distribution, and generalizes both the quasi-cyclic syndrome decoding (when $\mathbb{G}$ is a cyclic group), and the LPN or syndrome decoding assumption (when $\mathbb{G} = \{1\}$). The work of Bombar et al. [BCC+23] also provides extensive support for this assumption by showing that it resists all *linear attacks*, a class of attacks capturing the most known attacks on the LPN assumption and its variants, and proposes a set of parameters resisting all concrete attacks known at that time. The combination of these three properties allowed them to build an efficient programmable PCG for OLEs over $\mathbb{F}$.

Despite the progress made in [BCC+23], their programmable PCG construction is limited in that it applies only to generating OLE correlations over all finite fields $\mathbb{F}$ *except for* $\mathbb{F}_2$. This stems from the fact that there does not exist any group $\mathbb{G}$ such that $\mathbb{F}_2[\mathbb{G}]$ is isomorphic to $\mathbb{F}_2^n$ for $n > 1$ (see [BCC+23, Theorem 47]). In contrast, the case of $\mathbb{F}_2$, is *precisely* the case that we are interested in when considering Boolean circuits, which require generating Beaver triples over $\mathbb{F}_2$.

Additionally, the *concrete* efficiency of an FFT computed over the group algebra remains unclear, since Bombar et al. left estimating the performance of FFTs on $\mathbb{F}[\mathbb{G}]$ for future work. As such, the concrete efficiency of their programmable PCG construction is unknown, making it difficult to determine whether or not it is sufficiently efficient to be applied in practical applications (all other components of their construction consist of standard tools used in the PCG literature, which are known to have concretely efficient implementations).

Our contribution. Looking ahead, our main contribution is to build upon the work of Bombar et al. through a number of simple yet powerful observations that allow us to arrive at an efficient PCG for Beaver triples, suitable for use in secure multi-party computation of Boolean circuits.

- First, we show that we can use their programmable PCG for generating OLEs *over* $\mathbb{F}_4$ to generate multiplication triples *over* $\mathbb{F}_2$, sidestepping the "$\mathbb{F}_2$ barrier" of their PCG construction, at the cost of a *single bit of communication* per triple and per party in the preprocessing phase, or even without any communication when $N = 2$.

- Second, we introduce a number of concrete optimizations to the PCG construction of Bombar et al. [BCC+23] that are tailored to the special case of $\mathbb{F} = \mathbb{F}_4$, which gives us an incredibly efficient programmable PCG over $\mathbb{F}_4$. Compared with the fastest previous programmable PCGs of [BCG+20b], our optimized implementation shows that our construction is *two orders of magnitude* faster.

In the next few subsections, we provide more details on the above contributions.

5.3.3 $\mathbb{F}_2$-triples from $\mathbb{F}_4$-triples

Since $\mathbb{F}_4$ is an extension field of $\mathbb{F}_2$, a Boolean circuit can be viewed as an $\mathbb{F}_4$-arithmetic circuit. Hence, using an OLE correlation over $\mathbb{F}_4$ to construct N-party Beaver triples over $\mathbb{F}_4$ directly yields an MPC protocol for *Boolean* circuits in the preprocessing model via the GMW template [GMW87]. Unfortunately, compared to using $\mathbb{F}_2$-Beaver triples, the communication in the *online* phase is doubled, because each party has to send two elements of $\mathbb{F}_4$ per AND gate, hence 4 bits instead of 2 with GMW.

Our core observation is that one can make much better use of these N-party multiplication triples over $\mathbb{F}_4$: we show how to convert an $\mathbb{F}_4$-multiplication triple into an $\mathbb{F}_2$-multiplication triple using *a single bit of communication* per party. Once converted into $\mathbb{F}_2$-triples, these triples can be used within the standard GMW protocol that communicates two bits per party and per AND gate in the online phase. To explain the observation, let $(\llbracket a \rrbracket^4, \llbracket b \rrbracket^4, \llbracket ab \rrbracket^4)$ be a Beaver triple over $\mathbb{F}_4$. Writing $x = x(0) + \theta \cdot x(1)$ for any $x \in \mathbb{F}_4$, with θ a root of the polynomial $X^2 + X + 1$ (hence $\theta^2 = \theta + 1$), we have

$$\begin{aligned} a \cdot b =& a(0)b(0) + a(1)b(1) + \theta \cdot (a(0)b(1) + a(1)b(0) + a(1)b(1)) \\ &\implies (ab)(0) = a(0)b(0) + a(1)b(1). \end{aligned}$$

Now, assume that the parties reconstruct $b(1)$, which can be done using a single bit of communication per party from their shares $\llbracket b \rrbracket^4 = \llbracket b(0) \rrbracket^2 + \theta \cdot \llbracket b(1) \rrbracket^2$. Given $b(1)$, the parties can locally compute shares of $a(0)b(0)$ as follows:

$$\llbracket a(0)b(0) \rrbracket^2 = \llbracket ab \rrbracket^4(0) + b(1) \cdot \llbracket a \rrbracket^4(1).$$

Therefore, all parties output $(\llbracket a(0) \rrbracket^2, \llbracket b(0) \rrbracket^2, \llbracket ab \rrbracket^4(0) + b(1) \cdot \llbracket a \rrbracket^4(1))$, which forms a valid Beaver triple over $\mathbb{F}_2$. Security is straightforward: the only communication between the parties is the reconstruction of $b(1)$, which is a uniformly random bit independent of $a(0), b(0)$. From there, one immediately gets an improved protocol in the preprocessing model: in the preprocessing phase, given one $\mathbb{F}_4$-Beaver triple for each AND gate of the circuit, the parties broadcast one bit per gate, and then locally derive the $\mathbb{F}_2$-Beaver triples. In the online phase, the parties run the standard GMW protocol.]]

5.3.4 Improved Protocol from $\mathbb{F}_4$-OLEs for $N = 2$

In the setting of $N = 2$ parties, we obtain a much more efficient alternative: we observe that two parties can directly convert a single OLE over $\mathbb{F}_4$ into a Beaver triple over $\mathbb{F}_2$. (In contrast, recall that the standard approach requires two oblivious transfers for each triple.) We consider two parties, Alice and Bob, holding respectively $(a, \llbracket ab \rrbracket^4_A)$ and $(b, \llbracket ab \rrbracket^4_B)$ for a and $b \in \mathbb{F}_4$. We have

$$\begin{aligned} a \cdot b &= \llbracket ab \rrbracket^4_A(0) + \llbracket ab \rrbracket^4_B(0) + \theta \cdot (\llbracket ab \rrbracket^4_A(1) + \llbracket ab \rrbracket^4_B(1)) \\ &= (a(0)b(0) + a(1)b(1)) + \theta \cdot (a(0)b(1) + a(1)b(0) + a(1)b(1)), \end{aligned}$$

where θ is the primitive root of $X^2 + X + 1$. Considering only the $(a \cdot b)(0)$ term from the above equation (i.e., the parts not multiplied by θ), we get that

$$(a \cdot b)(0) = \llbracket ab \rrbracket^4_A(0) + \llbracket ab \rrbracket^4_B(0) = a(0)b(0) + a(1)b(1), \text{ and therefore,}$$

$$\underbrace{a(0)a(1) + \llbracket ab \rrbracket^4_A(0)}_{\text{known by } A} + \underbrace{b(0)b(1) + \llbracket ab \rrbracket^4_B(0)}_{\text{known by } B} = \underbrace{(a(0) + b(1))}_{\text{shared by } A,B} \cdot \underbrace{(a(1) + b(0))}_{\text{shared by } A,B}.$$

Above, the values $a(0)a(1) + [\![ab]\!]_A^4(0)$ (known by Alice) and $b(0)b(1) + [\![ab]\!]_B^4(0)$ (known by Bob) form additive shares of the product $(a(0) + b(1)) \cdot (a(1) + b(0))$, which Alice and Bob hold additive shares of. It is also easy to check that if the input is a random $\mathbb{F}_4$-OLE, the output is a random multiplication triple over $\mathbb{F}_2$. Therefore, following the local conversion procedure outlined above, Alice and Bob can transform a random $\mathbb{F}_4$-OLE instance into a random Beaver over $\mathbb{F}_2$ without having to communicate. We refer the reader to Section 5.7 of the Supplementary material for the formal statement of this optimization.

5.3.5 Fast Programmable PCG for $\mathbb{F}_4$-OLEs

In light of the above observations, the only missing piece of the puzzle is an efficient way of generating a large number of $\mathbb{F}_4$-OLEs. In the $N > 2$ setting, if the OLEs are additionally programmable, the parties can afterward locally convert $N \cdot (N-1)$ $\mathbb{F}_4$-OLE instances into an $\mathbb{F}_4$-Beaver triples.

Here, we build on the recent general programmable PCG construction of [BCC+23]. Because we are targeting OLEs over $\mathbb{F}_4$, we set the group $\mathbb{G}$ to $\mathbb{F}_3^n$, and the underlying group algebra becomes isomorphic to

$$\mathbb{F}_4[\mathbb{G}] \simeq \mathbb{F}_4[X_1, \dots, X_n]/(X_1^3 - 1, \dots, X_n^3 - 1) \simeq \mathbb{F}_4^{3^n}.$$

Before delving into the optimizations we develop for their construction, we describe the high-level ideas and main building blocks behind the PCG construction of Bombar et al. [BCC+23] when instantiated over $\mathbb{F}_4$.

The PCG construction of Bombar et al. As with previous constructions of PCGs [BCG+18; BCG+19b], the construction of Bombar et al. uses Distributed Point Functions (DPF) [BGI15; BGI16; GI14] as a core building block. Informally, a DPF with domain $[D]$ allows a dealer to succinctly secret share a unit vector over $[D]$. The most efficient DPFs have shares of size roughly $\lambda \cdot \log D$ [BGI16], for some security parameter λ, and the cost of decompressing the shares is dominated by D calls to a length-doubling pseudorandom generator.

Public parameters. For a fixed *compression factor* c (typically a small constant, e.g., $c = 3$) and *noise parameter* t (e.g., $t = 27$), the public parameters contain a length-c vector $\mathbf{a}$ of n-variate polynomials.

Distributing PCG seeds. In their construction, PCG.Gen does the following:

- it samples two length-c vectors $(\mathbf{e}_0, \mathbf{e}_1)$ of t-sparse polynomials over $\mathbb{F}_4[\mathbb{G}]$;
- outputs keys $(\mathsf{k}_0, \mathsf{k}_1)$ that contain $\mathbf{e}_0$ and $\mathbf{e}_1$, respectively, as well as *succinct* shares of $\mathbf{e}_0 \otimes \mathbf{e}_1$, encoded using a DPF.

The tensor product $\mathbf{e}_0 \otimes \mathbf{e}_1$ contains c^2 polynomials, each with at most t^2 nonzero coordinates. Hence, the vectors of coefficients of all polynomials in $\mathbf{e}_0 \otimes \mathbf{e}_1$ can be succinctly secret shared using $(ct)^2$ DPFs with domain 3^n, which requires roughly $(ct)^2 \cdot \lambda \log(3^n)$ bits using the state-of-the-art DPF constructions [BGI15; BGI16].[3]

Generating correlations. To output a vector of OLE correlations, PCG.Eval proceeds as follows for party 0 (the evaluation for party 1 is similar):

- evaluate all the DPFs to obtain a secret share of $[\![\mathbf{e}_0 \otimes \mathbf{e}_1]\!]_0$;
- set $x_0 \leftarrow \langle \mathbf{a}, \mathbf{e}_0 \rangle$ and $z_0 \leftarrow \langle \mathbf{a} \otimes \mathbf{a}, [\![\mathbf{e}_0 \otimes \mathbf{e}_1]\!]_0 \rangle$; $\triangleright$ Note: $z_0 = [\![\langle \mathbf{a} \otimes \mathbf{a}, \mathbf{e}_0 \otimes \mathbf{e}_1 \rangle]\!]_0$
- using the isomorphism $\mathbb{F}_4[\mathbb{G}] \simeq \mathbb{F}_4^{3^n}$, project $(x_0, z_0) \in \mathbb{F}_4[\mathbb{G}]^2$ onto 3^n pairs (x_0^i, z_0^i) of elements of $\mathbb{F}_4$.

[3]Using noise vector with a regular structure, the domain size of the DPFs can be reduced to $3^n/t$.

Above, the projection amounts to evaluating the multivariate polynomials over $\mathbb{F}_4[X_1, \ldots, X_n]/(X_1^3 - 1, \ldots, X_n^3 - 1)$ on the 3^n tuples of elements of $(\mathbb{F}_4^\times)^n$. Observe that

$$\begin{aligned} z_0 + z_1 &= \langle \mathbf{a} \otimes \mathbf{a}, [\![\mathbf{e}_0 \otimes \mathbf{e}_1]\!]_0 \rangle + \langle \mathbf{a} \otimes \mathbf{a}, [\![\mathbf{e}_0 \otimes \mathbf{e}_1]\!]_1 \rangle \\ &= \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{e}_0 \otimes \mathbf{e}_1 \rangle = \langle \mathbf{a}, \mathbf{e}_0 \rangle \cdot \langle \mathbf{a}, \mathbf{e}_1 \rangle = x_0 \cdot x_1. \end{aligned}$$

Since the isomorphism preserves additions and multiplications, it follows that all pairs (x_0^i, z_0^i) and (x_1^i, z_1^i) form OLEs over $\mathbb{F}_4$. Security boils down to the Quasi-Abelian Syndrome Decoding assumption (QA-SD) [BCC+23], which states (informally) that given the random vector $\mathbf{a}$, the element $\langle \mathbf{a}, \mathbf{e} \rangle + e_0$ (where $(e_0, \mathbf{e})$ are formed by random sparse polynomials) is indistinguishable from a random element of $\mathbb{F}_4[\mathbb{G}]$.

We now describe several observations that we make about their construction and how these observations allow us to significantly optimize the concrete efficiency of the PCG. While simple in retrospect, these observations allow us to turn a theoretical construction into a concretely efficient PCG for $\mathbb{F}_4$-OLEs (see Section 5.6 for our implementation and evaluation).

Early termination. The DPF construction of [BGI16] generates shares of a unit vector using a construction *à la* GGM [GGM19], generating a full binary tree of PRG evaluations starting from a root seed. The children of each node are computed by evaluating a length-doubling PRG on the node, and then adding some correction words. In this construction, each leaf of the tree is a λ-bit string (where typically $\lambda = 128$). In contrast, we wish to share unit vectors over $\mathbb{F}_4$. Hence, we can apply the *early termination* technique from [BGI16] that shaves several levels of PRG expansions. With early termination, to obtain a $D = 2^d$-long vector over $\mathbb{F}_4$, we use a tree of depth $2D/\lambda = 2^{d-6}$ (using $\lambda = 128$) and parse each of the 128-bit leaves as a 64-tuple of $\mathbb{F}_4$-elements. This immediately yields a 64-fold runtime improvement for each of the DPFs required in the PCG construction.

We note that while other constructions share a similar blueprint to the construction of Bombar et al., and in particular also require evaluating many DPFs under-the-hood, this early termination technique does not apply to them. The reason is that in silent OT extension protocols [BCG+19b; BCG+19a; CRR21; BCG+22; RRT23], the DPFs are used to compress secret shares of $\Delta \cdot \mathbf{e}$, where Δ is a 128-bit element from a suitable extension field, and in the previous PCG construction of [BCG+20b], the OLEs can only be generated over a large field $\mathbb{F}$ (chosen equal to $|\mathbb{F}| \approx 2^\lambda$ in their implementation). As such, early termination optimization appears to apply exclusively when specializing the PCG of [BCC+23] to work over small fields.

Using a single multi-evaluation step. Computing $\langle \mathbf{a} \otimes \mathbf{a}, [\![\mathbf{e}_0 \otimes \mathbf{e}_1]\!]_b \rangle$ (for $b = 0, 1$) requires c^2 polynomial multiplications. Fast polynomial multiplication is typically done using a multi-evaluation (i.e., an FFT) followed by a local product and an interpolation (i.e., an inverse FFT).

The above produces a single OLE over $\mathbb{F}_4[\mathbb{G}]$. When the end goal is to obtain OLEs over $\mathbb{F}_4$, the result is projected back onto $\mathbb{F}_4^{3^n}$ using a multi-evaluation. In this case, we show that we can reduce the sequence multi-evaluation $\to$ interpolation $\to$ multi-evaluation down to just a single multi-evaluation step. Concretely:

- Given that $\mathbf{a}$ is a random vector of polynomials (and part of the public parameters), it can directly be generated as c random length-3^n vectors over $\mathbb{F}_4^n$, corresponding to the vectors of the multi-evaluations of $\mathbf{a}$ over all n-tuples in $(\mathbb{F}_4^\times)^n$.
- The multi-evaluation of $\mathbf{a} \otimes \mathbf{a}$ can be computed once for all using pairwise products of elements of (the multi-evaluation of) $\mathbf{a}$, and included in the public parameters.
- Computing the multi-evaluation of $\langle \mathbf{a} \otimes \mathbf{a}, [\![\mathbf{e}_0 \otimes \mathbf{e}_1]\!]_b \rangle$ amounts to computing the multi-evaluation of $[\![\mathbf{e}_0 \otimes \mathbf{e}_1]\!]_b$ followed by component-wise inner products.

It follows that after expanding the shares $[\![\mathbf{e}_0 \otimes \mathbf{e}_1]\!]_b$, the cost of PCG.Expand is then dominated by c^2 instances of a multi-evaluation (i.e., an FFT). However, upon slightly closer inspection, we observe that it actually suffices to compute $c(c+1)/2$ FFTs (since the terms $e_0^i e_1^j$ and $e_0^j e_1^i$ share the same "coefficient" $a_i a_j$ in $\langle \mathbf{a} \otimes \mathbf{a}, \mathbf{e}_0 \otimes \mathbf{e}_1 \rangle$, hence the FFT can be evaluated on terms $e_0^i e_1^j + e_0^j e_1^i$ directly).

Blazing fast FFT. Our next observation is that the FFT over the group algebra $\mathbb{F}_4[\mathbb{G}]$ is actually *extremely* efficient. Indeed, given a polynomial $P(X_1, \cdots, X_n)$, one can rewrite P as

$$P_0(X_1, \cdots, X_{n-1}) \;+\; X_n P_1(X_1, \cdots, X_{n-1}) \;+\; X_n^2 P_2(X_1, \cdots, X_{n-1}).$$

Let us denote $\mathsf{FFT}(P, n)$ the functionality that evaluates P on all n-tuples over $(\mathbb{F}_4^\times)^n$, and outputs a multi-evaluation vector $\mathbf{v}$. By the above formula, computing $\mathsf{FFT}(P, n)$ reduces to

- computing $\mathbf{v}_i \leftarrow \mathsf{FFT}(P_i, n-1)$ for each $i \in \{0, 1, 2\}$, and
- setting $\mathbf{v} \leftarrow (\mathbf{v}_0 + \mathbf{v}_1 + \mathbf{v}_2 \,||\, \mathbf{v}_0 + \theta\mathbf{v}_1 + (\theta+1)\mathbf{v}_2 \,||\, \mathbf{v}_0 + (\theta+1)\mathbf{v}_1 + \theta\mathbf{v}_2)$.

Denoting $C(n)$ the cost of running $\mathsf{FFT}(P, n)$, we therefore have $C(n) = 3 \cdot C(n-1) + \ell \cdot 3^{n-1}$, where ℓ denotes the number of vector operations (naïvely, 6 additions of vectors and 4 scalar-vector products—but some additions and products can be reused). This yields a cost of $C(n) = n \cdot \ell \cdot 3^{n-1}$, where all operations are very fast: either additions of $\mathbb{F}_4$-vectors or multiplications by θ. Looking ahead, our implementation and evaluation (Section 5.6) confirm that, even with the straightforward recursive algorithm, the FFT results in minimal overhead compared to the cost of the DPFs.[4]

Stepping back: comparison with silent OT extension. To give an intuition about the efficiency of this construction, we provide a brief comparison with constructions of silent OT extension. In short, to get (say) 3^n OTs, these constructions run $c \cdot t$ DPFs on a domain of size $3^n/t$, followed by a multiplication with a compressive mapping. In the most efficient silent OT extension protocol to date [RRT23], this compressive mapping requires computing $21 \cdot c \cdot 3^n$ XORs, followed by 3^n XORs of random size-21 subsets of the bits of the resulting vector. Due to the overhead of many random memory accesses, the cost of computing this mapping dominates the overall runtime. In contrast, we need $(c \cdot t)^2$ DPFs with domain size $3^n/t$, but get a $64\times$ speedup from the early termination optimization. The cost of our DPFs should be essentially on par with that of [RRT23]. However, the FFT cost in our construction is largely dominated by the cost of the DPFs. Therefore, we expect (and this is confirmed by our implementation) that this PCG should produce $\mathbb{F}_4$-OLEs at a much faster rate compared with the rate at which [RRT23] produces OTs. In the two-party setting, when the goal is to generate Beaver triples over $\mathbb{F}_2$, we get an additional $2\times$ speedup from the technique of Section 5.3.4, as we generate one triple from *one* $\mathbb{F}_4$-OLE (whereas [RRT23] requires two OTs). We provide an optimized implementation of our scheme and evaluate how it compares to previous works in Section 5.6. Our implementation is about $6\times$ faster than the state of the art [RRT23].

5.3.6 Distributed Seed Generation

So far, we have only discussed the cost of expanding the PCG keys $(\mathsf{k}_0, \mathsf{k}_1)$. To obtain a full-fledged secure computation protocol, we need an efficient way for the parties to securely evaluate PCG.Gen procedure in a distributed fashion. In the following, as in all previous works on PCGs [BCG+19b; BCG+19a; BCG+20b; BCG+20a; CRR21; BCG+22; BCC+23; CD23], we assume that the noise follows a regular distribution. That is, a noise vector $\mathbf{e}$ is a vector of c polynomials $(e^1, \cdots e^c)$, where each

[4]Our implementation also exploits vectorized operations to perform a batch of multiple FFTs for essentially the cost of one, which further reduces the impact of FFTs on the overall runtime.

polynomial e^i is *regular*: its coordinates are divided into t block of (approximately) equal length $3^n/t$, and it has a single nonzero coefficient in each block. For any integer h, let $[h]$ denote the set $\{1, \cdots, h\}$. The previous work of [BCG+20b] outlined the following methodology to securely distribute PCG seeds for generating D OLEs (in our context, $D = 3^n$):

- **Sampling the noise vectors.** Each party P_b generates its noise vector $\mathbf{e}_b$ locally, by sampling c t-sparse regular polynomials. We write $\mathbf{e}_b = (e_b^1, \cdots, e_b^c)$. For each $i \in [c]$, we let $(\mathsf{p}_{b,1}^i, \cdots, \mathsf{p}_{b,t}^i) \in [3^n/t]^t$ denote the t positions of the nonzero entries in e_b^i, and $(v_{b,1}^i, \cdots, v_{b,t}^i) \in \mathbb{F}_4^t$ denote the value of these nonzero coefficients.

- **Sharing the positions and values.** For every $i_0, i_1 \in [c]$, for every $j_0, j_1 \in [t]$, the parties run a distributed protocol with respective inputs $\mathsf{p}_{0,j_0}^{i_0}$ and $\mathsf{p}_{1,j_1}^{i_1}$ (i.e., the position of the j_0-th and j_1-th nonzero coefficients in $e_0^{i_0}$ and $e_1^{i_1}$, respectively) which securely computes bitwise shares of the $(j_0 + j_1)$-th nonzero coefficient of $e_0^{i_0} e_1^{i_1}$. In parallel, they also run a distributed protocol with respective inputs $v_{0,j_0}^{i_0}$ and $v_{1,j_1}^{i_1}$ (the corresponding values of the nonzero coefficients) and securely compute bitwise shares of $v_{0,j_0}^{i_0} \cdot v_{1,j_1}^{i_1}$ (the value of the $(j_0 + j_1)$-th nonzero coefficient of $e_0^{i_0} e_1^{i_1}$).

- **Distributing the DPF keys.** For every $i_0, i_1 \in [c]$, for every $j_0, j_1 \in [t]$, the parties run the Doerner-shelat protocol [Ds17] with their bitwise shares of the position and value to securely obtain DPF keys forming succinct shares of the point function $f_{\alpha,\beta}$ which evaluates to $\beta := v_{0,j_0}^{i_0} \cdot v_{1,j_1}^{i_1}$ on the index α of the $(j_0 + j_1)$-th nonzero coefficient of $e_0^{i_0} e_1^{i_1}$, and to 0 on all other inputs.

Communication-wise, the Doerner-shelat protocol requires $2 \cdot \log(D/t)$ oblivious transfers for each DPF, for a total of $2(ct)^2 \log(D/t)$ oblivious transfers. Distributing the shares of the coefficients $v_{0,j_0}^{i_0} \cdot v_{1,j_1}^{i_1}$ is relatively straightforward: it involves two OLEs over $\mathbb{F}_4$ for each of the (ct^2) coefficients. As in [BCG+20b], these OLEs can be obtained at a minimal cost by running the PCG in a "bootstrapping mode": whenever two parties use the PCG to generate D $\mathbb{F}_4$-OLEs, they can instead use a marginally larger instance to generate $D + (ct)^2$ $\mathbb{F}_4$-OLE, and store the $(ct)^2$ extra OLEs for use in the next distributed PCG seed generation.

In the work of Boyle et al. [BCG+20b], an important overhead comes from the $(ct)^2$ instances of a distributed protocol to generate bitwise shares of the noise positions: each such instance requires securely running a Boolean adder to compute, from the bit decomposition of $\mathsf{p}_{0,j_0}^{i_0}$ and $\mathsf{p}_{1,j_1}^{i_1}$, the bit decomposition of the position of the corresponding entry in $e_0^{i_0} e_1^{i_1}$. In the construction of [BCG+20b], this contributes to a large portion of the (communication and computation) overhead of the seed distribution procedure: about half of the communication, computation, and rounds of the full protocol.

An improved seed distribution from ternary DPFs. We now introduce an optimization that removes the need to distribute shares of noise positions altogether by working *directly* in the ternary basis. Our improved protocol is tailored to the setting of noise vectors with components over $\mathbb{F}_4[\mathbb{G}] = \mathbb{F}_4[X_1, \ldots, X_n]/(X_1^3 - 1, \ldots, X_n^3 - 1)$. Observe that every monomial over $\mathbb{F}_4[\mathbb{G}]$ can be written as $\mathbf{X}^{\mathsf{p}} := \prod_{i=1}^n X_i^{p_i}$, where $\mathbf{p} = (p_1, \ldots, p_n) \in \mathbb{F}_3^n$. Therefore, we can uniquely identify the position of the coefficient c_{p} of a monomial $\mathbf{X}^{\mathsf{p}}$ with the $\mathbb{F}_3$-vector $\mathsf{p} \in \mathbb{F}_3^n$. Now, consider the product of two polynomials e_0, e_1 known by P_0 and P_1, respectively. Let $\mathsf{p}_0 \in \mathbb{F}_3^n$ be the position of a nonzero entry in e_0, and $\mathsf{p}_1 \in \mathbb{F}_3^n$ be the position of a nonzero entry in e_1. Then, the corresponding nonzero entry in $e_0 \cdot e_1$ is the coefficient of the monomial $\mathbf{X}^{\mathsf{p}_0} \cdot \mathbf{X}^{\mathsf{p}_1} = \mathbf{X}^{\mathsf{p}_0 + \mathsf{p}_1 \bmod 3}$. That is, the corresponding nonzero position in $e_0 e_1$ is exactly $\mathsf{p}_0 + \mathsf{p}_1$ (where the sum is taken modulo 3). In other words, the two parties *already hold shares* of the noise position in $e_0 e_1$—but over the ternary basis!

Unfortunately, the Doerner-shelat protocol requires the parties to hold binary shares of the position, because its binary decomposition corresponds to the path from the root to the leaf in the (binary) GGM tree underlying the DPF construction of [BGI16; BGI15]. To remedy this situation, we modify the underlying DPF construction to use a *ternary tree*. That is, the full tree is obtained by computing the three children of a node by evaluating a length-tripling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{3\lambda}$ on the node value. Adapting the DPF construction of [BGI15; BGI16] to this setting is relatively simple (though the security analysis becomes slightly more tedious, especially when adapting the Doerner-shelat protocol to work over a ternary basis), and requires increasing the number of correction words from 1 to 3 per level of the tree.[5] With this change, the path to a leaf is given directly by the leaf position written as a $\mathbb{F}_3$-vector. To securely generate the keys of this modified DPF, we adapt the Doerner-shelat protocol. Our adaptation requires two 1-out-of-3 oblivious transfers per level (instead of two 1-out-of-2 OTs as in [Ds17]), for the $\log_3(D/t)$ levels of the ternary DPF tree. In summary, we obtain a distributed seed generation protocol with the following pros-and-cons when compared to the original approach of [BCG+20b]:

+ The parties "natively" hold shares of the nonzero positions and do not have to run a secure protocol to compute them. In the protocol of Boyle et al. [BCG+20b], this step required $2(ct)^2 \cdot \log(D/t)$ oblivious transfers in $\log(D/t)$ rounds (i.e., half of the total number of rounds and OTs).

− The modified Doerner-shelat requires $2(ct)^2 \log_3(D/t)$ 1-out-of-3 OTs of 3λ-bit strings instead of $2(ct)^2 \log_2(D/t)$ 1-out-of-2 OTs of 2λ-bit strings, which represents slightly more communication and computation.

− Due to the use of a ternary DPF, which has more correction words, the PCG seed size is slightly increased, by a factor ≈ 1.5.

+ Expanding the PCG seeds becomes about 20% faster because the total number of PRG evaluations is reduced when computing a full ternary tree compared to a full binary tree with a similar number of leaves.

+ The number of rounds of the Doerner-shelat protocol is also reduced, from $\log_2(D/t)$ to $\log_3(D/t)$, by having a more shallow tree.

5.4 A Fast PCG for $\mathbb{F}_4$-OLEs

In this section, we present the construction of QA-SD$_{\mathsf{OLE}}$ over $\mathbb{F}_4$ (Figure 5.3) following optimizations via early termination and fast evaluation of polynomials for FFT. We first recall how to construct PCG-based OLE from the QA-SD assumption as below.

5.4.1 PCGs from QA-SD Assumption

In this section, we recall the construction of PCGs from the *Quasi-Abelian Syndrome Decoding* assumption (QA-SD) which was introduced in [BCC+23], and properly defined in Chapter 2. We start with a short overview of the PCG construction over $\mathbb{F}_q$ of Bombar et al. [BCC+23]. Let

[5]Unfortunately, in the ternary tree construction, using the optimization described in [BGI16] for removing one extra correction word does not immediately apply. We leave open the problem of finding a similar optimization in the ternary case.

$\mathcal{R} = \mathbb{F}_\mathsf{q}[\mathbb{G}] = \left\{ \sum_{g \in \mathbb{G}} a_g g \mid a_g \in \mathbb{F}_\mathsf{q} \right\}$, with $\mathbb{G}$ an abelian group. We refer to $\mathcal{R}_t$ as the set of ring elements of $\mathcal{R}$ of weight at most t. The goal is to construct a PCG that would achieve the functionality described in Figure 5.1.

Figure 5.1: Functionality $\mathsf{QA\text{-}SD_{OLE-Setup}}$

PARAMETERS: Security parameter 1^λ, $\mathsf{PCG_{OLE}} = (\mathsf{PCG_{OLE}.Gen}, \mathsf{PCG_{OLE}.Expand})$ as per Figure 5.2.

FUNCTIONALITY:

```
1: Sample $(k_0, k_1) \leftarrow \mathsf{PCG_{OLE}.Gen}(1^\lambda)$.
2: Output $k_\sigma$ to party $P_\sigma$ for $\sigma \in \{0, 1\}$.
```

The protocol is described in Figure 5.2. The goal of the OLE correlation is to give the two parties a pseudorandom $x_\sigma \in \mathcal{R}$, as well as an additive sharing of the product $x_0 \cdot x_1$. To achieve this, the authors constructed the framework on the Quasi-Abelian Syndrome Decoding assumption. The players first have access to a vector $\mathbf{a} = (1, a_1, \cdots, a_{c-1})$ of elements in $\mathcal{R}$, publicly. Taking advantage of the canonical notion of the sparseness of $\mathcal{R}$, players can define $x_\sigma = \langle \mathbf{a}, \mathbf{e_\sigma} \rangle$, where $e_\sigma = (e_\sigma^0, \cdots, e_\sigma^{c-1})$ is a vector of t-sparse elements of $\mathcal{R}$, for a given t. Because of the QA-SD assumption, x_σ is pseudorandom. Giving the parties additive sharing of $x_0 \cdot x_1$ can be achieved via Function Secret Sharing. Indeed, $x_0 \cdot x_1 = \langle \mathbf{a}, \mathbf{e_0} \rangle + \langle \mathbf{a}, \mathbf{e_1} \rangle$ can be fully expressed via the elements in $(\mathbf{e_0} \otimes \mathbf{e_1})$, and the public elements in the expression $\mathbf{a} \otimes \mathbf{a}$. Therefore we want to obtain an additive sharing of $(\mathbf{e_0} \otimes \mathbf{e_1})$. Because each e_σ^i are t-sparse element in $\mathcal{R}$, all the products $e_0^i \cdot e_1^j$ are t^2-sparse element. Therefore they can easily be concisely shared using t^2 Single Point Function Secret Sharing (SPFSS). This enables the parties to obtain a short seed k_σ which contains their FSS keys to obtain the full evaluation and recover the additive sharing of $\mathbf{e_0} \otimes \mathbf{e_1}$, as well as the descriptions of $\mathbf{e_\sigma}$. Later on, the parties can use their seed to recover x_σ and their additive share z_σ of $x_0 \cdot x_1$.

Theorem 5.4.1 ([BCC+23])**.** *Let $\mathbb{G}$ be an Abelian group. Assume that* SPFSS *is a secure* FSS *scheme for sums of point functions and that the* $\mathsf{QA\text{-}SD}(q, c, t, \mathbb{G})$ *assumption holds. Then there exists a generic scheme to construct a* PCG *to produce one* OLE *correlation (described on Figure 5.2). If the* SPFSS *is based on a* PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ *via the* PRG*-based construction from [BGI16], we obtain:*

- *Each party's seed has maximum size around :* $(ct)^2 \cdot ((\log |\mathbb{G}| - \log t + 1) \cdot (\lambda + 2) + \lambda + \log q) + ct(\log |\mathbb{G}| + \log q)$ *bits.*
- *The computation of* Expand *can be done with at most* $(2 + \lfloor (\log q)/\lambda \rfloor)|\mathbb{G}|c^2 t$ PRG *operations, and* $O(c^2 |\mathbb{G}| \log |\mathbb{G}|)$ *operations in* $\mathbb{F}_\mathsf{q}$.

5.4.2 PCGs over $\mathbb{F}_4$ from QA-SD Assumption

The general description of the framework based on it can be found in Section 5.4.1.

In [BCC+23], the authors point out that their $\mathsf{QA\text{-}SD_{OLE}}$ construction is the first to produce a large number of OLE correlations over $\mathbb{F}_\mathsf{q}$, for any $q \geq 3$. They propose using $\mathbb{G} = \prod_{i=1}^n \mathbb{Z}/(q-1)\mathbb{Z}$, $q \geq 3$. The direct consequence of this is that $\mathbb{F}_\mathsf{q}[\mathbb{G}] \simeq \mathbb{F}_\mathsf{q}[X_1, \ldots, X_n]/(X_1^{q-1} - 1, \ldots, X_n^{q-1} - 1) \simeq \prod_{i=1}^D \mathbb{F}_\mathsf{q}$, where the last isomorphism equivalence comes from the Chinese Remainder Theorem. Above, $D = (q-1)^n$ is the number of elements in the group, and the number of OLE's we can get over $\mathbb{F}_\mathsf{q}$ by applying this isomorphism. Looking closely, we instantiate our particular PCG over $\mathcal{R} \simeq \mathbb{F}_4[X_1, \ldots, X_n]/(X_1^3 - 1, \ldots, X_n^3 - 1)$, by setting $q = 4$. At the end of the protocol $\mathsf{QA\text{-}SD_{OLE}}$,

Figure 5.2: General construction of $\mathsf{QA\text{-}SD_{OLE}}$

Parameters: Security parameter λ, noise weight $t = t(\lambda)$, compression factor $c \geq 2$, $\mathbb{G}$ a finite abelian group, $\mathcal{R} = \mathbb{F}_q[\mathbb{G}]$. An FSS scheme (SPFSS.Gen,SPFSS.FullEval) for sums of t^2 point functions, with domain $[0 \ldots |\mathbb{G}|)$ and range $\mathbb{F}_q$.

Public Input: $c-1$ random ring elements $a_1, \ldots, a_{c-1} \in \mathcal{R}$.

```
PCG.Gen($1^\lambda$):
1: foreach $\sigma \in \{0,1\},\ i \in [0 \ldots c)$:
    1.1: $\mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}} \leftarrow (p^i_{\sigma,1}, \cdots, p^i_{\sigma,t})_{p^i_{\sigma,j} \in \mathbb{G}}$ and $\mathbf{v}_{\boldsymbol{\sigma}}^{\boldsymbol{i}} \leftarrow (\mathbb{F}_q^\times)^t$.
2: foreach $i, j \in [0 \ldots c)$:
    2.1: Sample FSS keys $(K_0^{i,j}, K_1^{i,j}) \xleftarrow{\$}$ SPFSS.Gen$(1^\lambda, 1^n, \mathbf{p}_{\mathbf{0}}^{\boldsymbol{i}} \otimes \mathbf{p}_{\mathbf{1}}^{\boldsymbol{j}}, \mathbf{v}_{\mathbf{0}}^{\boldsymbol{i}} \otimes \mathbf{v}_{\mathbf{1}}^{\boldsymbol{j}})$.
3: Let $\mathsf{k}_\sigma = ((K_\sigma^{i,j})_{i,j \in [0 \ldots c)}, (\mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}, \mathbf{v}_{\boldsymbol{\sigma}}^{\boldsymbol{i}})_{i \in [0 \ldots c)})$.
4: Output $(\mathsf{k}_0, \mathsf{k}_1)$.

PCG.Expand($\sigma, \mathsf{k}_\sigma$):
1: Parse $\mathsf{k}_\sigma$ as $((K_\sigma^{i,j})_{i,j \in [0 \ldots c)}, (\mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}, \mathbf{v}_{\boldsymbol{\sigma}}^{\boldsymbol{i}})_{i \in [0 \ldots c)})$.
2: foreach $i \in [0 \ldots c)$:
    2.1: Define the element of $\mathcal{R}_t$,
         $$e_\sigma^i = \sum_{j \in [0 \ldots t)} \mathbf{b}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}[j] \cdot \mathbf{p}_{\boldsymbol{\sigma}}^{\boldsymbol{i}}[j].$$
3: Compute $x_\sigma = \langle \mathbf{a}, \mathbf{e}_{\boldsymbol{\sigma}} \rangle$, where $\mathbf{a} = (1, a_1, \cdots, a_{c-1}), \mathbf{e}_{\boldsymbol{\sigma}} = (e_\sigma^0, \cdots, e_\sigma^{c-1})$.
4: foreach $i, j \in [0 \ldots c)$:
    4.1: Compute $u_{\sigma, i+cj} \leftarrow$ SPFSS.FullEval$(\sigma, K_\sigma^{i,j})$ and view it as a $c^2$ vector $\mathbf{u}_{\boldsymbol{\sigma}}$ of elements in $\mathcal{R}_{t^2}$.
5: Compute $z_\sigma = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_{\boldsymbol{\sigma}} \rangle$.
6: Output $x_\sigma, z_\sigma$.
```

the parties obtain one OLE over $\mathcal{R}$, the parties obtain one OLE over $\mathcal{R}$; a general description of the construction is given in Figure 5.2.

Let us denote (x_σ, z_σ) the output of party σ. To obtain many OLE's over $\mathbb{F}_4$, the parties have to evaluate $x_\sigma, z_\sigma \in \mathbb{F}_4[X_1, \ldots, X_n]/(X_1^3 - 1, \ldots, X_n^3 - 1)$ over the full domain $(\mathbb{F}_4^\times)^n$. The standard approach is to use a Fast Fourier Evaluation to efficiently obtain this result. Here, we remark that, in our group algebra, fast multiplication *also* requires FFT, first in a multi-evaluation form, and then in the interpolation form. Therefore, doing the interpolation again is wasteful as in the end we will evaluate again after interpolating. As such, we can avoid the intermediate steps of multi-evaluation-then-interpolation and work directly with the evaluations, without coming back to $\mathcal{R}$. That is, we do not construct the polynomials x_σ, z_σ over $\mathbb{F}_4[X_1, \ldots, X_n]/(X_1^3 - 1, \ldots, X_n^3 - 1)$ but instead, we focus directly on the polynomials evaluations.

Let $\mathsf{Eval}_n(f) = \{f(x_1, \ldots, x_n), (x_1, \ldots, x_n) \in \{1, \theta, \theta + 1\}^n\}$ be the set of all the possible evaluations. Instead of giving the parties the description of the coefficients of the polynomials $a_i \in \mathbf{a}$, we can give them the vectors of all the evaluations of all the polynomials, that is giving them $\mathsf{Eval}_n(a_i)$, for all i. Because we can write $x_\sigma = e_\sigma^0 + e_\sigma^1 a_1 + \cdots + e_\sigma^{c-1} a_{c-1}$, it follows that all the evaluations of x_σ can be obtained from $\mathsf{Eval}_n(e_\sigma^i)$ and $\mathsf{Eval}_n(a_i)$. All that remains is to evaluate the e_σ^i polynomials. They are sparse polynomials, and therefore their evaluations can be computed very efficiently i.e., if the polynomials have t non-zero coefficients, then the cost of the evaluation is linear in $t \cdot 3^n$. As a result, we can obtain $\mathsf{Eval}_n(x_\sigma)$ for a cost linear in 3^n.

The computation of $\mathsf{Eval}_n(z_\sigma)$ is a little trickier. As mentioned above, $x_0 \cdot x_1$ can be seen as a function of degree 2 in $(\mathbf{e_0}, \mathbf{e_1})$, with constant coefficients depending solely from $\mathbf{a} \otimes \mathbf{a}$. Because $\mathsf{Eval}_n(a_i)$ is already given to the parties, the evaluation of the coefficient from $\mathbf{a} \otimes \mathbf{a}$ can be obtained using only c^2 multiplications. It remains to compute the evaluations of the additive shares of the polynomials $e_0^i \cdot e_1^j$. There are c^2 such polynomials shared among the parties, and we can view each share as a random polynomial. Therefore, each party has to compute the evaluation of c^2 random polynomials. This is a crucial part of the scheme and we devote the next section to it. Figure 5.3 represents the PCG framework of [BCC+23] tailored to our setting, its correctness and security are implied by Theorem 5.4.1.

As an optimization, we use a regular noise distribution which we show in Remark 5.4.1 allows us to argue a regular noise distribution on the resulting polynomial $\mathbf{e_0} \otimes \mathbf{e_1}$. In particular, the outer sum of regular noise positions results in t noise positions per block.

Remark 5.4.1. *Let $t = 3^k$ be a power of 3, and let $\mathcal{R} = \mathbb{F}_4[\mathbb{G}] = \mathbb{F}_4[X_1, \dots, X_n]/(X_1^3 - 1, \dots, X_n^3 - 1)$. Let $\mathbf{e_0}$ and $\mathbf{e_1}$ be sampled from a t-regular noise distribution over $\mathcal{R}$. In other words, the coordinates of $\mathbf{e_i}$ can be divided into t consecutive blocks $B_0, \dots, B_{t-1}$ of size $3^n/t$, each block having a single nonzero coordinate. More precisely, considering the lexicographic ordering of the monomials, and since $t = 3^k$, block B_i is formed by all monomials $\mathbf{X}^\mathbf{p}$ such that the first k coordinates of $\mathbf{p}$ represent the ternary decomposition of the integer i (over k trits). For example, if $n = 4$ and $t = 9$, the $3^4 = 81$ monomials are split into 9 blocks $B_0, \dots, B_8$ of size 9, and a monomial $\mathbf{X}^\mathbf{p}$ lies in B_6 if and only if $\mathbf{p}$ is of the form $(2, 0, \star, \star)$ with $\star \in \{0, 1, 2\}$, where $[2\|0]$ is the ternary decomposition of the integer 6.*

We now show that the product $\mathbf{e} = \mathbf{e}_0 \cdot \mathbf{e}_1$ has at most t nonzero monomials in each block.[6] *Indeed, let $i \in \{0, \dots, 3^k - 1\}$ and let $\mathbf{X}^\mathbf{p}$ be a monomial appearing in $\mathbf{e}$ with a nonzero coefficient. In particular, the first k entries of $\mathbf{p}$ can be parsed as the ternary decomposition of i, which we denote by $[i]_3$. It is clear that $\mathbf{X}^\mathbf{p}$ is of the form $\mathbf{X}^{\mathbf{p}_0 + \mathbf{p}_1}$ where $\mathbf{p}_0$ (resp. $\mathbf{p}_1$) identifies one of the t nonzero monomials in $\mathbf{e}_0$ (resp. $\mathbf{e}_1$), and the sum is taken modulo 3 component-wise. In particular, there are at most t^2 such monomials, and for each nonzero monomial $\mathbf{X}^{\mathbf{p}_0}$ of $\mathbf{e}_0$, with first k entries $[i_0]_3$, there corresponds at most one nonzero monomial in $\mathbf{e}_1$ contributing to $\mathbf{X}^\mathbf{p}$, namely $\mathbf{X}^{\mathbf{p} - \mathbf{p}_0}$.*[7] *In other words, the monomial $\mathbf{X}^\mathbf{p}$ can be produced by at most t possible pairs of monomials $(\mathbf{X}^{\mathbf{p}_0}, \mathbf{X}^{\mathbf{p}_1})$, whose first k entries are $([i_0]_3, [i]_3 - [i_0]_3)$, with i_0 ranging over $\{0, \dots, t-1\}$.*

Example. *Let $n = 3$ and $t = 3$. Set $\mathbf{e}_0 := X_3^2 + X_1X_2X_3 + X_1^2$ (which corresponds to positions $(0,0,2), (1,1,1)$, and $(2,0,0)$) and $\mathbf{e}_1 := 1 + X_1 + X_1^2$ (which corresponds to positions $(0,0,0), (1,0,0)$, and $(2,0,0)$). Then,*

$$\mathbf{e}_0 \cdot \mathbf{e}_1 = \underbrace{(1 + X_3^2 + X_2X_3)}_{\in B_0} + \underbrace{(X_1 + X_1X_3^2 + X_1X_2X_3)}_{\in B_1} + \underbrace{(X_1^2 + X_1^2X_3^2 + X_1^2X_2X_3)}_{\in B_2}.$$

Proposition 5.4.1. *Let $\mathcal{R} = \mathbb{F}_4[\mathbb{G}] = \mathbb{F}_4[X_1, \dots, X_n]/(X_1^3 - 1, \dots, X_n^3 - 1)$ where $\mathbb{G} = \prod_{i=1}^n \mathbb{Z}/3\mathbb{Z}$ is an Abelian group. Assume that* SPFSS *is a secure* FSS *scheme for sums of point functions and that the* $\mathsf{QA\text{-}SD}(q, c, t, \mathbb{G})$ *assumption holds for regular noise distribution. Then there exists a generic scheme to*

[6] This crucially relies on the fact that since t is a power of 3, we can uniquely identify the block corresponding to a given monomial by looking at the first k entries of its exponent. When t is not a power of 3, this is not true anymore.

[7] Note that the corresponding monomial $\mathbf{X}^{\mathbf{p}_1}$ might not appear in $\mathbf{e}_1$.

construct a PCG *to produce one* OLE *correlation (described on Figure 5.3). If the* SPFSS *is based on a* PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ *via the* PRG*-based construction from [BGI16], we obtain:*

- *Each party's seed has maximum size around:* $(c \cdot t)^2 \cdot ((n \cdot \log(3) - \log t + 1) \cdot (\lambda + 2) + \lambda + 2) + c \cdot t \cdot (n \cdot \log(3) + 2)$ *bits.*
- *The computation of* Expand *can be done with at most* $\log(3) \cdot (2 + \lfloor (2)/\lambda \rfloor) \cdot n \cdot c^2 \cdot t$ PRG *operations, and* $O(n \cdot \log(3) \cdot c^2 \cdot 3^n)$ *operations in* $\mathbb{F}_4$.

The proof follows immediately from Theorem 5.4.1 and the analysis of [BCC+23].

5.4.3 Optimizations

Optimizing on FSS evaluation via early termination

We remark that we can use a very simple trick that enables the parties to obtain the evaluation of their MPFSS shares 64 times faster than with the standard construction (*and* at a slight reduction in communication). The trick comes from the fact that the standard construction of the DPF based on the GGM tree implies that each leaf is of size $\lambda = 128$ bits. It was pointed out in [BGI16] that we can consider *early termination* in the case of small outputs. In our case, we would like a single leaf to encode a value in $\mathbb{F}_4$. This only requires 2 bits instead of the 128 bits we get as output, making the naïve evaluation "waste" 126 bits of the output. Instead, we can avoid wasting computation by truncating the tree 6 levels earlier and setting the value of the new 128-bit leaf on the special path to encode a unit vector consisting of zeroes except on the exact 2 bits where it equals to the correct value of $\mathbb{F}_4$ element. This essentially involves "hard-coding" the end of the path into the leaf directly, as illustrated in Section 5.4.3. Using this idea, we reduce the computational cost of evaluating the DPF by $64\times$ and reduce the communication costs (key size of the DPF) by roughly $6 \cdot 128$ bits [BGI16]. This simple trick was initially introduced in the context of PIR applications [BGI16], but could not be applied to prior PCG constructions until now since all PCG constructions (except for the recent PCG construction of Bombar et al. [BCC+23]) required the DPF output to be encode elements of a large field. Similarly, in silent OT extension protocols [BCG+19b; BCG+19a; CRR21; BCG+22; RRT23], which are also bottlenecked by DPF evaluations, this optimization could not be applied because there, the DPF is used to output "authenticated" shares of a (potentially small) field element with a (large) MAC, which requires the leaves to encode 128 bit output value.

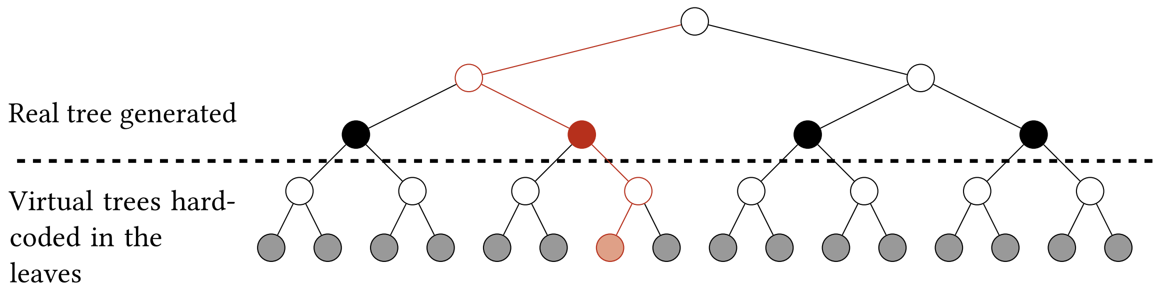

Figure 5.4: Early termination example in the case we truncate only two steps earlier. Solid black nodes represent "zero" leaves, whereas solid red leaves can take on any value.

Fast evaluation over $\mathbb{F}_4[X_1, \ldots, X_n]/(X_1^3 - 1, \ldots, X_n^3 - 1)$

The high-level idea. Given a polynomial P with n variables, the party wants to compute $\mathsf{Eval}_n(P)$, that is, to evaluate P over $\left(\mathbb{F}_4^\times\right)^n$ where $\mathbb{F}_4^\times = \{1, \theta, \theta + 1\}$. Here, we adapt the standard divide-

Figure 5.3: $\mathsf{QA\text{-}SD_{OLE}}$ **for** $\mathbb{F}_4$**OLEAGE over** $\mathcal{R}$ **from evaluations of functions**

Parameters: Noise weight $t = t(\lambda)$, compression factor c, ring $\mathcal{R} = \mathbb{F}_4[X_1, \ldots, X_n]/(X_1^3 - 1, \ldots, X_n^3 - 1)$. A SPFSS scheme $\mathsf{SPFSS} = (\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of t^2 point functions, with domain $[0 \ldots 3^n)$ and range $\mathbb{F}_4$.

Public Input: $c-1$ vectors of length 3^n over $\mathbb{F}_4$, corresponding to the result of $\mathsf{Eval}_n(a_i)$, for uniformly random $a_1, \cdots, a_{c-1} \in \mathcal{R}$, therefore the full evaluation of the c elements a_i.

```
PCG.Gen($1^\lambda$):
1: foreach $\sigma \in \{0,1\},\ i \in [0 \ldots c)$:
    1.1: Sample random $\mathbf{p}^{\boldsymbol{i}}_{\boldsymbol{\sigma}} \leftarrow \{(\mathbf{p}^{\boldsymbol{i}}_{\boldsymbol{\sigma,1}}, \ldots, \mathbf{p}^{\boldsymbol{i}}_{\boldsymbol{\sigma,t}}) \mid \mathbf{p}^{\boldsymbol{i}}_{\boldsymbol{\sigma,j}} \in \mathbb{F}_3^n\}$, and $\mathbf{v}^{\boldsymbol{i}}_{\boldsymbol{\sigma}} \leftarrow (\mathbb{F}_4^\times)^t$.
         ▷ [Optimization]: $\mathbf{p}^{i}_{\sigma}$ can be sampled from regular noise distribution. See Remark 5.4.1.
2: foreach $i, j \in [0 \ldots c)$:
    2.1: Sample FSS keys $(K_0^{i,j}, K_1^{i,j}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, 1^n, \mathbf{p}^{\boldsymbol{i}}_{\mathbf{0}} \boxplus \mathbf{p}^{\boldsymbol{j}}_{\mathbf{1}}, \mathbf{v}^{\boldsymbol{i}}_{\mathbf{0}} \otimes \mathbf{v}^{\boldsymbol{j}}_{\mathbf{1}})$.
         ▷ If using regular noise as an optimization, then
         ▷ SPFSS is for the sum of $t$ point functions with domain $[0, \ldots, 3^n/t)$.
3: Let $\mathsf{k}_\sigma = ((K_\sigma^{i,j})_{i,j \in [0 \ldots c)}, (\mathbf{p}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}, \mathbf{v}^{\boldsymbol{i}}_{\boldsymbol{\sigma}})_{i \in [0 \ldots c)})$.
4: Output $(\mathsf{k}_0, \mathsf{k}_1)$.

PCG.Expand($\sigma, \mathsf{k}_\sigma$):
1: Parse $\mathsf{k}_\sigma$ as $((K_\sigma^{i,j})_{i,j \in [0 \ldots c)}, (\mathbf{p}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}, \mathbf{v}^{\boldsymbol{i}}_{\boldsymbol{\sigma}})_{i \in [0 \ldots c)})$.
2: foreach $i \in [0 \ldots c)$:
    2.1: Define over $\mathbb{F}_4$ the polynomial:
              $$e_\sigma^i(X) = \sum_{j \in [0 \ldots t)} \mathbf{v}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}[j] \cdot \mathbf{X}^{\mathbf{p}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}[j]}.$$
    2.2: Compute $\mathsf{Eval}_n(e_\sigma^i)$.
3: Compute $x_\sigma = \langle \mathbf{a}, \mathbf{e}_{\boldsymbol{\sigma}} \rangle$, where $\mathbf{a} = (1, a_1, \cdots, a_{c-1})$, $\mathbf{e}_{\boldsymbol{\sigma}} = (e_\sigma^0, \cdots, e_\sigma^{c-1})$.
4: From $\mathsf{Eval}_n(e_\sigma^i)$ and $\mathsf{Eval}_n(a_i)$, compute $\mathsf{Eval}_n(x_\sigma)$.
5: foreach $i, j \in [0 \ldots c)$,
    5.1: Compute $u_{\sigma, i+cj} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{i,j})$ and view it as a $c^2$ vector $\mathbf{u}_{\boldsymbol{\sigma}}$ of elements in $\mathcal{R}$.
6: foreach $j \in [0 \ldots c^2)$:
    6.1: Compute $\mathsf{Eval}_n(u_{\sigma,j})$.
         ▷ [Optimization]: only need to perform $c(c+1)/2$ FFTs, see Section 5.3.5.
7: Compute $\mathsf{Eval}_n(z_\sigma)$, with $z_\sigma = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_{\boldsymbol{\sigma}} \rangle$.
8: Output $(\mathsf{Eval}_n(x_\sigma), \mathsf{Eval}_n(z_\sigma))$.
```

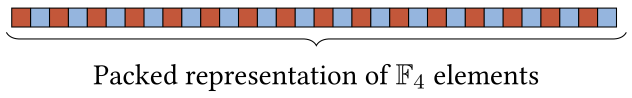

Figure 5.5: Representation of a vector of $\mathbb{F}_4$ elements. Red blocks represent the high-order bits while the blue blocks represent low-order bits.

and-conquer style algorithm to our case, see for example the seminal work of [CT65]. Remark that

$$P(X_1, \ldots, X_n) = P_0(X_1, \ldots, X_{n-1}) + X_n P_1(X_1, \ldots, X_{n-1}) + X_n^2 P_2(X_1, \ldots, X_{n-1}).$$

Instead of classically dividing our problem into 2 sub-problems, we divide it into 3 sub-problems. This is a ternary generalization of a standard FFT algorithm adapted to our case. Then,

$$\mathsf{Eval}_n(P) = \mathsf{Eval}_{n-1}(P_0) \cup X_n \mathsf{Eval}_{n-1}(P_1) \cup X_n^2 \mathsf{Eval}_{n-1}(P_2). \tag{5.1}$$

Denote by $\mathcal{C}(\mathsf{Eval}_n(P))$ the number of operations carried out to obtain all the 3^n evaluations on the set $\mathbb{F}_4^\times$. Then we have $\mathcal{C}(\mathsf{Eval}_n(P)) = 3\mathcal{C}(\mathsf{Eval}_n(P)) + 2 \cdot 3^n$, which leads us to $\mathcal{C}(\mathsf{Eval}_n(P)) = 4 \cdot n \cdot 3^n$. The concrete number of additions or multiplications is $2 \cdot n \cdot 3^n$. This quick back-of-the-envelope calculation captures the essence of the technique, even if it does not accurately count the cost of the various operations and does not take into account what is implemented in practice. We now turn to a concrete implementation of this idea:

Concrete implementation. An element of $\mathbb{F}_4$ has a direct canonical representation using 2 bits. Given an element $x \in \mathbb{F}_4$, we write $x(0)$ and $x(1)$ to denote the $\mathbb{F}_2$-coefficients of x viewed as a polynomial over $\mathbb{F}_2[X]/(X^2 + X + 1)$; that is, $x = x(0) + \theta \cdot x(1)$. Using a given machine word of 64 bits we represent a vector of size 32 over $\mathbb{F}_4$, such that the even indexed bits are high order and the odd indexed bits are low order.

This is illustrated in Figure 5.5.

As stated before, we use a recursive algorithm to compute all the evaluations, displayed in algorithm Figure 5.6. We considered using a non-recursive approach but no significant efficiency gains were observed, so we instead decided to use the recursive algorithm due to its conceptual simplicity.

Actual cost of the computation. A step in the algorithm of Figure 5.6 is to evaluate a polynomial of degree 2, with coefficient in $\mathbb{F}_4$, for the values $\{1, \theta, \theta + 1\}$. Let the polynomial be $a + bX_i + cX_i^2$.

- in the case $X_i = 1$, then the evaluation of the polynomial becomes $a + b + c$.
- in the case $X_i = \theta$, the evaluation becomes $(a + c) + \theta \cdot (b + c)$.
- in the case $X_i = \theta + 1$, the evaluation becomes $(a + b) + \theta \cdot (b + c)$.

Note that we want to compute all the different evaluations, and therefore we can try to reduce the overall costs by reusing several of the intermediate calculations. We can obtain the three evaluations via the following steps: (1)compute $a + b, a + c, b + c$; (2) compute $\theta \cdot (b + c)$; (3) compute $a + b + c$; (4) Compute $(a + c) + \theta \cdot (b + c)$, and $(a + b) + \theta \cdot (b + c)$. Therefore, we count 12 classical bit-by-bit XOR over $\mathbb{F}_2$, and a multiplication by θ to obtain the three needed evaluations of the polynomial.

Taking advantage of the computer words.

Today's processors offer XOR operations for machine words of size 64 bits.

We take advantage of this parallelism to run multiple FFTs in parallel with a small overhead compared to running a single FFT. With 64-bit machine words, we can perform up to 32 FFT in

Figure 5.6: Fast-Evaluation algorithm

```
PARAMETERS: $n > 0$ an integer, $P \in \mathbb{F}_3[X_1, \cdots, X_n]/(X_1^3 - 1, \cdots, X_n^3 - 1)$, a polynomial with $n$ variables.
FastEval$(n, P)$:
1: if $n = 1$ then
    1.1: return $\{P(1), P(\theta), P(\theta + 1)\}$
2: else
    2.1: Write $P(X_1, \cdots, X_n) = P_0(X_1, \cdots, X_{n-1}) + X_n P_1(X_1, \cdots, X_{n-1}) + X_n^2 P_2(X_1, \cdots, X_{n-1})$.
    2.2: $S := \{\}$.
    2.3: $\forall i \in \{0, 1, 2\}, S_i \leftarrow$ FastEval$(n - 1, P_i)$.
    2.4: foreach $i \in [|S_0|]$:
        2.4.1: $f_j(X) := S_0[j] + S_1[j]X + S_2[j]X^2$.
        2.4.2: $S \leftarrow S \cup \{f_j(1), f_j(\theta), f_j(\theta + 1)\}$.
    2.5: return $S$.
```

parallel. We pack the c^2 FFTs required by our PCG as follows: we let each machine word contain a single coefficient of the same monomial for each of the c^2 polynomials that we are trying to compute. This saves a factor of c^2, at no extra cost.[8] Therefore, the cost of the evaluation of a single polynomial being of $16n \cdot 3^{n-1}$ XOR, the optimization entails the cost of obtaining the full evaluation of the c^2 polynomials to be $16\lceil c^2/64 \rceil n \cdot 3^{n-1}$.

5.5 Distributed Seed Generation

In this section, we build a distributed point function that works over ternary indices. This generalization of the standard DPF construction allows us to cleanly work on a ternary basis. In particular, this makes the distributed seed generation protocol for our PCG construction in Figure 5.2 much more efficient by avoiding the use of expensive secure binary decomposition protocols when working over ternary secret shares.

5.5.1 A Ternary Distributed Point Function

In prior constructions [GI14; BGI15; BGI16], the domain of DPF was set to $\mathcal{X} = \{0, 1\}^n$. In contrast, we will use a ternary domain $\{0, 1, 2\}^n$. While this change may appear conceptually straightforward, the constructions of [GI14; BGI15] do not immediately generalize to non-binary input domains. We therefore construct a *ternary* DPF using the main ideas behind the two state-of-the-art constructions [BGI15; BGI16].

Definition 5.5.1 (Random Distributed Point Function (rDPF)). *We say a* DPF *scheme is a* random *DPF* (rDPF) *scheme if* DPF.Gen *does not take the parameter β as input, and the output value at index α is a secret share of a value $(s \| 1) \in \{0, 1\}^{\lambda+1}$, where s pseudorandom conditioned on K_σ, for $\sigma \in \{0, 1\}$.*

[8] In practice, using larger machine words has an impact by increasing stack usage, but this is only observed when performing an FFT over very large polynomials.

Lemma 5.5.1 (Adapted from [BGI16]). *Any random DPF scheme with output group $\{0,1\}^{\lambda+1}$ can be transformed into a DPF scheme for any choice of $\beta \in \mathbb{G}$ a the cost of increasing the key size by* $\log|\mathbb{G}|$ *bits.*

Remark 5.5.1. *Looking ahead to Section 5.5.2, using a random DPF makes our protocols and analysis simpler. In particular, describing a distributed key generation protocol for a random DPF eliminates edge cases associated with the output value β. Separately, we show how to generate an "output correction word" that can be used to go from the $s\|1$ output of an rDPF to an arbitrary output β.*

We present our construction for a ternary rDPF in Figure 5.7 and analyze security in Proposition 5.5.1. in Figure 5.7, our ternary DPF construction is with full-evaluation optimization where all s values are $\{0,1\}^\lambda$ bit strings and t values are bits. Superscripts 0 and 1 represent a party identifier which we write as $\sigma \in \{0,1\}$ when referring to a value held by party $\sigma \in \{0,1\}$. The construction follows a similar template to the DPF construction of [BGI15].

Proposition 5.5.1 (Ternary rDPF security). *Figure 5.7 satisfies the correctness and security properties of Definition 5.5.1.*

Proof. We prove correctness and security in turn.

Correctness. Fix an index $\alpha = (\alpha_1, \ldots, \alpha_n)$ in a ternary basis. Consider the ternary tree consisting of 3^n nodes. Let $L^\sigma_{i,j}$ be any node label at depth i and index $j \in [3^i]$ as computed by party σ. Define $L_{i,j} = L^0_{i,j} \oplus L^1_{i,j}$ and $\ell_i = 3^i\alpha_i + 3^{i-1}\alpha_{i-1} + \cdots + \alpha_1$. To prove correctness, we start by showing that the following two invariants are maintained throughout the tree:

1. For all node labels $L_{i,j}$ where $j \neq \ell_i$, $L_{i,j} = 0^\lambda\|0$.

2. For all node labels $L_{i,j}$ where $j = \ell_i$, $L_{i,j} = L'\|1$ where $L' \in \{0,1\}^\lambda \setminus \{0^\lambda\}$.

If we can show that the two invariants are maintained, it immediately follows that only one path along the tree contains non-zero labels. Once we've shown this, we can argue why the output at the non-zero label at the leaf is equal to β.

We first prove that all child node labels $(L_{i,3j}, L_{i,3j+1}, L_{i,3j+2})$ of parent nodes with labels $L_{i-1,j} = 0^\lambda\|0$ are also zero. That is, $(L_{i,3j}, L_{i,3j+1}, L_{i,3j+2}) = (0^{\lambda+1})^3$. By definition, $L_{i-1,j} = L^0_{i-1,j} \oplus L^1_{i-1,j}$, where $L^\sigma_{i-1,j} = s^\sigma_{i-1}\|t^\sigma_{i-1}$, for $\sigma \in \{0,1\}$. Since $L_{i-1,j} = 0^\lambda\|0$, it holds that $L^0_{i-1,j} = L^1_{i-1,j}$. Hence, $G(s^0_{i-1,j}) = G(s^1_{i-1,j})$, which implies that both parties compute the same τ^σ_i in Line 3.1 of Traverse. Moreover, because $L_{i-1,j} = s_{i-1}\|t_{i-1} = 0^\lambda\|0$, it follows that $t^0_{i-1} \oplus t^1_{i-1} = 0$. In turn, we have that $\gamma^0_i \oplus \gamma^1_i = \tau^0_i \oplus \tau^1_i$, since the correction word is not applied (it is multiplied by t^σ_{i-1}—an XOR share of zero). This implies that (1) $L_{i,3j} = s^0_{i,0}\|t^0_{i,0} \oplus s^1_{i,0}\|t^1_{i,0} = 0^\lambda\|0$, (2) $L_{i,3j+1} = s^0_{i,1}\|t^0_{i,1} \oplus s^1_{i,1}\|t^1_{i,1} = 0^\lambda\|0$, and (3) $L_{i,3j+2} = s^0_{i,2}\|t^0_{i,2} \oplus s^1_{i,2}\|t^1_{i,2} = 0^\lambda\|0$, as required.

In other words, the above proves that all child node labels that are off the "special path" described by α remain zero. We must now prove that all child node labels that are on the "special path" (i.e., child nodes where the parent node is non-zero) only have one non-zero sibling after the correction word is applied.

We prove this by induction starting with the root of the tree. Note that $L_0 = s^0_0\|t^0_0 \oplus s^1_0\|t^1_0$ and that $t^0_0 \oplus t^1_0 = 1$ by definition (Line 2 of DPF.Gen). Since the root label has no siblings, the base case is trivially satisfied. Now consider any parent node label $L_{i-1,j}$ of the form $L_{i-1,j} = s\|1$ for some $s \neq 0^\lambda$. Consider the child node labels $(L_{i,3j}, L_{i,3j+1}, L_{i,3j+2})$ of parent node label $L_{i-1,j}$. Since $L_{i-1,j} \neq 0^\lambda\|1$, it holds that $L^0_{i-1,j} \neq L^1_{i-1,j}$. Hence, with overwhelming probability, it holds that $G(s^0_{i-1,j}) \neq G(s^1_{i-1,j})$, which implies that both parties compute a different τ^σ_i in Line 3.1 of Traverse.

Figure 5.7: Construction of Ternary rDPF

Parameters: Pseudorandom generator $G\colon \{0,1\}^\lambda \to \{0,1\}^{3(\lambda+1)}$.

```
rDPF.Gen($1^\lambda, 1^n, \alpha$):
1: parse $\alpha = \alpha_1 \| \cdots \| \alpha_n$ where $\alpha_i \in \{0,1,2\}$ for all $i \in [n]$.
2: $s_0^0, s_0^1 \leftarrow_R \{0,1\}^\lambda$, $t_0^0 \leftarrow 0,\ t_0^1 \leftarrow 1$.
3: foreach $i \in [n]$:
    3.1: $(s_{i,0}^0 \| t_{i,0}^0 \| s_{i,1}^0 \| t_{i,1}^0 \| s_{i,2}^0 \| t_{i,2}^0) \leftarrow G(s_{i-1}^0)$.
    3.2: $(s_{i,0}^1 \| t_{i,0}^1 \| s_{i,1}^1 \| t_{i,1}^1 \| s_{i,2}^1 \| t_{i,2}^1) \leftarrow G(s_{i-1}^1)$.
    3.3: $s_{i,0} \| t_{i,0} \leftarrow (s_{i,0}^0 \| t_{i,0}^0) \oplus (s_{i,0}^1 \| t_{i,0}^1)$.
    3.4: $s_{i,1} \| t_{i,1} \leftarrow (s_{i,1}^0 \| t_{i,1}^0) \oplus (s_{i,1}^1 \| t_{i,1}^1)$.
    3.5: $s_{i,2} \| t_{i,2} \leftarrow (s_{i,2}^0 \| t_{i,2}^0) \oplus (s_{i,2}^1 \| t_{i,2}^1)$.
    3.6: $\mathsf{CW}_{i,j} \leftarrow s_{i,j} \| t_{i,j}$ for all $j \in \{0,1,2\} \setminus \{\alpha_i\}$.
    3.7: $r_i \leftarrow_R \{0,1\}^\lambda$.
    3.8: $\mathsf{CW}_{i,\alpha_i} \leftarrow (s_{i,\alpha_i} \oplus r_i) \| (t_{i,\alpha_i} \oplus 1)$.
    3.9: $\mathsf{CW}_i \leftarrow \mathsf{CW}_{i,0} \| \mathsf{CW}_{i,1} \| \mathsf{CW}_{i,2}$.
    3.10: $s_i^0 \| t_i^0 \leftarrow s_{i,\alpha_i}^0 \| t_{i,\alpha_i}^0 \oplus (t_{i-1}^0 \cdot \mathsf{CW}_{i,\alpha_i})$.
    3.11: $s_i^1 \| t_i^1 \leftarrow s_{i,\alpha_i}^1 \| t_{i,\alpha_i}^1 \oplus (t_{i-1}^1 \cdot \mathsf{CW}_{i,\alpha_i})$.
4: $\mathsf{pp} \leftarrow (\mathsf{CW}_1, \ldots, \mathsf{CW}_n)$.
5: $K_A \leftarrow (\mathsf{pp}, s_0^0 \| t_0^0)$, $K_B \leftarrow (\mathsf{pp}, s_0^1 \| t_0^1)$.
6: return $(K_A, K_B)$.

rDPF.FullEval($\sigma, K_\sigma$):
1: parse $K_\sigma = (\mathsf{pp}, s_0^\sigma \| t_0^\sigma)$.
2: $\mathsf{out} \leftarrow \mathsf{Traverse}(\sigma, \mathsf{pp}, s_0^\sigma, t_0^\sigma, 1)$, $\gamma \leftarrow 3^n$.
3: parse $\mathsf{out} = (v_1, \ldots, v_\gamma) \in (\{0,1\}^\lambda)^\gamma$.
4: return $(v_1, \ldots, v_\gamma)$.

Traverse($\sigma, \mathsf{pp}, s_{i-1}^\sigma, t_{i-1}^\sigma, i$):
1: parse $\mathsf{pp} = (\mathsf{CW}_1, \ldots, \mathsf{CW}_n)$.
2: if $i = n+1$ then
    2.1: return $s_{i-1}^\sigma \| t_{i-1}^\sigma$.
3: else
    3.1: $\tau_i^\sigma \leftarrow G(s_{i-1}^\sigma)$, $\gamma_i^\sigma \leftarrow \tau_i^\sigma \oplus (t_{i-1}^\sigma \cdot \mathsf{CW}_i)$.
    3.2: parse $\gamma_i^\sigma = s_{i,0}^\sigma \| t_{i,0}^\sigma \| s_{i,1}^\sigma \| t_{i,1}^\sigma \| s_{i,2}^\sigma \| t_{i,2}^\sigma \in \{0,1\}^{3(\lambda+1)}$.
    3.3: return $\mathsf{Traverse}(\sigma, \mathsf{pp}, s_0^\sigma, t_0^\sigma, i+1) \| \mathsf{Traverse}(\sigma, \mathsf{pp}, s_1^\sigma, t_1^\sigma, i+1) \| \mathsf{Traverse}(\sigma, \mathsf{pp}, s_2^\sigma, t_2^\sigma, i+1)$.
```

Moreover, because $t_i = t_i^0 \oplus t_i^1 = 1$, it holds that $\gamma_i = \gamma_i^0 \oplus \gamma_i^1 = (\tau_i^0 \oplus \tau_i^1) \oplus (\mathsf{CW}_{i,0} \| \mathsf{CW}_{i,1} \| \mathsf{CW}_{i,2})$. Then, by construction of $\mathsf{CW}_{i,0}$, $\mathsf{CW}_{i,1}$, $\mathsf{CW}_{i,2}$ (Lines 4.6–4.9 of DPF.Gen), we have that $\gamma_i = s_{i,0} \| t_{i,0} \| s_{i,1} \| t_{i,1} \| s_{i,2} \| t_{i,2}$ where $s_{i,k} \| t_{i,k} = 0^\lambda \| 0$ for all $k \in \{0,1,2\} \setminus \{\alpha_i\}$. In turn, we have that two (out of the three) child labels are zero. In words, this proves that at each level of the tree, only one label is non-zero, and the non-zero label is at index $\ell_i = 3^i \alpha_i + 3^{i-1}\alpha_{i-1} + \cdots + \alpha_1$ because the α_{i-1}-st child label of each non-zero parent label is always non-zero. This proves the second invariant.

To satisfy the rDPF definition, it remains to show that the non-zero leaf label of the tree is of the form $L_n = s \| 1$ where s is a pseudorandom value (from the viewpoint of either party). To see this, consider the base case of Traverse (when $i = n + 1$) for the non-zero leaf label. The output of Traverse is just $s_n \| t_n$. Therefore, for the non-zero child, $(s_n^0 \| t_n^0 \oplus s_n^1 \| t_n^1) = s_n \| t_n \oplus \mathsf{CW}_{n,\alpha_n} = s_n \oplus (s_n \oplus r_n) \| 1 = r_n \| 1$ by definition of CW_{n,α_n} in Line 3.8 of DPF.Gen. It follows that r_n is pseudorandom since it is chosen uniformly at random and is masked by s_n in CW_{n,α_n}, which is pseudorandom conditioned on either key.

Security. Informally, security holds because (1) the starting labels s_0^0 and s_0^1 given to party 0 and 1, respectively, are uniformly random and (2) the correction words (i.e., pp) consist of the expanded shares of party σ masked by the pseudorandom shares of party $1 - \sigma$ which makes them computationally indistinguishable from random from the viewpoint of party σ given K_σ.

Formally, we prove security via a sequence of hybrid distributions and reducing to the pseudorandomness of a GGM tree construction [GGM19] (generalized to have a branching factor of 3 to match our ternary tree). Fix $\alpha = \alpha_1 \| \ldots \| \alpha_n$.

Hybrid $\mathcal{H}_0$. This hybrid consists of the DPF key K_σ as defined in Figure 5.7. Specifically,

$$\mathcal{H}_0 = \{\underbrace{(\overbrace{\mathsf{CW}_{1,0} \| \mathsf{CW}_{1,1} \| \mathsf{CW}_{1,2}}^{\mathsf{CW}_1}, \ldots, \overbrace{\mathsf{CW}_{n,0} \| \mathsf{CW}_{n,1} \| \mathsf{CW}_{n,2}}^{\mathsf{CW}_n})}_{\mathsf{pp}}, s_0^\sigma \| t_0^\sigma\},$$

where $\mathsf{CW}_{i,j}$, for $i \in [n]$ and $j \in \{0,1,2\}$, is defined as $\mathsf{CW}_{i,j} = s_{i,j}^0 \| t_{i,j}^0 \oplus s_{i,j}^1 \| t_{i,j}^1$ if $j \neq \alpha_i$, $\mathsf{CW}_{i,j} = (s_{i,\alpha_i} \oplus r_i) \| (t_{i,j}^0 \oplus t_{i,j}^1 \oplus 1)$ if $j = \alpha_i \wedge i \neq n$, and $\mathsf{CW}_{n,j} = s_n \oplus \beta \| 0$ if $j = \alpha_i \wedge i = n$.

Hybrid $\mathcal{H}_1$. In this hybrid, we define each correction word as being party σ's share masked with a uniformly random mask (as opposed to being masked by a value computed by party $1 - \sigma$). Specifically,

$$\mathcal{H}_1 = \{(\overline{\mathsf{CW}}_{1,0} \| \overline{\mathsf{CW}}_{1,1} \| \overline{\mathsf{CW}}_{1,2}, \ldots, \overline{\mathsf{CW}}_{n,0} \| \overline{\mathsf{CW}}_{n,1} \| \overline{\mathsf{CW}}_{n,2}), s_0^\sigma \| t_0^\sigma\},$$

where $\overline{\mathsf{CW}}_{i,j}$, for $i \in [n]$ and $j \in \{0,1,2\}$, is defined as $\overline{\mathsf{CW}}_{i,j} = s_{i,j}^\sigma \| t_{i,j}^\sigma \oplus \mathsf{mask}_{i,j}$ if $j \neq \alpha_i$, $\overline{\mathsf{CW}}_{i,j} = (s_{i,\alpha_i}^\sigma \oplus r_i) \| (t_{i,j}^\sigma \oplus 1) \oplus \mathsf{mask}_{i,j}$ if $j = \alpha_i \wedge i \neq n$, and $\overline{\mathsf{CW}}_{n,j} = (s_n^\sigma \oplus \beta \| 0) \oplus \mathsf{mask}_{n,j}$ if $j = \alpha_i \wedge i = n$. Where for each i, j, $\mathsf{mask}_{i,j} \leftarrow_R \{0,1\}^{\lambda+1}$.

Hybrid $\mathcal{H}_2$. In this hybrid, we define each correction word as being a uniformly random string of appropriate length. That is,

$$\mathcal{H}_2 = \{(\overline{\mathsf{CW}}_{1,0}, \overline{\mathsf{CW}}_{1,1} \| \overline{\mathsf{CW}}_{1,2} \| \ldots, \overline{\mathsf{CW}}_{n,0} \| \overline{\mathsf{CW}}_{n,1} \| \overline{\mathsf{CW}}_{n,2}), s_0^\sigma \| t_0^\sigma\},$$

where $\overline{\mathsf{CW}}_{i,j}$, for all $i \in [n]$ and $j \in \{0,1,2\}$, is defined as $\overline{\mathsf{CW}}_{i,j} \leftarrow_R \{0,1\}^{\lambda+1}$.

Claim 1. $\mathcal{H}_0 \approx_c \mathcal{H}_1$

Proof. The claim follows from the pseudorandomness of the GGM construction [GGM19] (generalized to the ternary-arity case). Specifically, each $s_{i,j}^{1-\sigma} \| t_{i,j}^{1-\sigma}$ is computed as the output of a PRG applied to a pseudorandom seed $s_{i-1}^{1-\sigma} \| t_{i-1}^{1-\sigma}$, where the starting seed $s_0^{1-\sigma}$ is sampled independently of s_0^{σ}. Therefore, by the pseudorandomness of the GGM construction [GGM19, Theorem 3], for all $i \in [n], j \in \{0, 1, 2\}$, it follows that $s_{i,j}^{1-\sigma} \| t_{i,j}^{1-\sigma}$ is also pseudorandom. We can therefore replace the pseudorandom strings with *uniformly random* strings $\mathsf{mask}_{i,j}$, which proves the claim.

Claim 2. $\mathcal{H}_1 \equiv \mathcal{H}_2$

Proof. The claim follows immediately by noticing that the uniformly random strings $M_{i,j}$ and $\mathsf{mask}_{i,j}$ in $\mathcal{H}_1$ act as information-theoretic masks, making the distribution identical to $\mathcal{H}_2$.

Finally, to prove security we must prove the existence of an efficient simulator $\mathcal{S}$ that generates a computationally indistinguishable DPF key K_σ on input $(1^\lambda, 1^n, \sigma)$. The existence of $\mathcal{S}$ follows trivially by the fact that $\mathcal{H}_3$ consists of a uniformly random string of length $\{0, 1\}^{3(\lambda+1)+\lambda+1}$.

□

5.5.2 Distributed DPF Key Generation

In this section, we describe how two parties can generate DPF keys using secret-shares of the index α (i.e., the index at which the point function evaluates to a non-zero value). Our approach is inspired by the protocol of Doerner-shelat [Ds17], which makes only black-box use of OT to select the appropriate correction word at each level. While formally constructing such a protocol requires multiple functionalities and is quite tedious, conceptually, the core idea is very simple. At a high level, each party evaluates the DPF tree, layer-by-layer, and computes the correction word CW_i for layer i using a secure protocol that takes as input shares of the i-th trit $\alpha_i \in \{0, 1, 2\}$ and the "shares" of the left, middle, and right node labels $(s_{i,0} \| t_{i,0},\, s_{i,1} \| t_{i,1},\, s_{i,2} \| t_{i,2})$. The protocol then outputs CW_i exactly as computed in Figure 5.7 to both parties. However, this only results in the parties getting rDPF keys. To make the output consist of a chosen value $\beta \in \mathbb{F}_4$, we construct a separate protocol that outputs a special "output" correction word, denoted $\mathsf{CW}_{\mathsf{out}}$, that can be used to go from an rDPF output to a chosen output β (which the parties hold shares of). Conceptually, $\mathsf{CW}_{\mathsf{out}}$ is the last-layer correction word that encodes the "early termination" output in addition to β.

Overview of functionalities and instantiations. The main ideal functionality, $\mathcal{F}_{\mathsf{rDPF\text{-}DKG}}$, for computing the full rDPF keys (matching the distribution of rDPF.Gen in Figure 5.7) is presented in Figure 5.11. It is followed by an instantiation, $\Pi_{\mathsf{rDPF\text{-}DKG}}$, in Figure 5.12 where we show how to (1) compute the correction words in each i-th layer by executing the sub-protocol $\Pi_{\mathsf{rDPF\text{-}CW}}$, (2) define the input of $\Pi_{\mathsf{rDPF\text{-}CW}}$ for each i-th layer that maintains the correctness of rDPF.FullEval (indicator bits and constraints between correction words) and the rDPF.Gen can be distributed recursively. Since we are using a sub-protocol $\Pi_{\mathsf{rDPF\text{-}CW}}$, we construct its instantiation in Figure 5.9 and define its ideal functionality in $\mathcal{F}_{\mathsf{rDPF\text{-}CW}}$ Figure 5.8. $\Pi_{\mathsf{rDPF\text{-}CW}}$ shows how to securely compute the correction words for each i-th layer based on $\binom{1}{3}$-OT (note that our protocol $\Pi_{\mathsf{rDPF\text{-}CW}}$ only outputs CW_i, it does not handle the correctness of indicator bits and the constraints between correction words in two consecutive layers).

Then, in $\Pi_{\mathsf{Output\text{-}CW}}$ Figure 5.14, we show how to handle computing the "output" correction word that allows us to go from a random output (as computed by the rDPF) to a chosen output, by computing a final correction word $\mathsf{CW}_{\mathsf{out}}$ and satisfies the ideal functionality $\mathcal{F}_{\mathsf{Output\text{-}CW}}$ defined in Figure 5.13. The output of PCG-OLE is formed by the multiple shares of each party so an extra

OLE over $\mathbb{F}_4$ is used to convert from multiple shares to additive shares before being the input of Figure 5.14.

We show that all of our instantiations are secure in the semi-honest setting and we prove security in the UC model, where we only make use of the standard ideal functionality 1-out-of-3 chosen OT $\binom{1}{3}$-OT (Figure 5.10).

Figure 5.8: $\mathcal{F}_{\mathsf{rDPF\text{-}CW}}$ for computing the correction words

The functionality interacts with a party P_σ and an adversary $\mathcal{A}$.

PARAMETERS: Pseudorandom generator $G\colon \{0,1\}^\lambda \to \{0,1\}^{3(\lambda+1)}$.

FUNCTIONALITY:

```
1: Wait for input $(\llbracket\alpha_i\rrbracket_{\bar\sigma}, r_i^{\bar\sigma}, (s_{i,j}^{\bar\sigma}\|t_{i,j}^{\bar\sigma})_{j\in\{0,1,2\}}) \in \mathbb{F}_3 \times \{0,1\}^\lambda \times \{0,1\}^{3(\lambda+1)}$ from $\mathcal{A}$.
2: Wait for input $(\llbracket\alpha_i\rrbracket_{\sigma}, r_i^{\sigma}, (s_{i,j}^{\sigma}\|t_{i,j}^{\sigma})_{j\in\{0,1,2\}}) \in \mathbb{F}_3 \times \{0,1\}^\lambda \times \{0,1\}^{3(\lambda+1)}$ from party $P_\sigma$.
3: Set $\alpha_i := \llbracket\alpha_i\rrbracket_0 + \llbracket\alpha_i\rrbracket_1 \in \mathbb{F}_3$, $r_i := r_i^0 \oplus r_i^1$.
4: Compute $s_{i,j}\|t_{i,j} := (s_{i,j}^0\|t_{i,j}^0) \oplus (s_{i,j}^1\|t_{i,j}^1)$ for $j \in \{0,1,2\}$.
5: $\mathsf{CW}_{i,j} := s_{i,j}\|t_{i,j}$ for all $j \in \{0,1,2\}\setminus\{\alpha_i\}$, $\mathsf{CW}_{i,\alpha_i} := (s_{i,\alpha_i} \oplus r_i)\|(t_{i,\alpha_i} \oplus 1)$.
6: $\mathsf{CW}_i := \mathsf{CW}_{i,0}\|\mathsf{CW}_{i,1}\|\mathsf{CW}_{i,2}$.
7: Output $\mathsf{CW}_i$ to both $P_\sigma$ and $\mathcal{A}$.
```

Lemma 5.5.2 (Ternary rDPF-CW security). *The construction $\Pi_{\mathsf{rDPF\text{-}CW}}$ in Figure 5.9 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{rDPF\text{-}CW}}$ (Figure 5.8) against semi-honest adversaries in the $\binom{1}{3}$-OT hybrid model.*

Proof. We show correctness and security in turn.

Correctness. To prove the correctness, we show that in our construction $\Pi_{\mathsf{rDPF\text{-}CW}}$, $\mathsf{CW}_i = (\mathsf{CW}_{i,0}, \mathsf{CW}_{i,1}, \mathsf{CW}_{i,2})$ satisfies the following two conditions:

1. For all $j \in \{0,1,2\}\setminus\{\alpha_i\}$ then $\mathsf{CW}_{i,j} = (s_{i,j}^0\|t_{i,j}^0) \oplus (s_{i,j}^1\|t_{i,j}^1)$.
2. Otherwise, $\mathsf{CW}_{i,\alpha_i} = (s_{i,\alpha_i}^0\|t_{i,\alpha_i}^0) \oplus (s_{i,\alpha_i}^1\|t_{i,\alpha_i}^1) \oplus (r_i\|1)$.

Observe that in our construction from the definition of $\{\mathbf{C}_j^0, \mathbf{C}_j^1\}_{j\in\{0,1,2\}}$, we have $\mathbf{C}_{\alpha_i}^0 \oplus \mathbf{C}_{\alpha_i}^1 = \mathsf{CW}_i \oplus z^0 \oplus z^1$. Note that $\alpha_i = \llbracket\alpha_i\rrbracket_0 + \llbracket\alpha_i\rrbracket_1$. Consider,

$$\mathbf{M}_0^\sigma = (\mathbf{C}_0^\sigma, \mathbf{C}_1^\sigma, \mathbf{C}_2^\sigma),\ \mathbf{M}_1^\sigma = (\mathbf{C}_1^\sigma, \mathbf{C}_2^\sigma, \mathbf{C}_0^\sigma),\ \mathbf{M}_2^\sigma = (\mathbf{C}_2^\sigma, \mathbf{C}_0^\sigma, \mathbf{C}_1^\sigma),$$

for $\sigma \in \{0,1\}$. Observe that each $\mathbf{M}_j^\sigma = (\mathbf{C}_j^\sigma, \mathbf{C}_{j+1}^\sigma, \mathbf{C}_{j+2}^\sigma)$, for $j \in \{0,1,2\}$, is a cyclically shifted vector defined by shifting $(\mathbf{C}_0^\sigma, \mathbf{C}_1^\sigma, \mathbf{C}_2^\sigma)$ to the left j times.

We show that party $\sigma = 0$ obtains the correct correction word (the case where $\sigma = 1$ is symmetric). Party 0 invokes the $\binom{1}{3}$-OT as the receiver with input $\llbracket\alpha_i\rrbracket_0$, and party 1 plays the role of the sender with input

$$\mathbf{M}_{\llbracket\alpha_i\rrbracket_1}^1 = (\mathbf{C}_{\llbracket\alpha_i\rrbracket_1}^1, \mathbf{C}_{\llbracket\alpha_i\rrbracket_1+1}^1, \mathbf{C}_{\llbracket\alpha_i\rrbracket_1+2}^1).$$

After invoking $\binom{1}{3}$-OT, party 0 obtains $\mathbf{C}_{\llbracket\alpha_i\rrbracket_1+\llbracket\alpha_i\rrbracket_0}^1 = \mathbf{C}_{\alpha_i}^1$. (By a symmetric argument, party 1 gets $\mathbf{C}_{\alpha_i}^0$.) It is easy to see that $\mathbf{C}_{\alpha_i}^1 \oplus z^0$ and $\mathbf{C}_{\alpha_i}^0 \oplus z^1$ form shares of CW_i, since $(\mathbf{C}_{\alpha_i}^0 \oplus z^1) \oplus (\mathbf{C}_{\alpha_i}^1 \oplus z^0) = (\mathsf{CW}_i \oplus z^0 \oplus z^1) \oplus (z^0 \oplus z^1)$, where CW_i is defined identically to the i-th iteration of Figure 5.7. It follows that the output of the protocol (opening of the shares of CW_i) is correct.

Figure 5.9: Protocol $\Pi_{\mathsf{rDPF\text{-}CW}}$ for computing the correction words

PARAMETERS:

- Party $\sigma \in \{0,1\}$ has input $[\![\alpha_i]\!]_\sigma \in \mathbb{F}_3, r_i^\sigma \in \{0,1\}^\lambda, (s_{i,j}^\sigma \| t_{i,j}^\sigma)_{j\in\{0,1,2\}} \in \{0,1\}^{3(\lambda+1)}$.
- An instantiation of chosen $\binom{1}{3}$-OT.

PROTOCOL:

For each party $\sigma \in \{0,1\}$:

```
1: Sample $z^\sigma \leftarrow_R \{0,1\}^{3(\lambda+1)}$.
2: Define
```

$$\begin{aligned}
\mathbf{C}_0^\sigma &:= (r_i^\sigma \oplus s_{i,0}^\sigma \| (t_{i,0}^\sigma \oplus \sigma),\ s_{i,1}^\sigma \| t_{i,1}^\sigma,\ s_{i,2}^\sigma \| t_{i,2}^\sigma) \oplus z^\sigma \quad & \triangleright [\![\mathsf{CW}_i]\!]_\sigma \text{ when } \alpha_i = 0\\
\mathbf{C}_1^\sigma &:= (s_{i,0}^\sigma \| t_{i,0}^\sigma,\ r_i^\sigma \oplus s_{i,1}^\sigma \| (t_{i,1}^\sigma \oplus \sigma),\ s_{i,2}^\sigma \| t_{i,2}^\sigma) \oplus z^\sigma \quad & \triangleright [\![\mathsf{CW}_i]\!]_\sigma \text{ when } \alpha_i = 1\\
\mathbf{C}_2^\sigma &:= (s_{i,0}^\sigma \| t_{i,0}^\sigma,\ s_{i,1}^\sigma \| t_{i,1}^\sigma,\ r_i^\sigma \oplus s_{i,2}^\sigma \| (t_{i,2}^\sigma \oplus \sigma)) \oplus z^\sigma \quad & \triangleright [\![\mathsf{CW}_i]\!]_\sigma \text{ when } \alpha_i = 2\\
\mathbf{M}_0^\sigma &:= (\mathbf{C}_0^\sigma, \mathbf{C}_1^\sigma, \mathbf{C}_2^\sigma),\ \mathbf{M}_1^\sigma := (\mathbf{C}_1^\sigma, \mathbf{C}_2^\sigma, \mathbf{C}_0^\sigma),\ \mathbf{M}_2^\sigma := (\mathbf{C}_2^\sigma, \mathbf{C}_0^\sigma, \mathbf{C}_1^\sigma) &
\end{aligned}$$

```
3: Invoke $\binom{1}{3}$-OT with party $\bar\sigma$ as follows:
     - Party $\bar\sigma$ plays the role of the sender with inputs $\mathbf{M}^{\bar\sigma}_{[\![\alpha_i]\!]_{\bar\sigma}}$.
     - Party $\sigma$ plays the role of the receiver and inputs $[\![\alpha_i]\!]_\sigma \in \mathbb{F}_3$.
     - Party $\sigma$ gets $\mathbf{M}^{\bar\sigma}_{[\![\alpha_i]\!]_{\bar\sigma}}[[\![\alpha_i]\!]_\sigma] \in \{0,1\}^{3(\lambda+1)}$ while party $\bar\sigma$ gets nothing.
4: Define $[\![\mathsf{CW}_i]\!]_\sigma := \mathbf{M}^{\bar\sigma}_{[\![\alpha_i]\!]_{\bar\sigma}}[[\![\alpha_i]\!]_\sigma] \oplus z^\sigma$ and broadcast $[\![\mathsf{CW}_i]\!]_\sigma$.
5: Construct $\mathsf{CW}_i := [\![\mathsf{CW}_i]\!]_\sigma \oplus [\![\mathsf{CW}_i]\!]_{\bar\sigma} \in \{0,1\}^{3(\lambda+1)}$.
6: Output $(\mathsf{CW}_{i,0}, \mathsf{CW}_{i,1}, \mathsf{CW}_{i,2})$.
```

Figure 5.10: Ideal functionality $\binom{1}{3}$-OT in the semi-honest setting

PARAMETERS:

There are two parties, a sender and a receiver. The sender has input $(m_0, m_1, m_2) \in \{0,1\}^*$ while the receiver has input $b \in \{0,1,2\}$.

FUNCTIONALITY:

```
1: Wait for input $(m_0, m_1, m_2) \in \{0,1\}^*$ from the sender.
2: Wait for input $b \in \{0,1,2\}$ from the receiver.
3: Output $m_b$ to the receiver and $\perp$ to the sender.
```

Security. We prove our security in the UC model. In a nutshell, the proof boils down to the security of the 1-out-of-3 oblivious transfer functionality $\binom{1}{3}$-OT. We assume $\binom{1}{3}$-OT securely realizes the ideal functionality Figure 5.10 in a semi-honest setting. Since the role of the two parties is symmetric, we can build a simulator that simulates the view of both parties when one of them is corrupted. Without loss of generality, assume σ is the corrupted party, Sim interacts with $\mathcal{A}$ as in the hybrid model below.

Hybrid $\mathcal{H}_0$. This hybrid is identical to the real protocol, both parties are honest, and $\binom{1}{3}$-OT is executed honestly.

Figure 5.11: Ideal functionality $\mathcal{F}_{\textsf{rDPF-DKG}}$ for distributed key generation

The functionality interacts with a party P_σ and an adversary $\mathcal{A}$.

PARAMETERS: rDPF = (rDPF.Gen, rDPF.FullEval) as constructed in Figure 5.7.

FUNCTIONALITY:

```
1: Wait for input $(\llbracket\alpha_i\rrbracket_{\bar\sigma})_{i\in[n]} \in \mathbb{F}_3^n$, $(r_i^{\bar\sigma})_{i\in[n]} \in (\{0,1\}^\lambda)^n$, and $s_0^{\bar\sigma} \in \{0,1\}^\lambda$ from $\mathcal{A}$.
2: Wait for input $(\llbracket\alpha_i\rrbracket_{\sigma})_{i\in[n]} \in \mathbb{F}_3^n$, $(r_i^{\sigma})_{i\in[n]} \in (\{0,1\}^\lambda)^n$, and $s_0^{\sigma} \in \{0,1\}^\lambda$ from party $P_\sigma$.
3: Set $t_0^0 = 0, t_0^1 = 1$.
4: Set $\alpha_i := \llbracket\alpha_i\rrbracket_0 + \llbracket\alpha_i\rrbracket_1 \in \mathbb{F}_3$ and $r_i := r_i^0 + r_i^1 \in \{0,1\}^\lambda$ for each $i \in [n]$.
5: For each $i \in [n]$:
       Compute $\mathsf{CW}_i$ as done in Step 3 of $\mathsf{rDPF.Gen}(1^\lambda, 1^n, \alpha)$ in Figure 5.7.
6: Set $\mathsf{pp} := (\mathsf{CW}_1, \dots, \mathsf{CW}_n)$, $K_0 := (\mathsf{pp}, s_0^0 \| t_0^0)$, and $K_1 := (\mathsf{pp}, s_0^1 \| t_0^1)$.
7: Output $K_\sigma$ to $P_\sigma$ and $K_{\bar\sigma}$ to $\mathcal{A}$.
```

Hybrid $\mathcal{H}_1$. This hybrid is identical to $\mathcal{H}_0$ except that now Sim plays the role of $\mathcal{A}$ and inputs $\{\llbracket\alpha\rrbracket_\sigma, r_i^\sigma, (s_{i,j}^\sigma \| t_{i,j}^\sigma)_{j\in\{0,1,2\}}\}$ to the ideal functionality $\mathcal{F}_{\textsf{rDPF-CW}}$ and receives CW_i as output.

The distribution of this hybrid is identical to that of the previous hybrid since in this hybrid, Sim does not interact with $\mathcal{A}$.

Hybrid $\mathcal{H}_2$. This hybrid distribution is identical to $\mathcal{H}_1$ except now Sim emulates $\binom{1}{3}$-OT as follows.

- When $\mathcal{A}$ is the sender, learns the input vector $\mathbf{M}^\sigma_{\llbracket\alpha_i\rrbracket_\sigma}$ of $\mathcal{A}$. Since Sim knows $(s_{i,j}^\sigma \| t_{i,j}^\sigma)_{j\in\{0,1,2\}}$ then from $\mathbf{M}^\sigma_{\llbracket\alpha_i\rrbracket_\sigma}$, $\llbracket\alpha_i\rrbracket_\sigma$, Sim recomputes z^σ.
- When $\mathcal{A}$ is the receiver with input $\llbracket\alpha_i\rrbracket_\sigma$, Sim plays the role of $\binom{1}{3}$-OT and gives $\mathcal{A}$ an random string $\mathbf{M}^{\bar\sigma}_{\llbracket\alpha_i\rrbracket_{\bar\sigma}}[\llbracket\alpha_i\rrbracket_\sigma]$.

This hybrid is indistinguishable from $\mathcal{H}_1$ since by assumption we assume that $\binom{1}{3}$-OT realizes the ideal functionality of one-out-of-three OTs.

Hybrid $\mathcal{H}_3$. In this hybrid, Sim defines $\llbracket\mathsf{CW}_i\rrbracket_{\bar\sigma} := \mathbf{M}^{\bar\sigma}_{\llbracket\alpha_i\rrbracket_{\bar\sigma}}[\llbracket\alpha_i\rrbracket_\sigma] \oplus z^\sigma$ and $\llbracket\mathsf{CW}_i\rrbracket_{\bar\sigma} := \mathsf{CW}_i - \llbracket\mathsf{CW}_i\rrbracket_\sigma \in \{0,1\}^{3(\lambda+1)}$. Sim plays the role of party σ, sends $\llbracket\mathsf{CW}_i\rrbracket_{\bar\sigma}$ to $\mathcal{A}$.

This hybrid is indistinguishable from the previous hybrid. First, from the definition of $\llbracket\mathsf{CW}_i\rrbracket_\sigma$ and $\llbracket\mathsf{CW}_i\rrbracket_{\bar\sigma}$, the output CW_i of ideal world and real world are identically distributed. Second, in the view of $\mathcal{A}$ after invoking the $\binom{1}{3}$-OT, the message it gets $\mathbf{M}^{\bar\sigma}_{\llbracket\alpha_i\rrbracket_{\bar\sigma}}[\llbracket\alpha_i\rrbracket_\sigma]$ is uniformly random since $z^{\bar\sigma}$ used as a mask is random over $\{0,1\}^{3(\lambda+1)}$. This concludes the proof. □

Proposition 5.5.2 (Ternary rDPF-DKG security)**.** *The construction in Figure 5.12 securely realizes the ideal functionality $\mathcal{F}_{\textsf{rDPF-DKG}}$ (Figure 5.11) against semi-honest adversaries in the $\mathcal{F}_{\textsf{rDPF-CW}}$ hybrid model.*

Proof. **Correctness.** Fix an index $\alpha^i = (\alpha_1, \dots, \alpha_i)$ in a ternary tree of i-th layer. The correctness follows by the correctness of $\Pi_{\textsf{rDPF-CW}}$ Lemma 5.5.2. Since both parties invoke $\Pi_{\textsf{rDPF-CW}}$ and get $\mathsf{CW}_i \leftarrow \Pi_{\textsf{rDPF-CW}}(i, \llbracket\alpha_i\rrbracket_\sigma, r_i^\sigma, (s_{i,j}^\sigma \| t_{i,j}^\sigma)_{j\in\{0,1,2\}})$. where $\mathsf{CW}_i = (\mathsf{CW}_{i,0}, \mathsf{CW}_{i,1}, \mathsf{CW}_{i,2})$ such that:

1. For all $j \in \{0,1,2\} \setminus \{\alpha_i\}$ then $\mathsf{CW}_{i,j} = (s_{i,j}^0 \| t_{i,j}^0) \oplus (s_{i,j}^1 \| t_{i,j}^1)$.

Figure 5.12: Protocol $\Pi_{\mathsf{rDPF\text{-}DKG}}$ for Distributed Key Generation

Parameters:

- Pseudorandom generator $G\colon \{0,1\}^\lambda \to \{0,1\}^{3(\lambda+1)}$.
- There are two parties $\sigma, \bar{\sigma} \in \{0,1\}$ with input $(\llbracket \alpha_i \rrbracket_\sigma)_{i\in[n]} \in \mathbb{F}_3^n$, $(r_i^\sigma)_{i\in[n]} \in (\{0,1\}^\lambda)^n$, $s_{0,0}^\sigma \in \{0,1\}^\lambda$.

Protocol:

For each party $\sigma \in \{0,1\}$:

```
1: Set $\hat{t}^\sigma_{0,1} := \sigma$ and $\hat{s}_{0,1} := s_{0,0}$.
2: **foreach** $i \in [n]$:
   3.1: Set $d := 3^i$.
   3.2: **foreach** $j \in [d-1]$:
      3.1.1: $(s^\sigma_{i,3j} \| t^\sigma_{i,3j} \| s^\sigma_{i,3j+1} \| t^\sigma_{i,3j+1} \| s^\sigma_{i,3j+2} \| t^\sigma_{i,3j+2}) \leftarrow G(\hat{s}^\sigma_{i-1,j})$.
   3.3: $s^\sigma_{i,0} \| t^\sigma_{i,0} := \bigoplus_{j=1}^d (s^\sigma_{i,3j} \| t^\sigma_{i,3j})$.
   3.4: $s^\sigma_{i,1} \| t^\sigma_{i,1} := \bigoplus_{j=1}^d (s^\sigma_{i,3j+1} \| t^\sigma_{i,3j+1})$.
   3.5: $s^\sigma_{i,2} \| t^\sigma_{i,2} := \bigoplus_{j=1}^d (s^\sigma_{i,3j+2} \| t^\sigma_{i,3j+2})$.
   3.6: Invoke $\Pi_{\mathsf{rDPF\text{-}CW}}$ with party $\bar{\sigma}$:
        $\mathsf{CW}_i \leftarrow \Pi_{\mathsf{rDPF\text{-}CW}}(i, \llbracket \alpha_i \rrbracket_\sigma, r_i^\sigma, (s^\sigma_{i,j} \| t^\sigma_{i,j})_{j\in\{0,1,2\}})$.
   3.7: **foreach** $j \in [d-1]$:
      3.7.1: $(\hat{s}^\sigma_{i,3j} \| \hat{t}^\sigma_{i,3j} \| \hat{s}^\sigma_{i,3j+1} \| \hat{t}^\sigma_{i,3j+1} \| \hat{s}^\sigma_{i,3j+2} \| \hat{t}^\sigma_{i,3j+2}) :=$
             $(s^\sigma_{i,3j} \| t^\sigma_{i,3j} \| s^\sigma_{i,3j+1} \| t^\sigma_{i,3j+1} \| s^\sigma_{i,3j+2} \| t^\sigma_{i,3j+2}) \oplus (\hat{t}^\sigma_{i-1,j} \cdot \mathsf{CW}_i)$.
3: $\mathsf{pp} := (\mathsf{CW}_1, \ldots, \mathsf{CW}_n)$.
4: $K_A := (\mathsf{pp}, s^0_{0,0} \| 0)$, $K_B := (\mathsf{pp}, s^1_{0,0} \| 1)$.
5: **return** $(K_A, K_B)$.
```

2. Otherwise, $\mathsf{CW}_{i,\alpha_i} = (s^0_{i,\alpha_i} \| t^0_{i,\alpha_i}) \oplus (s^1_{i,\alpha_i} \| t^1_{i,\alpha_i}) \oplus (r_i \| 1)$.

Therefore, CW_i defined in $\Pi_{\mathsf{rDPF\text{-}CW}}$ has the same properties as CW_i defined in Figure 5.7. Let $s = \llbracket s \rrbracket^0 \oplus \llbracket s \rrbracket^1$. To prove correctness of rDPF.FullEval when using CW_i computed via $\Pi_{\mathsf{rDPF\text{-}CW}}$, we show that for each $i \in [n]$,

$$\hat{t}_{i-1,\alpha^i} := \hat{t}^0_{i-1,\alpha^i} \oplus \hat{t}^1_{i-1,\alpha^i} = 1,$$

otherwise $\hat{t}_{i-1,\alpha^i} = 0$ for $i \in [3^i - 1] \setminus \{\alpha^i\}$. This is done by induction for $i \in [n]$. For $i = 1$, this follows immediately. For $i \geq 1$, we have

$$\begin{aligned}&(\hat{s}_{i,3j} \| \hat{t}_{i,3j} \| \hat{s}_{i,3j+1} \| \hat{t}_{i,3j+1} \| \hat{s}_{i,3j+2} \| \hat{t}_{i,3j+2})\\ &= (s_{i,3j} \| t_{i,3j} \| s_{i,3j+1} \| t_{i,3j+1} \| s_{i,3j+2} \| t_{i,3j+2}) \oplus (\hat{t}_{i-1,j} \cdot \mathsf{CW}_i).\end{aligned}$$

Assume $\alpha^i \in \{3j^*, 3j^*+1, 3j^*+2\}$ for some $j^* \in [3^i - 1]$, from the definition of $(s^\sigma_{i,j} \| t^\sigma_{i,j})_{j\in\{0,1,2\},\sigma\in\{0,1\}}$ then we have:

1. $(s_{i,3j} \| t_{i,3j} \| s_{i,3j+1} \| t_{i,3j+1} \| s_{i,3j+2} \| t_{i,3j+2}) = (0,0,0)$ for $j \neq j^*$.

2. $\mathsf{CW}_{i,j} = s_{i,3j*+j} \| t_{i,3j*+j}$ for $j \neq \alpha_i$,
   otherwise $\mathsf{CW}_{i,\alpha_i} = (s_{i,3j*+\alpha_i} \| t_{i,3j*+\alpha_i}) \oplus (r_1 \| 1)$.

then by induction $\hat{t}_{i-1,j^*} = 1 \implies \hat{t}_{i,3j^*+\alpha_{i+1}} = 1$ otherwise, $\hat{t}_{i,j} = 1$ for $i \in [3^{i+1} - 1]$.

Security. Since our instantiation does not have any interaction between two parties except both parties invoke to $\Pi_{\mathsf{rDPF\text{-}CW}}$. So our security against semi-honest setting is directly achieved from the security of $\Pi_{\mathsf{rDPF\text{-}CW}}$ Lemma 5.5.2. Note that the distribution of our CW_i is indistinguishable from the distribution of CW_i defined in Figure 5.7 and it is proved to be secure in the view of party σ given K_σ as below. Informally, security holds because (1) the starting labels s_0^0 and s_0^1 given to party 0 and 1, respectively, are uniformly random and (2) the correction words (i.e., pp) consist of the expanded shares of party σ masked by the pseudorandom shares of party $1-\sigma$ which makes them computationally indistinguishable from random from the viewpoint of party σ given K_σ.

Formally, we prove security via a sequence of hybrid distributions and reducing to the pseudorandomness of a GGM tree construction [GGM19] (generalized to have a branching factor of 3 to match our ternary tree). Fix $\alpha = \alpha_1 \| \ldots \| \alpha_n$.

Hybrid $\mathcal{H}_0$. This hybrid consists of the DPF key K_σ as defined in Figure 5.7. Specifically,

$$\mathcal{H}_0 = \{\underbrace{(\overbrace{\mathsf{CW}_{1,0} \| \mathsf{CW}_{1,1} \| \mathsf{CW}_{1,2}}^{\mathsf{CW}_1}, \ldots, \overbrace{\mathsf{CW}_{n,0} \| \mathsf{CW}_{n,1} \| \mathsf{CW}_{n,2}}^{\mathsf{CW}_n})}_{\mathsf{pp}}, s_0^\sigma \| t_0^\sigma\},$$

where $\mathsf{CW}_{i,j}$, for $i \in [n]$ and $j \in \{0,1,2\}$, is defined as $\mathsf{CW}_{i,j} = s_{i,j}^0 \| t_{i,j}^0 \oplus s_{i,j}^1 \| t_{i,j}^1$ if $j \neq \alpha_i$, $\mathsf{CW}_{i,j} = (s_{i,\alpha_i} \oplus r_i) \| (t_{i,j}^0 \oplus t_{i,j}^1 \oplus 1)$ if $j = \alpha_i \wedge i \neq n$, and $\mathsf{CW}_{n,j} = s_n \oplus \beta \| 0$ if $j = \alpha_i \wedge i = n$.

Hybrid $\mathcal{H}_1$. In this hybrid, we define each correction word as being party σ's share masked with a uniformly random mask (as opposed to being masked by a value computed by party $1-\sigma$). Specifically,

$$\mathcal{H}_1 = \{(\overline{\mathsf{CW}}_{1,0} \| \overline{\mathsf{CW}}_{1,1} \| \overline{\mathsf{CW}}_{1,2}, \ldots, \overline{\mathsf{CW}}_{n,0} \| \overline{\mathsf{CW}}_{n,1} \| \overline{\mathsf{CW}}_{n,2}), s_0^\sigma \| t_0^\sigma\},$$

where $\overline{\mathsf{CW}}_{i,j}$, for $i \in [n]$ and $j \in \{0,1,2\}$, is defined as $\overline{\mathsf{CW}}_{i,j} = s_{i,j}^\sigma \| t_{i,j}^\sigma \oplus \mathsf{mask}_{i,j}$ if $j \neq \alpha_i$, $\overline{\mathsf{CW}}_{i,j} = (s_{i,\alpha_i}^\sigma \oplus r_i) \| (t_{i,j}^\sigma \oplus 1) \oplus \mathsf{mask}_{i,j}$ if $j = \alpha_i \wedge i \neq n$, and $\overline{\mathsf{CW}}_{n,j} = (s_n^\sigma \oplus \beta \| 0) \oplus \mathsf{mask}_{n,j}$ if $j = \alpha_i \wedge i = n$. Where for each i, j, $\mathsf{mask}_{i,j} \leftarrow_R \{0,1\}^{\lambda+1}$.

Hybrid $\mathcal{H}_2$. In this hybrid, we define each correction word as being a uniformly random string of appropriate length. That is,

$$\mathcal{H}_2 = \{(\overline{\mathsf{CW}}_{1,0}, \overline{\mathsf{CW}}_{1,1} \| \overline{\mathsf{CW}}_{1,2} \| \ldots, \overline{\mathsf{CW}}_{n,0} \| \overline{\mathsf{CW}}_{n,1} \| \overline{\mathsf{CW}}_{n,2}), s_0^\sigma \| t_0^\sigma\},$$

where $\overline{\mathsf{CW}}_{i,j}$, for all $i \in [n]$ and $j \in \{0,1,2\}$, is defined as $\overline{\mathsf{CW}}_{i,j} \leftarrow_R \{0,1\}^{\lambda+1}$.

Claim 3. $\mathcal{H}_0 \approx_c \mathcal{H}_1$

Proof. The claim follows from the pseudorandomness of the GGM construction [GGM19] (generalized to the ternary-arity case). Specifically, each $s_{i,j}^{1-\sigma} \| t_{i,j}^{1-\sigma}$ is computed as the output of a PRG applied to a pseudorandom seed $s_{i-1}^{1-\sigma} \| t_{i-1}^{1-\sigma}$, where the starting seed $s_0^{1-\sigma}$ is sampled independently of s_0^σ. Therefore, by the pseudorandomness of the GGM construction [GGM19, Theorem 3], for all $i \in [n], j \in \{0,1,2\}$, it follows that $s_{i,j}^{1-\sigma} \| t_{i,j}^{1-\sigma}$ is also pseudorandom. We can therefore replace the pseudorandom strings with *uniformly random* strings $\mathsf{mask}_{i,j}$, which proves the claim.

Claim 4. $\mathcal{H}_1 \equiv \mathcal{H}_2$

Proof. The claim follows immediately by noticing that the uniformly random strings $M_{i,j}$ and $\mathsf{mask}_{i,j}$ in $\mathcal{H}_1$ act as information-theoretic masks, making the distribution identical to $\mathcal{H}_2$.

Finally, to prove security we must prove the existence of an efficient simulator $\mathcal{S}$ that generates a computationally indistinguishable DPF key K_σ on input $(1^\lambda, 1^n, \sigma)$. The existence of $\mathcal{S}$ follows trivially by the fact that $\mathcal{H}_3$ consists of a uniformly random string of length $\{0,1\}^{3(\lambda+1)+\lambda+1}$. □

Figure 5.13: Ideal functionality $\mathcal{F}_{\mathsf{Output\text{-}CW}}$ with message β

PARAMETERS:

- The functionality interacts with a party P_σ and an adversary $\mathcal{A}$.
- Pseudorandom generator $G\colon \{0,1\}^\lambda \to (\mathbb{F}_4)^{3^t}$.

FUNCTIONALITY:

```
1: Wait for input $((\llbracket\alpha_i\rrbracket_{\bar\sigma})_{i\in[t]}, \llbracket\beta\rrbracket_{\bar\sigma}, s^{\bar\sigma}) \in (\mathbb{F}_3)^t \times \mathbb{F}_4 \times \{0,1\}^\lambda$ from $\mathcal{A}$.
2: Wait for input $((\llbracket\alpha_i\rrbracket_{\sigma})_{i\in[t]}, \llbracket\beta\rrbracket_{\sigma}, s^{\sigma}) \in (\mathbb{F}_3)^t \times \mathbb{F}_4 \times \{0,1\}^\lambda$ from party $P_\sigma$.
3: Set $\alpha_i := \llbracket\alpha_i\rrbracket_0 + \llbracket\alpha_i\rrbracket_1 \in \mathbb{F}_3$, $\alpha := \sum_{i=1}^t \alpha_i 3^{i-1} \in [3^t]$,
   $\beta := \llbracket\beta\rrbracket^0 + \llbracket\beta\rrbracket^1 \in \mathbb{F}_4$.
4: $\mathsf{CW}_t \leftarrow e_\alpha \cdot \beta \oplus G(s^0) \oplus G(s^1)$, where $e_\alpha \in (\mathbb{F}_4)^{3^t}$ is the $\alpha$-th indicator vector.
5: Output $\mathsf{CW}_t$ to $P_\sigma$ and $\mathcal{A}$.
```

Proposition 5.5.3 (Ternary Output-CW security). *The construction $\Pi_{\mathsf{Output\text{-}CW}}$ in Figure 5.14 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{Output\text{-}CW}}$ (Figure 5.13) against semi-honest adversaries in the $\binom{1}{3}$-OT hybrid model.*

Proof. Informally, the construction $\Pi_{\mathsf{Output\text{-}CW}}$ is very similar to the way we compute the intermediate CW_i in Figure 5.9, except that (1) the inputs of the sender in each $\binom{1}{3}$-OT execution of the i-th iteration are always the share of a vector $(\mathbf{C}^1_{i,\alpha_i} \oplus z^0_i) \oplus (\mathbf{C}^0_{i,\alpha_i} \oplus z^1_i) \in (\mathbb{F}_4)^{3^i}$ (the entry of position $\alpha := \sum_{k=0}^i \alpha_k 3^{k-1}$ is β, all others entries are 0), and (2) $\Pi_{\mathsf{Output\text{-}CW}}$ is computed iteratively such that the value that each party obtains (seen as $\llbracket\mathsf{CW}_i\rrbracket_\sigma$) is kept secret.

Correctness. We show that for each $i \in [t]$,

$$\llbracket\beta\rrbracket_0 \oplus \llbracket\beta\rrbracket_1 = \beta, \text{ where, } \llbracket\beta\rrbracket_i \in (\mathbb{F}_4)^{3^t}$$

As shown in the correctness proof of Lemma 5.5.2, $\mathbf{M}^\sigma_j = (\mathbf{C}^\sigma_j, \mathbf{C}^\sigma_{j+1}, \mathbf{C}^\sigma_{j+2})$ for $j \in \{0,1,2\}$ and $\sigma \in \{0,1\}$ is obtained by cyclically shifting the vector $(\mathbf{C}^\sigma_0, \mathbf{C}^\sigma_1, \mathbf{C}^\sigma_2)$ to the left j times. Moreover, we have that $\llbracket\alpha_i\rrbracket_1 + \llbracket\alpha_i\rrbracket_0 = \alpha_i$.

Consider party 0 (the case for party 1 is symmetric). Party 0 invokes the $\binom{1}{3}$-OT functionality as the receiver with input $\llbracket\alpha_i\rrbracket_0$ and party 1 plays the role of the sender with input

$$\mathbf{M}^1_{\llbracket\alpha_i\rrbracket_1} = (\mathbf{C}^1_{i,\llbracket\alpha_i\rrbracket_1}, \mathbf{C}^1_{\llbracket\alpha_i\rrbracket_1+1}, \mathbf{C}^1_{\llbracket\alpha_i\rrbracket_1+2}),$$

(viewed as a vector of vectors). Then, it holds that party 0 obtains as output

$$\mathbf{M}^1_{\llbracket\alpha_i\rrbracket_1}[\llbracket\alpha_i\rrbracket_0] = \mathbf{C}^1_{i,\llbracket\alpha_i\rrbracket_1+\llbracket\alpha_i\rrbracket_0} = \mathbf{C}^1_{i,\alpha_i}.$$

Figure 5.14: Protocol $\Pi_{\textsf{Output-CW}}$ for computing the last CWs with constraint β

PARAMETERS:

- There are two parties $\sigma, \bar{\sigma} \in \{0,1\}$ with input $(\llbracket \alpha_i \rrbracket_\sigma)_{i \in [t]} \in (\mathbb{F}_3)^t, \llbracket \beta \rrbracket_\sigma \in \mathbb{F}_4, s^\sigma \in \{0,1\}^\lambda$.
- An instantiation of chosen $\binom{1}{3}$-OT.
- Pseudorandom generator $G\colon \{0,1\}^\lambda \to (\mathbb{F}_4)^{3^t}$.

PROTOCOL:
For each party $\sigma \in \{0,1\}$, for $i \in [t]$:

```
1: Sample $z_i^\sigma \leftarrow_R (\mathbb{F}_4)^{3^i}$.
2: Define
```

$$\begin{aligned}
\mathbf{C}_{i,0}^\sigma &= (\llbracket \beta \rrbracket_\sigma, 0, 0) \oplus z_i^\sigma \in (\mathbb{F}_4)^{3^i},\\
\mathbf{C}_{i,1}^\sigma &= (0, \llbracket \beta \rrbracket_\sigma, 0) \oplus z_i^\sigma \in (\mathbb{F}_4)^{3^i},\\
\mathbf{C}_{i,2}^\sigma &= (0, 0, \llbracket \beta \rrbracket_\sigma) \oplus z_i^\sigma \in (\mathbb{F}_4)^{3^i},\\
\mathbf{M}_0^\sigma &= (\mathbf{C}_{i,0}^\sigma, \mathbf{C}_{i,1}^\sigma, \mathbf{C}_{i,2}^\sigma),\ \mathbf{M}_1^\sigma = (\mathbf{C}_{i,1}^\sigma, \mathbf{C}_{i,2}^\sigma, \mathbf{C}_0^\sigma),\ \mathbf{M}_2^\sigma = (\mathbf{C}_{i,2}^\sigma, \mathbf{C}_{i,0}^\sigma, \mathbf{C}_{i,1}^\sigma)
\end{aligned}$$

```
3: Invoke $\binom{1}{3}$-OT with party $\bar{\sigma}$ as follows:
    - Party $\bar{\sigma}$ plays the role of the sender with inputs $\mathbf{M}^{\bar{\sigma}}_{\llbracket \alpha_i \rrbracket_{\bar{\sigma}}}$.
    - Party $\sigma$ plays the role of the receiver and inputs $\llbracket \alpha_i \rrbracket_\sigma \in \mathbb{F}_3$.
    - Party $\sigma$ gets $\mathbf{M}^{\bar{\sigma}}_{\llbracket \alpha_i \rrbracket_{\bar{\sigma}}}[\llbracket \alpha_i \rrbracket_\sigma] \in (\mathbb{F}_4)^{3^i}$ while party $\bar{\sigma}$ gets nothing.
4: Define $\llbracket \beta \rrbracket_\sigma := \mathbf{M}_i^{\bar{\sigma}}[\llbracket \alpha_i \rrbracket_\sigma] \oplus z_i^\sigma \in (\mathbb{F}_4)^{3^i}$.
Output $\llbracket \mathsf{CW} \rrbracket_t := \llbracket \beta \rrbracket_\sigma \oplus G(s^\sigma)$.
```

By symmetry, party 1 then obtains $\mathbf{C}_{i,\alpha_i}^0$ by invoking the $\binom{1}{3}$-OT protocol with party 0 now playing the role of the sender. Then, letting $\llbracket \beta \rrbracket_1 = \mathbf{C}_{i,\alpha_i}^0 \oplus z_i^1$ and $\llbracket \beta \rrbracket_0 = \mathbf{C}_{i,\alpha_i}^1 \oplus z_i^0$ be the shares of β, because the z_i^0 and z_i^1 terms cancel out the masking terms added by each party, we get that

$$\llbracket \beta \rrbracket_0 \oplus \llbracket \beta \rrbracket_1 = \mathbf{C}_{i,\alpha_i}^1 \oplus z_i^1 \oplus \mathbf{C}_{i,\alpha_i}^0 \oplus z_i^0 = \begin{cases} (\beta, 0, 0), & \text{if } \alpha_i = 0. \\ (0, \beta, 0), & \text{if } \alpha_i = 1. \\ (0, 0, \beta), & \text{if } \alpha_i = 2. \end{cases}$$

or in other words, $\llbracket \beta \rrbracket_0 \oplus \llbracket \beta \rrbracket_1 = \mathbf{e}_{\alpha_i} \cdot \beta$, where $\mathbf{e}_{\alpha_i}$ is the α_i-th standard basis vector over $\mathbb{F}_4^{3^i}$.

To see that correctness still holds after the t-th iteration, $\{\mathbf{C}_{i+1,j}^\sigma\}_{\{j \in \{0,1,2\}}$ in $(i+1)$-th iteration is defined by $\llbracket \beta \rrbracket_\sigma \in (\mathbb{F}_4)^{3^i}$ from the i-th iteration and adding 0's to make sure that β and the position of share value in vector of size $(\mathbb{F}_4)^{3^i}$ is $\sum_{k=0}^{i} \alpha_k 3^{k-1}$. This leads to $\mathsf{CW} := e_\alpha \cdot \beta \oplus G(s^0) \oplus G(s^1) \in (\mathbb{F}_4)^{3^t}$ where $\alpha = \sum_{i=0}^{t} \alpha_i 3^{i-1}$, as required.

Security Analysis. The simulator Sim is constructed similarly to the simulator in the proof of Lemma 5.5.2, except that Sim does not play the role of party σ to output $\llbracket \beta \rrbracket_\sigma$ for each iteration. Instead, it only outputs the share for the last iteration, which allows both parties to construct the output CW. Sim emulates $\binom{1}{3}$-OT to learn the mask $z_i^{\bar{\sigma}}$ and inputs of $\mathcal{A}$ that plays the role of the sender to $\binom{1}{3}$-OT in the i-th iteration. Then, using this information, Sim simulates the $(i+1)$-th iteration. And for the last step, to simulate the output of the honest party, Sim defines $[\![\mathsf{CW}]\!]_\sigma$ from $[\![\mathsf{CW}]\!]_\sigma$ that is constructed by emulating $\binom{1}{3}$-OT and CW the output of ideal functionality $\mathcal{F}_{\mathsf{Output\text{-}CW}}$. □

5.6 Implementation and Evaluation

We implement $\mathbb{F}_4$OLEAGE in C (v15.0.0) as a library that consists of two main components: (1) an optimized implementation of the ternary DPF construction and (2) an implementation of the FFT over $\mathbb{F}_4$.

The open-source code for our $\mathbb{F}_4$OLEAGE PCG benchmarks is available online.[9]

Parameter Estimation and Revaluation of [BCC+23].

We provide a SageMath [Ste+24] script to help choose a set of concrete parameters for QA-SD. It computes the probability distribution of the weight of the folded error and computes the cost of the best attack. It also takes into account the previous attack.

As a by-product, we can use it to give a new estimation of the security of [BCC+23]. Results are given in Section 5.6. It shows that we would need to significantly increase the value of t in order to achieve 128 bits of security. The parameters are chosen using a new and improved analysis of the QA-SD assumption, the details of cryptanalysis can be found in the full version [BBC+24].

n	c	t	$(n_{\text{fold}}, k_{\text{fold}}, \omega_0)$	$(\mathrm{N}_{\text{iter}}, \mathrm{Cost}_{\text{Decoding}})$	Number of subgroups	Actual security
25	4	16	$(2048, 1536, 54)$	$(2^{14}, 2^{89})$	2^{145}	118
30	4	16	$(2048, 1536, 54)$	$(2^{14}, 2^{89})$	2^{190}	118
35	4	16	$(2048, 1536, 54)$	$(2^{14}, 2^{89})$	2^{235}	118

Table 5.2: Reestimation of the security for the parameters given in [BCC+23]. They were considered to yield more than 128 bits of security, they were even considered to be conservative. Note that in [BCC+23], all the parameters were for $q = 3$. Here t is the number of errors per block, while in [BCC+23] it was the total number of errors. n_{fold} and k_{fold} are respectively the length and dimension of the folded code. N_{iter} is the number of different foldings necessary to run the attack, and ω_0 is the optimal target weight.

Implementation details. Our DPF implementation takes advantage of the `AES-NI` instruction to implement a fast PRG G using fixed-key AES (from the OpenSSL library [Ope]) and the Davies-Meyer transform. We experimented with using the half-tree optimization of [GYW+23]. However, we observed a minimal performance gains (2-4%) from this optimization when applied to a ternary tree. This is because the half-tree optimization is tailored to the binary tree DPF construction where it can shave a larger fraction of total AES calls. [10]

We implement the recursive FFT over $\mathbb{F}_4$ described in Section 5.4.3 and perform the FFT in parallel by packing all the coefficients into one machine word (for our parameters, we will require 16

[9] https://github.com/sachaservan/FOLEAGE-PCG.

[10] Since there are roughly 1/3 nodes for which we can hope to shave AES calls in the ternary tree, but we can only shave one-out-of-three AES calls using the “half-tree” optimization in the ternary-tree case, we can only hope to save $1/3 \cdot 1/3 \approx 10\%$ of the AES computation time. However, given that AES calls are dominant but not the entire cost of the DPF (we also need to compute many XORs), we end up with roughly 2-4% savings. This could perhaps be optimized a bit further, but the performance savings would plateau at roughly 6%.

FFTs, so we can perform them in parallel using a `uint32` type for packing). While the FFT could possibly be optimized further using an iterative algorithm and taking advantage of AVX instructions, the simplicity of the recursive algorithm coupled with the parallel packing makes it sufficiently fast for $\mathbb{F}_4$OLEAGE. This is especially true given that the DPF evaluations end up being the dominant cost (roughly 70% of the total computation). We do not implement the distributed seed generation protocol given that it consists of black-box invocations of any one-out-of-three OT. However, we do estimate the concrete performance and communication costs of distributed seed generation by benchmarking the libOTe library on state-of-the-art OT protocols [RR].

Benchmarks. We perform our benchmarks using AWS `c5.metal` (3.4GHz CPU) and `t2.large` instances. All experiments are averaged across ten trials and evaluated on a single core. To gain a better understanding of the overhead involved with each component, we start by benchmarking the SPFSS (sum of many DPFs) and FFT implementation separately and report the results in Tables 5.3 and 5.4. Concretely, if we are packing 3^n coefficients over $\mathbb{F}_4$, we want the output of the DPF to be close to a power of 3. To achieve this, we terminate 5 levels early and pack 512 elements of $\mathbb{F}_4$ in the virtual leaves by having the DPF output be a 1024 bit block. Therefore, the key size of each DPF is $3 \cdot 128 \cdot (n-5) + 128 + 2 \cdot 512$ when using AES with 128-bit keys. We report the SPFSS benchmarks in Table 5.3 when evaluating the sum of 730 DPFs (this corresponds to the $t = 27$ regime in Figure 5.3, since the SPFSS needs to be instantiated with $t^2 = 729$ DPFs). When evaluating the SPFSS, we observe a roughly $1.8\times$ reduction in computation time over evaluating just one DPF. This is due to better cache performance when evaluating many DPFs and working over the same memory allocation to evaluate consecutive DPFs. Our choice of DPF range 3^{11}, 3^{13}, and 3^{15} correspond to the size of a regular noise block when $D = 3^{14}$, $D = 3^{16}$, and $D = 3^{18}$, respectively (see Table 5.5).

Range	SPFSS.Gen	SPFSS.FullEval	AES	**Key Size**
(elements of $\mathbb{F}_4$)	(c5.metal | t2.large)	(c5.metal | t2.large)	(c5.metal | t2.large)	(per party)
3^{11}	5 ms | 11 ms	26 ms | 39 ms	18 ms | 27 ms	315 kB
3^{13}	7 ms | 13 ms	260 ms | 364 ms	174 ms | 253 ms	385 kB
3^{15}	8 ms | 16 ms	2357 ms | 3272 ms	1526 ms | 2229 ms	456 kB

Table 5.3: Performance of our SPFSS (for the sum of 730 DPFs) on two EC2 instances and comparison to the raw AES computation time required for the PRG evaluations.

Number of Variables	Packed FFT ($4\times$)	Packed FFT ($16\times$)	Packed FFT ($32\times$)
	(c5.metal | t2.large)	(c5.metal | t2.large)	(c5.metal | t2.large)
14	20 ms | 30 ms	21 ms | 33 ms	28 ms | 45 ms
16	180 ms | 280 ms	213 ms | 329 ms	312 ms | 475 ms
18	1682 ms | 2608 ms	2165 ms | 3280 ms	4913 ms | 7478 ms

Table 5.4: Performance of our FFT implementation over $\mathbb{F}_4$ on two different EC2 instances. Packing increases throughput almost linearly with the packing size. However, with a large number of variables (> 16), it is more efficient to use smaller packing values to avoid the increased memory usage from the recursive FFT function calls.

Benchmarking our PCG. Next, we benchmark the performance of the PCG from Figure 5.3 on various parameters. The parameter $D = 3^n$ determines the number of Beaver triples we generate in total. In contrast, the parameters c (compression factor) and t (noise weight) influence the size of the PCG key and evaluation time. Specifically, evaluating the PCG requires $(c \cdot t)^2$ calls to the DPF on domain size D/t (due to regular noise) and $c(c+1)/2$ calls to the FFT (which we can parallelize by a factor of up to 32 using packing on 64-bit architectures). The DPF evaluation cost ends up being the dominant factor (approximately 70%) in the total computation. The FFT accounts for less than 5% of the total computation. Interestingly, *packing* the FFT (which requires computing a matrix transpose of dimension $c(c+1)/2 \times 3^n$ to translate from $c(c+1)/2$ polynomials to a packed representation suitable for computing the FFT in parallel) accounts for 15% of the total computation! This motivates using small values of c, such as $c = 4$, as otherwise this transpose becomes the dominant cost in the entire PCG expansion. We leave exploring the possibility of implementing fast SIMD-based matrix-transpose algorithms (e.g., [TE76; AS20]) as a promising direction for future work, since it may allow using a smaller noise weight (e.g., $t = 9$) and larger c.

We set $t = 27$ since we need it to be a power of 3 (see Remark 5.4.1), and report the computational costs of the PCG for different values of D in Table 5.5 and c.

The choice of $(c = 4, t = 27)$ corresponds to a conservative parameter choice based on our calculations. To show the influence of c on the performance, we also evaluate our PCG construction on $c = 3$, which corresponds to a more aggressive parameter choice. We observe a much smaller PCG seeds and better concrete performance with $c = 3$ compared to $c = 4$.

(a) Parameters: ($c = 4$, $t = 27$)

D	PCG.Expand (c5.metal | t2.large)	**Key Size** (per party)
3^{14}	579 ms | 890 ms	5.0 MB
3^{16}	5.9 s | 8.4 s	6.2 MB
3^{18}	54.3 s | –	7.3 MB

(b) Parameters: ($c = 3$, $t = 27$)

D	PCG.Expand (c5.metal | t2.large)	**Key Size** (per party)
3^{14}	346 ms | 534 ms	2.8 MB
3^{16}	3.5 s | 5.2 s	3.5 MB
3^{18}	32.1 s | –	4.1 MB

Table 5.5: Performance of our PCG implementation on two different EC2 instances. We set the noise parameter to $t = 27$ and let $c = 4$ in the left table (our conservative parameter choice) and $c = 3$ in the right table (our aggressive parameter choice); $D = 3^{18}$ ran out of memory on the `t2.large`.

Estimating setup costs. We use the libOTe library [RR] to benchmark the state-of-the-art OT protocols. We run libOTe on `localhost` and evaluated both SoftSpoken OT [Roy22] and the RRT' silent OT [RRT23]. For SoftSpoken, we measured roughly 50,000,000 OT/s on the `c5.metal` machine and roughly 32,000,000 OT/s on the `t2.large`. For the RRT, we measure a throughput of nearly 7,000,000 on `c5.metal` and 4,000,000 on the `t2.large`. To run our distributed DPF key generation protocol, we require $n = 14$ (at $D = 3^{14}$) and $n = 18$ (at $D = 3^{18}$) rounds per DPF. All the $(ct)^2$ DPF keys can be computed in parallel. Therefore, in total, using our conservative parameters of $c = 4$ and $t = 27$, we require roughly 11,600 parallel calls to an OT functionality in n rounds. Our aggressive parameters of $c = 3$ and $t = 27$ only require 6,561 parallel OT calls.

5.7 N-party MPC with Preprocessing from $\mathbb{F}_4$-OLEs

5.7.1 Secure Computation in the $\mathcal{F}_{\mathsf{cBT}}$-Hybrid Model

In this section, we show how to securely compute arbitrary Boolean circuits in the preprocessing model, given access to an ideal functionality generating Beaver triples over $\mathbb{F}_4$. Because $\mathbb{F}_4$ is an extension field of $\mathbb{F}_2$, we note that simply replacing the $\mathbb{F}_2$-Beaver triples with $\mathbb{F}_4$-Beaver triples in the classical instantiation of the GMW protocol in the preprocessing model works out-of-the-box. However, doing so naïvely *doubles* the communication during the online phase, from 2 bits per AND gate and per party to 4 bits per AND gate and per party (due to using masks over $\mathbb{F}_4$ instead of $\mathbb{F}_2$). In the technical overview, we introduced an improved strategy, which first converts each $\mathbb{F}_4$-triple into an $\mathbb{F}_2$-triple using one bit of communication per party, and then runs the standard GMW protocol

Figure 5.15: Ideal Functionality $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F})$ for sampling N-party Beaver triples

The functionality interacts with N parties $P_1, \ldots, P_N$ and an adversary $\mathcal{A}$.

FUNCTIONALITY:

1: Wait for the input $\mathsf{Corr} \subsetneq [N]$ from $\mathcal{A}$ consisting of the set of corrupted parties and a list $(\llbracket a \rrbracket_i, \llbracket b \rrbracket_i, \llbracket c \rrbracket_i)_{i \in \mathsf{Corr}}$ of triples in $\mathbb{F}$.

2: Wait for the command init from each party P_i for $i \notin \mathsf{Corr}$.

3: Sample $(\llbracket a \rrbracket_i, \llbracket b \rrbracket_i, \llbracket c \rrbracket_i)_{i \in [N] \setminus \mathsf{Corr}} \in \mathbb{F}^3$ uniformly at random conditioned on

$$a \cdot b = \left(\sum_{i=1}^{N} \llbracket a \rrbracket_i\right) \cdot \left(\sum_{i=1}^{N} \llbracket b \rrbracket_i\right) = \sum_{i=1}^{N} \llbracket c \rrbracket_i.$$

4: Output $(\llbracket a \rrbracket_i, \llbracket b \rrbracket_i, \llbracket c \rrbracket_i)$ to each party P_i for $i \notin \mathsf{Corr}$.

over $\mathbb{F}_2$. To formally prove this result, we introduce on Figure 5.15 a corruptible functionality for generating an N-party Beaver triple over a field $\mathbb{F}$. Here, corruptible means that the adversary can freely choose the shares obtained by the corrupted parties; it is known that this functionality suffices to securely instantiate GMW [BCG+19b] and can be securely instantiated given a programmable PCG for OLE over $\mathbb{F}$ [BCG+20b]. That is:

Theorem 5.7.1 ([BCG+19b, Theorem 19, Theorem 41]). *Assume that there is a programmable PCG for generating m OLEs over $\mathbb{F}$. Then there exists a protocol securely realizing m calls to the functionality $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F})$ in Figure 5.15 using $N \cdot (N-1)$ instances of a protocol to securely distribute the seeds of the PCG, and no further communication.*

We formally state our construction as a protocol $\Pi_{\mathsf{BT}}(\mathbb{F}_4 \to \mathbb{F}_2)$ that securely instantiates the $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_2)$ functionality in the $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_4)$-hybrid model, using a single call to $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_4)$ and one bit of communication per party. The protocol is represented in Figure 5.16.

Figure 5.16: N-party protocol $\Pi_{\mathsf{BT}}(\mathbb{F}_4 \to \mathbb{F}_2)$

PROTOCOL:

1: The parties invoke the functionality $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_4)$ with init. Each party P_i receives a triple $(\llbracket a \rrbracket_i^4, \llbracket b \rrbracket_i^4, \llbracket c \rrbracket_i^4) \in \mathbb{F}_4^3$.

2: Each party P_i broadcasts $\llbracket b \rrbracket_i^4(1)$. All parties reconstruct $b(1) = \sum_{i=1}^{N} \llbracket b \rrbracket_i^4(1)$.

OUTPUT: Each party P_i outputs $(\llbracket a \rrbracket_i^4(0), \llbracket b \rrbracket_i^4(0), \llbracket c \rrbracket_i^4(0) + b(1) \cdot \llbracket a \rrbracket_i^4(1))$.

Lemma 5.7.1. *The protocol $\Pi_{\mathsf{BT}}(\mathbb{F}_4 \to \mathbb{F}_2)$ of Figure 5.16 securely realizes the $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_2)$ corruptible functionality in the $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_4)$-hybrid model, using one bit of communication per party and a single call to $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_4)$.*

Combining this lemma with the GMW protocol yields:

Corollary 5.7.1. *There exists an N-party computation protocol that securely evaluates all Boolean circuits with m AND gates in the preprocessing model using m calls to the $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_4)$ functionality. The protocol uses $N \cdot m$ bits of communication in the preprocessing phase, and two bits of communication per AND gate and per party in the online phase.*

Proof of Lemma 5.7.1.

Proof. Sim emulates the $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_4)$ functionality, and receives from $\mathcal{A}$ the set $\mathsf{Corr} \subsetneq [N]$ of corrupted parties and the list $([\![a]\!]_i^4, [\![b]\!]_i^4, [\![c]\!]_i^4)_{i\in\mathsf{Corr}}$ of corrupted triples over $\mathbb{F}_4$. On behalf of each honest party P_i for $i \notin \mathsf{Corr}$, Sim broadcast a uniformly random bit $[\![b]\!]_i^4(1)$. Then, Sim reconstructs $b(1) \leftarrow \sum_{i=1}^N [\![b]\!]_i^4(1)$ and sends $([\![a]\!]_i^4(0), [\![b]\!]_i^4(0), [\![c]\!]_i^4(0) + b(1)\cdot[\![a]\!]_i^4(1))_{i\in\mathsf{Corr}}$ to the ideal functionality $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_2)$ on behalf of the ideal adversary. As the $[\![b]\!]_i^4(1)$ are sampled uniformly and independently at random by $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_4)$, it only remains to show that the honest parties output in an execution of $\Pi_{\mathsf{BT}}(\mathbb{F}_4 \to \mathbb{F}_2)$ is distributed as the output of the adversaries from $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_2)$ in the simulated game. In turn, this follows immediately from the fact that from the viewpoint of $\mathcal{A}$, the shares $[\![a]\!]_i^4(0), [\![b]\!]_i^4(0)$ are uniformly distributed for every $i \notin \mathsf{Corr}$, and the values $[\![c]\!]_i^4(0) + b(1)\cdot[\![a]\!]_i^4(1)$ for $i \notin \mathsf{Corr}$ form uniformly random shares of

$$\begin{aligned}
\sum_{i\notin\mathsf{Corr}} \left([\![c]\!]_i^4(0) + b(1)\cdot[\![a]\!]_i^4(1)\right) &= \sum_{i=1}^N \left([\![c]\!]_i^4(0) + b(1)\cdot[\![a]\!]_i^4(1)\right) - C \\
&= a(0)b(0) + a(1)b(1) + b(1)\cdot a(1) - C \\
&= a(0)b(0) - C,
\end{aligned}$$

where $C \leftarrow \sum_{i\in\mathsf{Corr}} [\![c]\!]_i^4(0) + b(1)\cdot[\![a]\!]_i^4(1)$ denote the sum of the corrupted parties' last output. This concludes the proof. □

5.7.2 Improved Protocol for $N = 2$ Parties

In the previous section, we described how N parties can construct an $\mathbb{F}_2$-triple using one invocation to $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_4)$ and N bits of communication. Typically, the functionality $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_4)$ is realized by making $N\cdot(N-1)$ calls to a (programmable) OLE functionality over $\mathbb{F}_4$. When $N = 2$, this translates to using two calls to the OLE functionality, and two bits of communication. In this section, we introduce an improved construction, where $N = 2$ parties generate a Beaver triple over $\mathbb{F}_2$ using a *single* call to an OLE functionality over $\mathbb{F}_4$, *and no communication.* The corruptible functionality for generating OLEs over $\mathbb{F}_4$ is represented on Figure 5.17.

Figure 5.17: Ideal Functionality $\mathbb{F}\text{-}\mathsf{OLE}(\mathbb{F})$ for sampling an OLE

The functionality interacts with 2 parties A, B and an adversary $\mathcal{A}$.

FUNCTIONALITY:

1: Wait for an input $(\mathsf{Corr}, u, v) \in \{A, B, \bot\} \times \mathbb{F} \times \mathbb{F}$ from $\mathcal{A}$, and for the command init from each party.

2: If $\mathsf{Corr} = \bot$, sample $(a, b, [\![ab]\!]_A) \leftarrow\$\ \mathbb{F}^3$ and set $[\![ab]\!]_B \leftarrow a\cdot b - [\![ab]\!]_A$. If $\mathsf{Corr} = A$, sample $b \leftarrow\$\ \mathbb{F}$, set $(a, [\![ab]\!]_A) \leftarrow (u, v)$ and $[\![ab]\!]_B \leftarrow a\cdot b - [\![ab]\!]_A$. If $\mathsf{Corr} = B$, sample $a \leftarrow\$\ \mathbb{F}$, set $(b, [\![ab]\!]_B) \leftarrow (u, v)$ and $[\![ab]\!]_A \leftarrow a\cdot b - [\![ab]\!]_B$.

3: Output $(a, [\![ab]\!]_A)$ to A and $(b, [\![ab]\!]_B)$ to B.

In Figure 5.18, we represent our protocol for realizing $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_2)$ when $N = 2$ in the $\mathbb{F}\text{-}\mathsf{OLE}(\mathbb{F}_4)$-hybrid model.

Lemma 5.7.2. *The protocol $\Pi(\mathbb{F}_4\mathsf{OLE} \to \mathbb{F}_2\mathsf{BT})$ of Figure 5.18 securely realizes the $\mathcal{F}_{\mathsf{cBT}}(\mathbb{F}_2)$ corruptible functionality for $N = 2$ parties in the $\mathbb{F}\text{-}\mathsf{OLE}(\mathbb{F}_4)$-hybrid model, using no communication and a single call to $\mathbb{F}\text{-}\mathsf{OLE}(\mathbb{F}_4)$.*

Figure 5.18: A 2-party protocol $\Pi(\mathbb{F}_4\mathsf{OLE} \to \mathbb{F}_2\mathsf{BT})$

PROTOCOL:

1: The parties invoke the functionality $\mathbb{F}\text{-}\mathsf{OLE}(\mathbb{F}_4)$ with init, and receive $(a, [\![ab]\!]_A^4)$ and $(b, [\![ab]\!]_B^4)$, respectively.

OUTPUT:

Alice outputs $(a(0), a(1), a(0)a(1) + [\![ab]\!]_A^4(0))$ and Bob outputs $(b(1), b(0), b(0)b(1) + [\![ab]\!]_B^4(0))$.

We refer the reader to the technical overview (Section 5.3.4) for the correctness analysis. The proof of security is straightforward and we omit it.

5.8 Faster Seed Expansion from Hashing

In this section, we describe how to combine hashing techniques with our PCG construction to reduce the number of DPFs required to share the noise vector. This has the advantage of making seed expansion faster with the number of PRGs operations being $O(2|\mathbb{G}|)$, but comes at the cost of requiring a trusted setup. As such, this optimization is primarily suited to silent-OT applications where a trusted setup process is assumed.

5.8.1 Faster Seed Expansion

We propose a better $\mathsf{fQA\text{-}SD_{OLE}}$ scheme in terms of seed expansion efficiency that is based on the DPF scheme and hashing techniques. First, we take advantage of Cuckoo hashing and simple hashing [KKR+16; PSW+18; BC23; RS21] to distribute the noise positions. After that, we use a DPF scheme for each bin to obtain the shares of value in each noise position by using a DPF for each bin. In general, using our Cuckoo hashing trick, we obtain a faster seed expansion but need to assume a trusted dealer because the Doerner-shelat protocol does not apply out-of-the-box. However, our $\mathsf{fQA\text{-}SD_{OLE}}$ construction has an advantage compared to the $\mathsf{QA\text{-}SD_{OLE}}$ protocol, i.e., the computation cost of Expand is smaller and independent of t (number of noisy coordinates). Concretely, the number of PRG operations is reduced from $O(t|\mathbb{G}|)$ to $O(2|\mathbb{G}|)$. Just as with the $\mathsf{QA\text{-}SD_{OLE}}$ construction, our $\mathsf{fQA\text{-}SD_{OLE}}$ construction can be used to obtain OLE correlation over any finite field $\mathbb{F}$ (except for $\mathbb{F}_2$), however, to be consistent with the concept of our main contribution, we cast our $\mathsf{fQA\text{-}SD_{OLE}}$ over $\mathbb{F}_3$ for concrete efficiency.

Hashing schemes

We fix K random hash functions $h_1, \cdots, h_K$, where $h_i : \mathbb{G} \to [m]$ and $m = O(n)$. Using these K hash functions, the formal definition and parameter choices of Cuckoo hashing and simple hashing schemes (similarly as in PSI works [KD08; KKR+16; BC23; RS21]) are as follows:

1. **Cuckoo hashing schemes** with parameters $(\mathbb{G}, K, m, n)$ enable mapping a set of n item into a table T of size m using K hash functions $(h_i)_{i \leq K}$ such that each bin in T has at most one item. The algorithm takes an item $x \in \mathbb{G}$ and inserts it into the bin $T[h_1(x)]$, if this bin is occupied then evicts the item in this bin and relocates it using h_2, this process continues until all items are inserted in table T. The hashing algorithm can fail if a cycle eviction is found or a threshold number of relocations has been performed, this failure can be avoided with high probability by choosing appropriate parameters (K, m, n) or using a stash to store

the last item in each cycle eviction if it exists [PSW+18]. Here, we choose the parameters such that $K = 2, m = 2n$ [BC23; RS21], and do not use a stash. Looking ahead, this will imply that insertion will have a noticeable failure probability; however, we will show that in this case, the trusted dealer can simply re-sample the noise vector until insertion succeeds. This induces a small bias on the noise, but a straightforward reduction shows that it does not harm the security of the underlying syndrome decoding assumption (if the insertion fails with probability α, then the reduction loses a factor α in the advantage). For $K = 2$, $m = 2n$, the failure probability of Cuckoo hashing is known to be $\sqrt{2/3} + o(1)$ [KD08], which translates to a small constant security loss. We note in passing that the same observation applies to the use of Cuckoo hashing in a previous work [SGR+19] and allows them to reduce the computational overhead of their LPN-based construction from 3 to 2 compared to their regular LPN-based construction.

2. **Simple hashing** with parameters $(\mathbb{G}, K, m)$ uses K hash functions $(h_i)_{i \le K}$ to insert each item $x \in \mathbb{G}$ to the bin $B[h_i(x)]$ of a table B of size m. With very high probability, for $m = O(n \log n)$ bins, the maximum possible items per bin is $O(\log m)$. In particular, by the randomness of hash functions, with high probability, the maximum number of items per bin (denoted as max_load) is bounded by $\frac{3 \ln m}{\ln \ln m}$. That is, $\Pr\left[\mathsf{max\_load} \ge \frac{3 \ln m}{\ln \ln m}\right] \le \frac{1}{m}$. To estimate our concrete efficiency, we highlight the total number of items in all bins always is $K \cdot |\mathbb{G}|$ since each item in $\mathbb{G}$ is mapped to K bins using different K hash functions.

OLE from hashing techniques

In our construction we have $\mathbb{G} = \prod_{i=1}^{n} \mathbb{Z}/(q-1)\mathbb{Z}$. Because we are working over $\mathbb{F}_4$, $|\mathbb{G}| = 3^n$. We later work on "balls and bins" where each bin has a different number of balls and is associated with a DPF to distributed shares of a point vector then we make use of notation DPF_n for DPF scheme with arbitrary domain $[n]$.

We reuse all notations defined in Section 5.3.5, the intuition of the construction is the same as in Section 5.3.5 except that we provide a more efficient way to distribute the position of noise coordinates before using the DPF to give each party the shares of point vector. Briefly, to construct an OLE correlation over the ring $\mathcal{R}$, we want to give the parties shares of $x_0 \cdot x_1$. Note that $x_0 \cdot x_1$ is a degree-2 function in $(\mathbf{e_0}, \mathbf{e_1})$; therefore, it suffices to share $\mathbf{e_0} \otimes \mathbf{e_1}$. Since $\mathbf{e_0}, \mathbf{e_0} \in \mathcal{R}_t^c$ both are sparse vectors of weight t over $\mathcal{R}_t$, where the product of two sparse vectors is a sparse vector with sparsity t^2. So the goal here is to securely distribute the tensor $\mathbf{e_0} \otimes \mathbf{e_1}$ to both parties as in other existing PCGs (Figure 5.3 and Figure 5.2). Note that $\mathbf{e_\sigma} = (e_{\sigma^0}, \cdots, e_{\sigma^{c-1}})$, $\sigma \in \{0, 1\}$, and each random $e_{\sigma^i} \in \mathbb{R}_t$ is defined by a pair of vectors $\mathbf{p_\sigma^i} \in \mathbb{G}^t$ and $\mathbf{v_\sigma^i} \in \mathbb{F}_3^t$, which can be considered as the set of positions of non-zero entries of a vector over $\mathbb{R}_t$ and the corresponding values of these entries.

Then, $\mathbf{e_0} \otimes \mathbf{e_1}$ is defined as a vector over $\mathbb{R}_{t^2}$, where $(\mathbf{v_0^i} \otimes \mathbf{v_1^j})[k]$, with $k \in [t^2]$, is the value of the entry at position $(\mathbf{p_0^i} \otimes \mathbf{p_1^j})[k]$.

Now, the seed generation and seed expansion are processed as follows. Simple hashing is applied to all elements in the group $\mathbb{G}$ to get a table $\mathbf{B}$ of size m where each bin in this table contains elements of $\mathbb{G}$ and items in all bins are sorted in some canonical order. Then Gen uses Cuckoo hashing and distributes each entry $(\mathbf{p_0^i} \otimes \mathbf{p_1^j})[k]$ of $\mathbf{p_0^i} \otimes \mathbf{p_1^j}$ to only one bin (denote l_k) of a table $\mathbf{T}$ of size $m = O(t^2)$ while in simple hashing this entry is inserted to K bins $\{i_1, \cdots, i_K\}$ instead of one (note that $l_k \in \{i_1, \cdots, i_K\}$), then

1. For bin l_k, using $\mathsf{DPF}_{|\mathbf{B}_{l_k}|}.\mathsf{Gen}$ to generate keys such that the point function is defined by position in bin l_k where $(\mathbf{p_0^i} \otimes \mathbf{p_1^j})[k]$ is inserted and the value output shared is $(\mathbf{v_0^i} \otimes \mathbf{v_1^j})[k]$,

2. For other bins $\{i_1, \cdots, i_K\} \setminus \{l_k\}$, in the position where $(\mathbf{p_0^i} \otimes \mathbf{p_1^j})[k]$ is inserted, the value shared is 0.

The formal constructions of key generation and key expansion are shown in Figure 5.19 and Figure 5.20 respectively.

Figure 5.19: Seed generation of fQA-SD$_{\mathsf{OLE}}$ based on QA-SD and hashing techniques

PARAMETERS: Security parameter λ, noise weight $t = t(\lambda)$, compression factor $c = 4$, ring $\mathcal{R} = \mathbb{F}_4[X_1, \dots, X_n]/(X_1^3 - 1, \dots, X_n^3 - 1)$.

- An DPF_n scheme $(\mathsf{DPF}_n.\mathsf{Gen}, \mathsf{DPF}_n.\mathsf{FullEval})$ with an arbitrary domain $[n]$ and range $\mathbb{F}_4$.
- K hash functions $h_1, \cdots, h_K : \mathbb{G} \to [m]$.
- Cuckoo hashing and simple hashing schemes with parameters $(\mathbb{G}, K, m, t^2)$ and $(\mathbb{G}, K, m)$ respectively where $m = O(t^2)$.

PUBLIC INPUT: $c - 1$ vectors of length 3^n over $\mathbb{F}_4$, for $a_1, \cdots, a_{c-1} \in \mathcal{R}$.

```
fQA-SD.Gen $(1^\lambda)$:
1: foreach $\sigma \in \{0,1\},\ i \in [0 \dots c)$:
    1.1: Sample random vectors $\mathbf{p_\sigma^i} \leftarrow (p_{\sigma,1}^i, \cdots, p_{\sigma,t}^i)_{p_{\sigma,j}^i \in \mathbb{G}}$ and $\mathbf{v_\sigma^i} \leftarrow (\mathbb{F}_3)^t$.
2: Hashing algorithm:
    2.1: Use simple hashing scheme $(\mathbb{G}, K, m)$ to get a table $\mathbf{B}$ having $m$ bins $(\mathbf{B}_i)_{i \le m}$ such that:
             $$\mathbf{B}_i = \{x \in \mathbb{G} \mid \exists j \in [K] \wedge h_j(x) = i\}.$$
         Each bin is sorted in some canonical order.
    2.2: foreach $i, j \in [0 \dots c)$:
        2.2.1: Use the Cuckoo hashing scheme $(\mathbb{G}, K, m, t^2)$ to insert $\mathbf{p}_{i,j} := \mathbf{p_0^i} \otimes \mathbf{p_1^j}$ to table $\mathbf{T}_{i,j}$.
               We denote each $k$-th entry value $\mathbf{p}_{i,j}[k]$ of vector $\mathbf{p}_{i,j}$ is inserted to the bin $l_k \in [m]$ of
               table $\mathbf{T}_{i,j}$ i.e. $\mathbf{T}_{i,j}[l_k] = \mathbf{p}_{i,j}[k]$ and when considering in bin $\mathbf{B}_{l_k}$ of table $\mathbf{B}$ (obtained
               from simple hashing), denote the position of $\mathbf{p}_{i,j}[k]$ as $r_k$ i.e. $\mathbf{B}_{l_k}[r_k] = \mathbf{p}_{i,j}[k]$.
3: foreach $i, j \in [0 \dots c),\ l_k \in [m]$:
    3.1: If $\mathbf{p}_{i,j}[k] = \mathbf{T}_{i,j}[l_k]$ and $\mathbf{p}_{i,j}[k] = \mathbf{B}_{l_k}[r_k]$, sample DPF keys for each bin $\mathbf{B}_{l_k}$:
             $$(K_{0,l_k}^{i,j}, K_{1,l_k}^{i,j}) \leftarrow \mathsf{DPF}_{|\mathbf{B}_{l_k}|}.\mathsf{Gen}(1^\lambda, r_k, \mathbf{v_0^i} \otimes \mathbf{v_1^j}[k]).$$
    3.2: Otherwise, generate randomly $r_k \leftarrow_R |\mathbf{B}_{l_k}|$ and sample DPF keys:
             $$(K_{0,l_k}^{i,j}, K_{1,l_k}^{i,j}) \leftarrow \mathsf{DPF}_{|\mathbf{B}_{l_k}|}.\mathsf{Gen}(1^\lambda, r_k, 0).$$
4: Let $\mathsf{k}_\sigma = (K_{\sigma,l_k}^{i,j})_{i,j \in [0 \dots c), l_k \in [m]}, ((\mathbf{p_\sigma^i}, \mathbf{v_\sigma^i})_{i \in [0 \dots c)})$.
5: Output $(\mathsf{k}_0, \mathsf{k}_1)$.
```

Theorem 5.8.1. *Let $\mathcal{R} = \mathbb{F}_4[\mathbb{G}] = \mathbb{F}_4[X_1, \dots, X_n]/(X_1^3 - 1, \dots, X_n^3 - 1)$ where $\mathbb{G} = \prod_{i=1}^n \mathbb{Z}/3\mathbb{Z}$. Assume that DPF_n is a secure FSS scheme for point function with domain n and that the $\mathsf{QA\text{-}SD}(c, t, \mathbb{G})$ assumption holds. Then, there exists a generic construction scheme to construct a PCG to produce one OLE correlation (described in Figure 5.19 and Figure 5.20). Using the DPF [BGI16] based on a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ and Cuckoo hashing scheme parameters $(\mathbb{G}, K, m, t^2)$ then we obtain:*

Figure 5.20: Seed expansion of fQA-SD$_{\mathsf{OLE}}$ based on QA-SD and hashing techniques

```
fQA-SD.Expand $(\sigma, \mathsf{k}_\sigma)$:
1: Parse $\mathsf{k}_\sigma$ as $((K^{i,j}_{\sigma,l_k})_{i,j\in[0\dots c),l_k\in[m]}, ((\mathbf{p}^i_{\boldsymbol{\sigma}}, \mathbf{v}^{\boldsymbol{i}}_{\boldsymbol{\sigma}})_{i\in[0\dots c)})$.
2: foreach $i \in [0\dots c)$:
    2.1: Define the element of $\mathcal{R}_t$:
         $e^i_\sigma = \sum_{j\in[0\dots t)} \mathbf{v}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}[j] \cdot \mathbf{p}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}[j]$.
3: Compute $x_\sigma = \langle \mathbf{a}, \mathbf{e}_{\boldsymbol{\sigma}} \rangle$, where $\mathbf{a} = (1, a_1, \cdots, a_{c-1}), \mathbf{e}_{\boldsymbol{\sigma}} = (e^0_\sigma, \cdots, e^{c-1}_\sigma)$.
4: foreach $i, j \in [0\dots c)$:
    4.1: $\forall k \in [m]$, compute $\mathbf{w}_{\sigma,k} \leftarrow \mathsf{DPF}_{|\mathbf{B}_k|}.\mathsf{FullEval}(\sigma, K^{i,j}_{\sigma,k})$.
    4.2: foreach $k \in |\mathbb{G}|$:
        4.2.1: $\forall l \in [1, K]$, compute $h_l(k) = l_k$, then find the bin $r_k$ of $k$ in bin $\mathbf{B}_{l_k}$ i.e., $k = \mathbf{B}_{l_k}[r_k]$.
        4.2.2: Define a vector $\mathbf{u}_{\sigma,i+cj}$ over $\mathcal{R}$ such that $\mathbf{u}_{\sigma,i+cj}[k] = \sum_{l=1}^K \mathbf{w}_{\sigma,l_k}[r_k]$.
    4.3: View the set of $\mathbf{u}_{\sigma,i+cj}$ as a $c^2$ vector $\mathbf{u}_{\boldsymbol{\sigma}}$ of element in $\mathcal{R}$.
5: Compute $z_\sigma = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_{\boldsymbol{\sigma}} \rangle$.
6: Output $x_\sigma, z_\sigma$.
```

- *Each party's seed has maximum size:* $c^2 \cdot (0.9m) \cdot ((\log(3)n - \log t + 1) \cdot (\lambda + 2) + \lambda + 2) + 4t \cdot (\log(3)n + 2)$ *bits.*
- *The computation of* Expand *can be done with at most* $(2 + \lfloor 2/\lambda \rfloor) \cdot (K3^n) \cdot c^2$ *PRG operations, and* $O(c^2 \cdot (\log(3)n) \cdot 3^n)$ *operations in* $\mathbb{F}_4$.

Proof sketch. The security of our construction is based on the security of FSS scheme and is followed the same as in [BCG+20b; BCC+23] since (1) The parameter of the simple hashing scheme is public then both parties can self-compute the table $\mathbf{B}$ obtained from this scheme (2) and $\mathbf{p}^i_0 \otimes \mathbf{p}^j_1$ and $\mathbf{v}^i_0 \otimes \mathbf{v}^j_1$ are mapped to the table $\mathbf{T}_{i,j}$ by using Cuckoo hashing scheme later they are inputs of DPF.
Now we argue the correctness and efficiency in turn.

Correctness. First, note that

$$e^i_0 \cdot e^j_1 = \left(\sum_{k\in[0\dots t)} \mathbf{v}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}[k] \cdot \mathbf{p}^{\boldsymbol{i}}_{\boldsymbol{\sigma}}[k] \right) \cdot \left(\sum_{l\in[0\dots t)} \mathbf{v}^{\boldsymbol{j}}_{\boldsymbol{\sigma}}[l] \cdot \mathbf{p}^{\boldsymbol{j}}_{\boldsymbol{\sigma}}[l] \right)$$
$$= \sum_{k,l\in[0\dots t)} \left(\mathbf{v}^{\boldsymbol{i}}_{\mathbf{0}}[k] \cdot \mathbf{v}^{\boldsymbol{j}}_{\mathbf{1}}[l] \right) \cdot \left(\mathbf{p}^{\boldsymbol{i}}_{\mathbf{0}}[k] \cdot \mathbf{p}^{\boldsymbol{j}}_{\mathbf{1}}[l] \right) = \sum_{k\in[0\dots t^2)} (\mathbf{v}^{\boldsymbol{i}}_{\mathbf{0}} \otimes \mathbf{v}^{\boldsymbol{j}}_{\mathbf{1}})[k] \cdot (\mathbf{p}^{\boldsymbol{i}}_{\mathbf{0}} \otimes \mathbf{p}^{\boldsymbol{j}}_{\mathbf{1}})[k].$$

Observe that $\mathbf{u}_{0,i+cj}[k] + \mathbf{u}_{1,i+cj}[k] = \sum_{l=1}^K (\mathbf{w}_{0,l_k}[r_k] + \mathbf{w}_{1,l_k}[r_k])$ then from the correctness of DPF:

1. If $k \in \mathbf{A}_{i,j}$ i.e., k is inserted to table $\mathbf{T}_{i,j}$ using Cuckoo hashing scheme, denote t as the bin of $\mathbf{T}_{i,j}$ where k is inserted then:

$$\mathbf{u}_{0,i+cj}[k] + \mathbf{u}_{1,i+cj}[k] = \mathbf{w}_{0,t}[r_k] + \mathbf{w}_{1,t}[r_k] = (\mathbf{v}^{\boldsymbol{i}}_{\mathbf{0}} \otimes \mathbf{v}^{\boldsymbol{j}}_{\mathbf{1}})[k].$$

2. Otherwise, $k \notin \mathbf{A}_{i,j}$ then $\mathbf{u}_{0,i+cj}[k] + \mathbf{u}_{1,i+cj}[k] = 0$.

Therefore,

$$\mathbf{u}_0 + \mathbf{u}_1 = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{e}_0 \otimes \mathbf{e}_1 \rangle = \langle \mathbf{a}, \mathbf{e}_0 \rangle \cdot \langle \mathbf{a}, \mathbf{e}_1 \rangle = x_0 \cdot x_1.$$

Efficiency. We show how to obtain the party's seed size and the computation cost of Expand in Theorem 5.8.1 by using two optimizations which are tailored on the choice of Cuckoo's parameters (to reduce the number of bins) and regular noise distribution (to reduce a factor t in the number of PRG calls).

The formulas we get are adapted from [BCG+20b; BCC+23] for $|\mathbb{G}| = 3^n$ and followed by our optimizations.

1. Party's seed size, since we have m bins and each bin $(\mathbf{B}_i)_{i\in[m]}$ needs to use a $\mathsf{DPF}_{|\mathbf{B}_i|}$ scheme so in total, we have $(m \cdot c^2)$ pairs of keys $(K^{i,j}_{\sigma,l_k})_{i,j\in[0\ldots c),l_k\in[m]}$ having a size of

$$c^2 \cdot m \cdot ((\log(3)n - \log t + 1) \cdot (\lambda + 2) + \lambda + 2) + 4t \cdot (\log(3)n + 2).$$

   We make an observation that since the number of bins m is defined to avoid the failure of the Cuckoo hashing scheme. However, because the distributed key generation phase is honestly executed, we allow the Cuckoo hashing scheme to fail with an acceptable probability (say 90% of standard experimental failure probability [PSW+18]). In the case of failure, then we repeat the Cuckoo hashing scheme (with a new set of functions) until it succeeds (the number of repetitions is very small in expectation). This leads to an optimization for the number of bins m to be reduced to $0.9m$ while the trade-off in security is reasonable (the adversary knows the distribution of noise sampled has to make sure the Cuckoo hashing succeeds while the number of bins is only $0.9m$). Then, the seed size is reduced roughly to

$$c^2 \cdot (0.9m) \cdot ((\log(3)n - \log t + 1) \cdot (\lambda + 2) + \lambda + 2) + 4t \cdot (\log(3)n + 2).$$

2. For the number of PRG calls in seed expansion, in each bin $(\mathbf{B}_i)_{i\in[m]}$, we make use of a $\mathsf{DPF}_{|\mathbf{B}_i|}$ with domain $|\mathbf{B}_i|$ and from the property of simple hashing scheme $\sum_{i\leq m} |\mathbf{B}_i| = K \cdot |\mathbb{G}|$ then the number of PRG operations is at most $(2 + \lfloor 2/\lambda \rfloor) \cdot (K3^n) \cdot c^2$ (reduced by a factor t from the regular noise distribution).

□

5.8.2 Application of OLE over $\mathbb{F}_4$ to Silent OT Extension

In this section, we show how to convert an $\mathbb{F}_4$-OLE into a random 1-out-of-2 OT in $\mathbb{F}_2$ using a single bit of communication. To explain the observation, let us consider two parties, Alice and Bob, holding respectively $(a, [\![ab]\!]^4_A)$ and $(b, [\![ab]\!]^4_B)$ for a and $b \in \mathbb{F}_4$. We have

$$\begin{aligned} a \cdot b &= [\![a \cdot b]\!]^4_A(0) + [\![a \cdot b]\!]^4_B(0) + ([\![a \cdot b]\!]^4_A(1) + [\![a \cdot b]\!]^4_B(1))\theta \\ &= (a(0) \cdot b(0) + a(1) \cdot b(1)) + (a(0) \cdot b(1) + a(1) \cdot b(0) + a(1) \cdot b(1)) \cdot \theta, \end{aligned}$$

where θ is the primitive root of $X^2 + X + 1$. Considering only the $(a \cdot b)(1)$ term from the above equation (i.e., the parts multiplied by θ, while in conversion from single OLE to Beaver triple over $\mathbb{F}_2$ the part taken is without θ), we get that

$$(a \cdot b)(1) = [\![a \cdot b]\!]^4_A(1) + [\![a \cdot b]\!]^4_B(1) = a(0) \cdot b(1) + a(1) \cdot b(0) + a(1) \cdot b(1),$$

and therefore,

$$\underbrace{[\![a \cdot b]\!]^4_A(1)}_{\text{known by } A} + a(1) \cdot (b(0) + b(1)) = a(0) \cdot b(1) + \underbrace{[\![a \cdot b]\!]^4_B(1)}_{\text{known by } B}.$$

If Bob sends $(b(0) + b(1))$ to Alice then the equation becomes

$$\underbrace{[\![a \cdot b]\!]^4_A(1) + a(1) \cdot (b(0) + b(1))}_{\text{known by } A} = a(0) \cdot b(1) + \underbrace{[\![a \cdot b]\!]^4_B(1)}_{\text{known by } B}.$$

It turns out that if Alice as a receiver in OT sets $t = a(0), m_t = [\![a \cdot b]\!]^4_A(1) + a(1) \cdot (b(0) + b(1))$ and Bob as a sender in OT sets $m_0 = [\![a \cdot b]\!]^4_B(1), m_1 := b(1) + [\![a \cdot b]\!]^4_B(1)$ then we have an instantiation of 1-out-of-2 OT over $\mathbb{F}_2$. The correctness is followed by the above equation and security is straightforward: the only communication between the parties is sending $b(0) + b(1)$ from Bob to Alice, which is a uniform random bit in the view of Alice (from the randomness of OLE).

Efficiency. To get 3^n random OT, our main $\mathsf{QA\text{-}SD_{OLE}}$ PCG needs $2(c^2 \cdot t)$ PRG calls (omitting some common factors) along with a factor-64 speedup from the early termination optimization while $\mathsf{fQA\text{-}SD}$ only needs $(K \cdot c^2)$ PRG calls where $K = 2,\ t = 27$ (see full version [BBC+24] for details) and can be optimized by the same optimizations. Hence, we estimate to get 3^N-OT over $\mathbb{F}_2$, $\mathsf{fQA\text{-}SD}$ can be about $30\times$ faster in terms of computation compared to [RRT23], since the cost of $\mathsf{QA\text{-}SD_{OLE}}$ is essentially on par with that of [RRT23] (see Section 5.3 for detail).

Chapter 6

Conclusion

6.1 Conclusion

This thesis studies Multi-Party Computation (MPC) in the correlated randomness model. We present several contributions centered around Pseudorandom Correlation Generators (PCGs) and Pseudorandom Correlation Functions (PCFs), specifically in the context of Private Set Intersection (PSI), Zero-Knowledge Proofs (ZKPs), and Multi-Party Computation (MPC) for boolean circuit. The contributions revolve around three key areas:

- Enhancing PSI Efficiency. We have developed new protocols for PSI, leveraging vector OLE as a PCG. These protocols offer significant improvements in communication efficiency, particularly when dealing with databases containing small entries. We introduce a semi-honest PSI protocol that integrates subfield vector OLE with hash-based techniques, achieving communication complexity that is independent of the computational security parameter. We also extend this to a malicious setting using the dual execution technique, which ensures that the communication remains unaffected by security parameters. Additionally, we present a maliciously secure PSI protocol in the standard model, utilizing subfield ring-OLE and the ring-LPN assumption. This protocol is highly efficient and batchable, allowing for reused messages to compute intersections across multiple servers.

- Designated-Verifier Zero-Knowledge Proofs (ZKPs). We have introduced a compiler for converting zero-knowledge proofs for SIMD circuits into general circuits efficiently. This compiler improves communication complexity for general circuits by reducing the cost of the state-of-the-art from $\mathcal{O}(C^{3/4})$ to $\mathcal{O}(C^{1/2})$. Additionally, we have demonstrated how a new public-key PCFs, which we introduce, can be used to construct reusable Designated-Verifier Non-Interactive Zero-Knowledge Proofs (DV-NIZKs) for NP. This approach upgrades non-reusable DV-NIZKs into reusable ones, providing efficient public-key PCF-based OT for secure communication between parties.

- FOLEAGE-based MPC for Boolean Circuits. We have proposed $\mathbb{F}_4$OLEAGE, a preprocessing protocol tailored for efficient MPC involving Boolean circuits, providing semi-honest security and tolerating up to $N-1$ corruptions. $\mathbb{F}_4$OLEAGE is highly efficient, generating over 12 million multiplication triples per second in a two-party setting, while minimizing communication during the preprocessing phase. By utilizing PCGs for multiplication triples over the field $\mathbb{F}_4$, we improve the efficiency of the offline phase in MPC protocols. $\mathbb{F}_4$OLEAGE significantly outperforms existing methods in both two-party and multi-party settings, offering faster execution and lower communication overhead.

These contributions, discussed in detail within the thesis, showcase the impact of PCGs and PCFs on optimizing protocols in cryptographic applications, pushing the boundaries of efficiency and security for PSI, ZKPs, and MPC.

6.2 Open Questions

For future research direction, we would like to address some open question as follow.

- **Application of PCGs/PCFs for PSI:**

  1. *Standard PSI.* The current state-of-the-art for standard PSI relies on subfield-vector OLE. Efficiency is crucial for PSI due to its wide range of practical applications in real-world scenarios. Optimizing the efficiency of PCGs/PCFs directly enhances PSI performance. A promising direction is exploring how PCGs/PCFs can be adapted to improve the efficiency of the PSI framework. Additionally, investigating the use of PCGs/PCFs to construct variants of PSI, such as PSI-Sum, PSI-Cardinality, or threshold PSI, presents an exciting avenue for future research.
  2. *Fuzzy PSI.* Fuzzy structured-aware PSI in high dimensions is designed to enable one party to learn the output even with a small fuzzy error. In particular, in fuzzy structured-aware PSI, the receiver holds N balls of radius σ and dimension d and the sender holds a set of M points. In the end, the receiver learns which of the sender's points inside one of their balls. The initial construction of fuzzy structured-aware PSI uses Function Secret Sharing as one of the central techniques to instantiate the construction. We aim to further investigate this area, with a focus on improving efficiency and strengthening security guarantees.

- **Zero-Knowledge Proofs based on PCGs/PCFs:**

  1. *Sublinear ZKP based on VOLE for Circuit Satisfiability.* Our current construction is a private coin ZKP, since the Information-theoretic polynomial authentication code (IP-PAC) is constructed from additive homomorphic encryption (AHE). Turning it into a public coin using techniques such as VOLE-in-the-Head is a possible direction. An open question is whether we can achieve a publicly verifiable ZKP based on PCGs/PCFs with sublinear communication and linear computation complexity, with a small linear factor. Exploring streaming ZK proofs based on PCGs/PCFs is another promising direction.
  2. *DV-NIZK based on Public-Key PCF.* Our DV-NIZK scheme is built from non-interactive OT, instantiated from PK-PCF-based OT. The efficiency of our DV-NIZK relies on the underlying PK-PCF. As a result, further research into PK-PCFs can lead to advancements in DV-NIZK applications. Improving the efficiency of PK-PCFs can be achieved by optimizing the distributed key generation process. In the long term, post-quantum PK-PCFs would be advantageous, which would require key generation and silent expansion to be realized from post-quantum assumptions. Currently, PK-PCFs have only been instantiated for OTs and VOLEs, exploring other correlations, such as OLEs, is a natural next step.

- **MPC-based OLE for Boolean Circuits:** In our FOLEAGE-based MPC for Boolean circuits, we construct PCG-based OLEs in $\mathbb{F}_4$, and subsequently achieve OLEs over $\mathbb{F}_2$ with additional communication. Whether it is possible to achieve PCF-based OLEs in $\mathbb{F}_2$ remains an open question. Answering this question directly will lead to more efficient MPC-based OLE for Boolean circuits.

List of Figures

List of Tables

Bibliography

[AMZ21] Aydin Abadi, Steven J. Murdoch, and Thomas Zacharias. *Polynomial Representation Is Tricky: Maliciously Secure Private Set Intersection Revisited.* Cryptology ePrint Archive, Report 2021/1009. https://ia.cr/2021/1009. 2021.

[ABP15] Michel Abdalla, Fabrice Benhamouda, and Alain Passelègue. "An Algebraic Framework for Pseudorandom Functions and Applications to Related-Key Security". In: *CRYPTO 2015, Part I.* Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 388–409. DOI: 10.1007/978-3-662-47989-6_19.

[ABB+24] Masayuki Abe, David Balbás, Dung Bui, Miyako Ohkubo, Zehua Shang, and Mehdi Tibouchi. *Critical Round in Multi-Round Proofs: Compositions and Transformation to Trapdoor Commitments.* ePrint Archive. Available at https://eprint.iacr.org/2024/252. 2024.

[ADD+23] Martin R. Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. *Crypto Dark Matter on the Torus: Oblivious PRFs from shallow PRFs and FHE.* Cryptology ePrint Archive, Report 2023/232. https://eprint.iacr.org/2023/232. 2023.

[AHI+17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *ACM CCS 2017.* Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 2087–2104. DOI: 10.1145/3133956.3134104.

[AS20] Hossein Amiri and Asadollah Shahbahrami. "SIMD programming using Intel vector extensions". In: *J. Parallel Distrib. Comput.* 135.C (Jan. 2020), pp. 83–100. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2019.09.012.

[ADI+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. "Secure Arithmetic Computation with Constant Computational Overhead". In: *CRYPTO 2017, Part I.* Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 223–254. DOI: 10.1007/978-3-319-63688-7_8.

[AL16] Benny Applebaum and Shachar Lovett. "Algebraic attacks against random local functions and their countermeasures". In: *48th ACM STOC.* Ed. by Daniel Wichs and Yishay Mansour. ACM Press, June 2016, pp. 1087–1100. DOI: 10.1145/2897518.2897554.

[AMN+18] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. "Constrained PRFs for NC^1 in Traditional Groups". In: *CRYPTO 2018, Part II.* Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. LNCS. Springer, Heidelberg, Aug. 2018, pp. 543–574. DOI: 10.1007/978-3-319-96881-0_19.

[BBH+22] Laasya Bangalore, Rishabh Bhadauria, Carmit Hazay, and Muthuramakrishnan Venkitasubramaniam. “On Black-Box Constructions of Time and Space Efficient Sublinear Arguments from Symmetric-Key Primitives”. In: *TCC 2022, Part I.* Ed. by Eike Kiltz and Vinod Vaikuntanathan. Vol. 13747. LNCS. Springer, Heidelberg, Nov. 2022, pp. 417–446. DOI: 10.1007/978-3-031-22318-1_15.

[BGM+20] James Bartusek, Sanjam Garg, Daniel Masny, and Pratyay Mukherjee. “Reusable Two-Round MPC from DDH”. In: *TCC 2020, Part II.* Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12551. LNCS. Springer, Heidelberg, Nov. 2020, pp. 320–348. DOI: 10.1007/978-3-030-64378-2_12.

[BMR+21] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. “Mac’n’Cheese: Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions”. In: *CRYPTO 2021, Part IV.* Ed. by Tal Malkin and Chris Peikert. Vol. 12828. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 92–122. DOI: 10.1007/978-3-030-84259-8_4.

[Bea92] Donald Beaver. “Efficient Multiparty Protocols Using Circuit Randomization”. In: *CRYPTO’91.* Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 420–432. DOI: 10.1007/3-540-46766-1_34.

[Bea96] Donald Beaver. “Correlated Pseudorandomness and the Complexity of Private Computations”. In: *28th ACM STOC.* ACM Press, May 1996, pp. 479–488. DOI: 10.1145/237814.237996.

[BM90] Mihir Bellare and Silvio Micali. “Non-Interactive Oblivious Transfer and Applications”. In: *CRYPTO’89.* Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 547–557. DOI: 10.1007/0-387-34805-0_48.

[Ber06] Daniel J. Bernstein. “Curve25519: New Diffie-Hellman Speed Records”. In: *PKC 2006.* Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Vol. 3958. LNCS. Springer, Heidelberg, Apr. 2006, pp. 207–228. DOI: 10.1007/11745853_14.

[BCC+13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. “Recursive composition and bootstrapping for SNARKS and proof-carrying data”. In: *45th ACM STOC.* Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM Press, June 2013, pp. 111–120. DOI: 10.1145/2488608.2488623.

[BC12] Nir Bitansky and Alessandro Chiesa. “Succinct Arguments from Multi-prover Interactive Proofs and Their Efficiency Benefits”. In: *CRYPTO 2012.* Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 255–272. DOI: 10.1007/978-3-642-32009-5_16.

[BG22] Alexander R. Block and Christina Garman. *Honest Majority Multi-Prover Interactive Arguments.* Cryptology ePrint Archive, Report 2022/557. https://eprint.iacr.org/2022/557. 2022.

[BHR+20] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. “Public-Coin Zero-Knowledge Arguments with (almost) Minimal Time and Space Overheads”. In: *TCC 2020, Part II.* Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12551. LNCS. Springer, Heidelberg, Nov. 2020, pp. 168–197. DOI: 10.1007/978-3-030-64378-2_7.

[BHR+21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. “Time- and Space-Efficient Arguments from Groups of Unknown Order”. In: *CRYPTO 2021, Part IV.* Ed. by Tal Malkin and Chris Peikert. Vol. 12828. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 123–152. DOI: 10.1007/978-3-030-84259-8_5.

[Blu86] Manuel Blum. *How to Prove a Theorem So No One Else Can Claim It.* Invited 45 minute address to the International Congress of Mathematicians, 1986. To appear in the Proceedings of ICM 86. Aug. 1986. URL: http://www.mathunion.org/ICM/ICM1986.2/Main/icm1986.2.1444.1451.ocr.pdf.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. "Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)". In: *20th ACM STOC*. ACM Press, May 1988, pp. 103–112. DOI: 10.1145/62212.62222.

[BCD+09] Peter Bogetoft et al. "Secure Multiparty Computation Goes Live". In: *Financial Cryptography and Data Security*. Ed. by Chris Clifford. Vol. 5628. Lecture Notes in Computer Science. Accessed: 2024-11-10. Springer, Berlin, Heidelberg, 2009, pp. 325–343. DOI: 10.1007/978-3-642-00468-1_2. URL: https://link.springer.com/chapter/10.1007/978-3-642-00468-1_2.

[BBC+24] Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, Clément Ducros, and Sacha Servan-Schreiber. "FOLEAGE: $\mathbb{F}_4$OLE-Based Multi-party Computation for Boolean Circuits". In: *Advances in Cryptology – ASIACRYPT 2024*. Ed. by Kai-Min Chung and Yu Sasaki. Singapore: Springer Nature Singapore, 2024, pp. 69–101. ISBN: 978-981-96-0938-3.

[BCC+23] Maxime Bombar, Geoffroy Couteau, Alain Couvreur, and Clément Ducros. "Correlated Pseudorandomness from the Hardness of Quasi-Abelian Decoding". In: *CRYPTO 2023, Part IV*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14084. LNCS. Springer, Heidelberg, Aug. 2023, pp. 567–601. DOI: 10.1007/978-3-031-38551-3_18.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. "Hierarchical Identity Based Encryption with Constant Size Ciphertext". In: *EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. LNCS. Springer, Heidelberg, May 2005, pp. 440–456. DOI: 10.1007/11426639_26.

[BIP+18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. "Exploring Crypto Dark Matter: New Simple PRF Candidates and Their Applications". In: *TCC 2018, Part II*. Ed. by Amos Beimel and Stefan Dziembowski. Vol. 11240. LNCS. Springer, Heidelberg, Nov. 2018, pp. 699–729. DOI: 10.1007/978-3-030-03810-6_25.

[BCH+22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. "Gemini: Elastic SNARKs for Diverse Environments". In: *EUROCRYPT 2022, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. LNCS. Springer, Heidelberg, May 2022, pp. 427–457. DOI: 10.1007/978-3-031-07085-3_15.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Halo: Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. https://eprint.iacr.org/2019/1021. 2019.

[BCG+18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. "Compressing Vector OLE". In: *ACM CCS 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM Press, Oct. 2018, pp. 896–912. DOI: 10.1145/3243734.3243868.

[BCG+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. "Correlated Pseudorandomness from Expand-Accumulate Codes". In: *CRYPTO 2022, Part II*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13508. LNCS. Springer, Heidelberg, Aug. 2022, pp. 603–633. DOI: 10.1007/978-3-031-15979-4_21.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. "Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation". In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 291–308. DOI: 10.1145/3319535.3354255.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. "Efficient Pseudorandom Correlation Generators: Silent OT Extension and More". In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 489–518. DOI: 10.1007/978-3-030-26954-8_16.

[BCG+20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. "Correlated Pseudorandom Functions from Variable-Density LPN". In: *61st FOCS*. IEEE Computer Society Press, Nov. 2020, pp. 1069–1080. DOI: 10.1109/FOCS46700.2020.00103.

[BCG+20b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. "Efficient Pseudorandom Correlation Generators from Ring-LPN". In: *CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. LNCS. Springer, Heidelberg, Aug. 2020, pp. 387–416. DOI: 10.1007/978-3-030-56880-1_14.

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. "Homomorphic Secret Sharing: Optimizations and Applications". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 2105–2122. DOI: 10.1145/3133956.3134107.

[BCM23] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. "Sublinear-Communication Secure Multiparty Computation Does Not Require FHE". In: *EUROCRYPT 2023, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. LNCS. Springer, Heidelberg, Apr. 2023, pp. 159–189. DOI: 10.1007/978-3-031-30617-4_6.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function secret sharing". In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2015, pp. 337–367.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing: Improvements and Extensions". In: *ACM CCS 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1292–1303. DOI: 10.1145/2976749.2978429.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) Fully Homomorphic Encryption without Bootstrapping". In: vol. 6. 3. New York, NY, USA: Association for Computing Machinery, July 2014. DOI: 10.1145/2633600. URL: https://doi.org/10.1145/2633600.

[BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. "NIZK from LPN and Trapdoor Hash via Correlation Intractability for Approximable Relations". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 738–767. DOI: 10.1007/978-3-030-56877-1_26.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. "Constrained Key-Homomorphic PRFs from Standard Lattice Assumptions - Or: How to Secretly Embed a Circuit in Your PRF". In: *TCC 2015, Part II*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 1–30. DOI: 10.1007/978-3-662-46497-7_1.

[BCP03] Emmanuel Bresson, Dario Catalano, and David Pointcheval. “A Simple Public-Key Cryptosystem with a Double Trapdoor Decryption Mechanism and Its Applications”. In: *ASIACRYPT 2003*. Ed. by Chi-Sung Laih. Vol. 2894. LNCS. Springer, Heidelberg, Nov. 2003, pp. 37–54. DOI: 10.1007/978-3-540-40061-5_3.

[BCE+23] Chris Brzuska, Geoffroy Couteau, Christoph Egger, Pihla Karanko, and Pierre Meyer. *New Random Oracle Instantiations from Extremely Lossy Functions*. Cryptology ePrint Archive, Paper 2023/1145. https://eprint.iacr.org/2023/1145. 2023. URL: https://eprint.iacr.org/2023/1145.

[Bui25] Dung Bui. “Efficient Multi-instance Vector Commitment and Application to Post-quantum Signatures”. In: *Information Security and Privacy – ACISP 2025*. Springer Nature Singapore, 2025. URL: https://eprint.iacr.org/2024/254.

[BCC+25] Dung Bui, Eliana Carozza, Geoffroy Couteau, Dahmun Goudarzi, and Antoine Joux. “Faster Signatures from MPC-in-the-Head”. In: *Advances in Cryptology – ASIACRYPT 2024*. Ed. by Kai-Min Chung and Yu Sasaki. Singapore: Springer Nature Singapore, 2025, pp. 396–428. ISBN: 978-981-96-0875-1.

[BCC+24] Dung Bui, Haotian Chu, Geoffroy Couteau, Xiao Wang, Chenkai Weng, Kang Yang, and Yu Yu. “An Efficient ZK Compiler from SIMD Circuits to General Circuits”. In: *Journal of Cryptology* 38.1 (Dec. 2024), p. 10. ISSN: 1432-1378. DOI: 10.1007/s00145-024-09531-4. URL: https://doi.org/10.1007/s00145-024-09531-4.

[BCS24] Dung Bui, Kelong Cong, and Cyprien Delpech de Saint Guilhem. *Improved All-but-One Vector Commitment with Applications to Post-Quantum Signatures*. ePrint Archive. Available at https://eprint.iacr.org/2024/255. 2024.

[BC23] Dung Bui and Geoffroy Couteau. “Improved Private Set Intersection for Sets with Small Entries”. In: *PKC 2023, Part II*. Ed. by Alexandra Boldyreva and Vladimir Kolesnikov. Vol. 13941. LNCS. Springer, Heidelberg, May 2023, pp. 190–220. DOI: 10.1007/978-3-031-31371-4_7.

[BCM24] Dung Bui, Geoffroy Couteau, and Nikolas Melissaris. *Structured-Seed Local Pseudorandom Generators and their Applications*. ePrint Archive. Available at https://eprint.iacr.org/2024/253. 2024.

[BCM+24] Dung Bui, Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. “Fast Public-Key Silent OT and More from Constrained Naor-Reingold”. In: *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*. Ed. by Marc Joye and Gregor Leander. Vol. 14656. Lecture Notes in Computer Science. Springer, 2024, pp. 88–118. DOI: 10.1007/978-3-031-58751-1_4.

[BCL+21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. “Proof-Carrying Data Without Succinct Arguments”. In: *CRYPTO 2021, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 681–710. DOI: 10.1007/978-3-030-84242-0_24.

[CNs07] Jan Camenisch, Gregory Neven, and abhi shelat. “Simulatable Adaptive Oblivious Transfer”. In: *EUROCRYPT 2007*. Ed. by Moni Naor. Vol. 4515. LNCS. Springer, Heidelberg, May 2007, pp. 573–590. DOI: 10.1007/978-3-540-72540-4_33.

[CFF+21] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. "Lunar: A Toolbox for More Efficient Universal and Updatable zkSNARKs and Commit-and-Prove Extensions". In: *ASIACRYPT 2021, Part III*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13092. LNCS. Springer, Heidelberg, Dec. 2021, pp. 3–33. DOI: 10.1007/978-3-030-92078-4_1.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. "LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs". In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2075–2092. DOI: 10.1145/3319535.3339820.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[CC18] Pyrros Chaidos and Geoffroy Couteau. "Efficient Designated-Verifier Non-interactive Zero-Knowledge Proofs of Knowledge". In: *EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, Apr. 2018, pp. 193–221. DOI: 10.1007/978-3-319-78372-7_7.

[CDI+19] Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. "Reusable Non-Interactive Secure Computation". In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 462–488. DOI: 10.1007/978-3-030-26954-8_15.

[CM20] Melissa Chase and Peihan Miao. "Private Set Intersection in the Internet Setting from Lightweight Oblivious PRF". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 34–63. DOI: 10.1007/978-3-030-56877-1_2.

[CBB+23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates". In: *EUROCRYPT 2023, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. LNCS. Springer, Heidelberg, Apr. 2023, pp. 499–530. DOI: 10.1007/978-3-031-30617-4_17.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. "Fast Private Set Intersection from Homomorphic Encryption". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1243–1255. DOI: 10.1145/3133956.3134061.

[CCK+21] Jung Hee Cheon, Wonhee Cho, Jeong Han Kim, and Jiseung Kim. "Adventures in Crypto Dark Matter: Attacks and Fixes for Weak Pseudorandom Functions". In: *PKC 2021, Part II*. Ed. by Juan Garay. Vol. 12711. LNCS. Springer, Heidelberg, May 2021, pp. 739–760. DOI: 10.1007/978-3-030-75248-4_26.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Heidelberg, May 2020, pp. 738–768. DOI: 10.1007/978-3-030-45721-1_26.

[CT65] James W. Cooley and John W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series". In: *Math. Comput.* 19 (1965), pp. 297–301. DOI: 10.1090/S0025-5718-1965-0178586-1.

[CL15] Craig Costello and Patrick Longa. "Four$\mathbb{Q}$: Four-Dimensional Decompositions on a $\mathbb{Q}$-curve over the Mersenne Prime". In: *ASIACRYPT 2015, Part I.* Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9452. LNCS. Springer, Heidelberg, Nov. 2015, pp. 214–235. DOI: 10.1007/978-3-662-48797-6_10.

[CD23] Geoffroy Couteau and Clément Ducros. "Pseudorandom Correlation Functions from Variable-Density LPN, Revisited". In: *PKC 2023, Part II.* Ed. by Alexandra Boldyreva and Vladimir Kolesnikov. Vol. 13941. LNCS. Springer, Heidelberg, May 2023, pp. 221–250. DOI: 10.1007/978-3-031-31371-4_8.

[CDM+18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. "On the Concrete Security of Goldreich's Pseudorandom Generator". In: *ASIACRYPT 2018, Part II.* Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. LNCS. Springer, Heidelberg, Dec. 2018, pp. 96–124. DOI: 10.1007/978-3-030-03329-3_4.

[CH20] Geoffroy Couteau and Dominik Hartmann. "Shorter Non-interactive Zero-Knowledge Arguments and ZAPs for Algebraic Languages". In: *CRYPTO 2020, Part III.* Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 768–798. DOI: 10.1007/978-3-030-56877-1_27.

[CH19] Geoffroy Couteau and Dennis Hofheinz. "Designated-Verifier Pseudorandom Generators, and Their Applications". In: *EUROCRYPT 2019, Part II.* Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 562–592. DOI: 10.1007/978-3-030-17656-3_20.

[CJJ+23] Geoffroy Couteau, Abhishek Jain, Zhengzhong Jin, and Willy Quach. "A Note on Non-interactive Zero-Knowledge from CDH". In: *CRYPTO 2023, Part IV.* Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14084. LNCS. Springer, Heidelberg, Aug. 2023, pp. 731–764. DOI: 10.1007/978-3-031-38551-3_23.

[CKL+21] Geoffroy Couteau, Michael Klooß, Huang Lin, and Michael Reichle. "Efficient Range Proofs with Transparent Setup from Bounded Integer Commitments". In: *EUROCRYPT 2021, Part III.* Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12698. LNCS. Springer, Heidelberg, Oct. 2021, pp. 247–277. DOI: 10.1007/978-3-030-77883-5_9.

[CMP+23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. "Constrained Pseudorandom Functions from Homomorphic Secret Sharing". In: *EUROCRYPT 2023, Part III.* Ed. by Carmit Hazay and Martijn Stam. Vol. 14006. LNCS. Springer, Heidelberg, Apr. 2023, pp. 194–224. DOI: 10.1007/978-3-031-30620-4_7.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. "Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes". In: *CRYPTO 2021, Part III.* Ed. by Tal Malkin and Chris Peikert. Vol. 12827. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 502–534. DOI: 10.1007/978-3-030-84252-9_17.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. "Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols". In: *CRYPTO'94.* Ed. by Yvo Desmedt. Vol. 839. LNCS. Springer, Heidelberg, Aug. 1994, pp. 174–187. DOI: 10.1007/3-540-48658-5_19.

[CHH+07] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, abhi shelat, and Vinod Vaikuntanathan. "Bounded CCA2-Secure Encryption". In: *ASIACRYPT 2007.* Ed. by Kaoru Kurosawa. Vol. 4833. LNCS. Springer, Heidelberg, Dec. 2007, pp. 502–518. DOI: 10.1007/978-3-540-76900-2_31.

[CS02] Ronald Cramer and Victor Shoup. “Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption”. In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Heidelberg, Apr. 2002, pp. 45–64. DOI: 10.1007/3-540-46035-7_4.

[DMR+09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. “Efficient Robust Private Set Intersection”. In: *ACNS 09*. Ed. by Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud. Vol. 5536. LNCS. Springer, Heidelberg, June 2009, pp. 125–142. DOI: 10.1007/978-3-642-01957-9_8.

[DNN+17] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. “The TinyTable Protocol for 2-Party Secure Computation, or: Gate-Scrambling Revisited”. In: *CRYPTO 2017, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 167–187. DOI: 10.1007/978-3-319-63688-7_6.

[DPS+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. “Multiparty Computation from Somewhat Homomorphic Encryption”. In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[DV21] Amit Daniely and Gal Vardi. “From Local Pseudorandom Generators to Hardness of Learning”. In: *Proceedings of Thirty Fourth Conference on Learning Theory*. Ed. by Mikhail Belkin and Samory Kpotufe. Vol. 134. Proceedings of Machine Learning Research. PMLR, 15–19 Aug 2021, pp. 1358–1394. URL: https://proceedings.mlr.press/v134/daniely21a.html.

[DKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. “Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model”. In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 213–231. DOI: 10.1007/978-3-642-17373-8_13.

[DW07] Martin Dietzfelbinger and Christoph Weidling. “Balanced allocation and dictionaries with tightly packed constant size bins”. In: *Theoretical Computer Science* 380.1-2 (2007), pp. 47–68.

[DGH+21] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. “MPC-Friendly Symmetric Cryptography from Alternating Moduli: Candidates, Protocols, and Applications”. In: *CRYPTO 2021, Part IV*. Ed. by Tal Malkin and Chris Peikert. Vol. 12828. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 517–547. DOI: 10.1007/978-3-030-84259-8_18.

[DIO20] Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. *Line-Point Zero Knowledge and Its Applications*. Cryptology ePrint Archive, Report 2020/1446. https://eprint.iacr.org/2020/1446. 2020.

[Ds17] Jack Doerner and abhi shelat. “Scaling ORAM for Secure Computation”. In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 523–535. DOI: 10.1145/3133956.3133967.

[DMR23] Aurélien Dupin, Pierrick Méaux, and Mélissa Rossi. “On the algebraic immunity—resiliency trade-off, implications for Goldreich’s pseudorandom generator”. In: *Designs, Codes and Cryptography* (2023), pp. 1–45.

[DMM+21] Sébastien Duval, Pierrick Méaux, Charles Momin, and François-Xavier Standaert. “Exploring Crypto-Physical Dark Matter and Learning with Physical Rounding”. In: *IACR TCHES* 2021.1 (2021). https://tches.iacr.org/index.php/TCHES/article/view/8738, pp. 373–401. ISSN: 2569-2925. DOI: 10.46586/tches.v2021.i1.373-401.

[EFK+20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. “SPARKs: Succinct Parallelizable Arguments of Knowledge”. In: *EUROCRYPT 2020, Part I.* Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Heidelberg, May 2020, pp. 707–737. DOI: 10.1007/978-3-030-45721-1_25.

[EGL82] Shimon Even, Oded Goldreich, and Abraham Lempel. “A Randomized Protocol for Signing Contracts”. In: *CRYPTO’82.* Ed. by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Plenum Press, New York, USA, 1982, pp. 205–210.

[FHN+16] Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. “Efficient Set Intersection with Simulation-Based Security”. In: *Journal of Cryptology* 29.1 (Jan. 2016), pp. 115–155. DOI: 10.1007/s00145-014-9190-0.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. “Efficient Private Matching and Set Intersection”. In: *EUROCRYPT 2004.* Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 1–19. DOI: 10.1007/978-3-540-24676-3_1.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge.* Cryptology ePrint Archive, Report 2019/953. https://eprint.iacr.org/2019/953. 2019.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. “Oblivious Key-Value Stores and Amplification for Private Set Intersection”. In: *CRYPTO 2021, Part II.* Ed. by Tal Malkin and Chris Peikert. Vol. 12826. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 395–425. DOI: 10.1007/978-3-030-84245-1_14.

[GGP+13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. “Quadratic Span Programs and Succinct NIZKs without PCPs”. In: *EUROCRYPT 2013.* Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 626–645. DOI: 10.1007/978-3-642-38348-9_37.

[GN19] Satrajit Ghosh and Tobias Nilges. “An Algebraic Approach to Maliciously Secure Private Set Intersection”. In: *EUROCRYPT 2019, Part III.* Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. LNCS. Springer, Heidelberg, May 2019, pp. 154–185. DOI: 10.1007/978-3-030-17659-4_6.

[GS19] Satrajit Ghosh and Mark Simkin. “The Communication Complexity of Threshold Private Set Intersection”. In: *CRYPTO 2019, Part II.* Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11693. LNCS. Springer, Heidelberg, Aug. 2019, pp. 3–29. DOI: 10.1007/978-3-030-26951-7_1.

[Gil99] Niv Gilboa. “Two Party RSA Key Generation”. In: *CRYPTO’99.* Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 116–129. DOI: 10.1007/3-540-48405-1_8.

[GI14] Niv Gilboa and Yuval Ishai. “Distributed point functions and their applications”. In: *Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33.* Springer. 2014, pp. 640–658.

[Gol00] Oded Goldreich. *Candidate One-Way Functions Based on Expander Graphs.* Cryptology ePrint Archive, Report 2000/063. https://eprint.iacr.org/2000/063. 2000.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to Construct Random Functions (Extended Abstract)". In: *25th FOCS.* IEEE Computer Society Press, Oct. 1984, pp. 464–479. DOI: 10.1109/SFCS.1984.715949.

[GGM19] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to construct random functions". In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali.* New York, NY, USA: Association for Computing Machinery, 2019, pp. 241–264. ISBN: 9781450372664. URL: https://doi.org/10.1145/3335741.3335752.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *19th ACM STOC.* Ed. by Alfred Aho. ACM Press, May 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[GO94] Oded Goldreich and Yair Oren. "Definitions and properties of zero-knowledge proof systems". In: *J. Cryptol.* 7.1 (Dec. 1994), pp. 1–32. ISSN: 0933-2790. DOI: 10.1007/BF00195207. URL: https://doi.org/10.1007/BF00195207.

[Goo23] Google Research. *Don't be Dense: Efficient Keyword PIR for Sparse Databases.* Accessed: 2024-11-10. 2023. URL: https://eprint.iacr.org/2023/466.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. "Non-interactive Zaps and New Techniques for NIZK". In: *CRYPTO 2006.* Ed. by Cynthia Dwork. Vol. 4117. LNCS. Springer, Heidelberg, Aug. 2006, pp. 97–111. DOI: 10.1007/11818175_6.

[GS08] Jens Groth and Amit Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups". In: *EUROCRYPT 2008.* Ed. by Nigel P. Smart. Vol. 4965. LNCS. Springer, Heidelberg, Apr. 2008, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24.

[GYW+23] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. "Half-tree: Halving the cost of tree expansion in COT and DPF". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer. 2023, pp. 330–362.

[Haz15] Carmit Hazay. "Oblivious Polynomial Evaluation and Secure Set-Intersection from Algebraic PRFs". In: *TCC 2015, Part II.* Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 90–120. DOI: 10.1007/978-3-662-46497-7_4.

[HL10] Carmit Hazay and Yehuda Lindell. *A Note on the Relation between the Definitions of Security for Semi-Honest and Malicious Adversaries.* Cryptology ePrint Archive, Report 2010/551. https://eprint.iacr.org/2010/551. 2010.

[HN10] Carmit Hazay and Kobbi Nissim. "Efficient Set Operations in the Presence of Malicious Adversaries". In: *PKC 2010.* Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. LNCS. Springer, Heidelberg, May 2010, pp. 312–331. DOI: 10.1007/978-3-642-13013-7_19.

[HOS+18] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. "Concretely Efficient Large-Scale MPC with Active Security (or, TinyKeys for TinyOT)". In: *ASIACRYPT 2018, Part III.* Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11274. LNCS. Springer, Heidelberg, Dec. 2018, pp. 86–117. DOI: 10.1007/978-3-030-03332-3_4.

[HV17] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. "Scalable Multi-party Private Set-Intersection". In: *PKC 2017, Part I.* Ed. by Serge Fehr. Vol. 10174. LNCS. Springer, Heidelberg, Mar. 2017, pp. 175–203. DOI: 10.1007/978-3-662-54365-8_8.

[HKL+12] Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. "Lapin: An Efficient Authentication Protocol Based on Ring-LPN". In: *FSE 2012*. Ed. by Anne Canteaut. Vol. 7549. LNCS. Springer, Heidelberg, Mar. 2012, pp. 346–365. DOI: 10.1007/978-3-642-34047-5_20.

[HR18] Justin Holmgren and Ron Rothblum. "Delegating Computations with (Almost) Minimal Time and Space Overhead". In: *59th FOCS*. Ed. by Mikkel Thorup. IEEE Computer Society Press, Oct. 2018, pp. 124–135. DOI: 10.1109/FOCS.2018.00021.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. "Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?" In: *NDSS 2012*. The Internet Society, Feb. 2012.

[HFH99] Bernardo A. Huberman, Matt Franklin, and Tad Hogg. "Enhancing Privacy and Trust in Electronic Communities". In: *Proceedings of the 1st ACM Conference on Electronic Commerce*. EC '99. Denver, Colorado, USA: Association for Computing Machinery, 1999, pp. 78–86. ISBN: 1581131763. DOI: 10.1145/336992.337012. URL: https://doi.org/10.1145/336992.337012.

[IBM21] IBM Research. *IBM/TSS: Threshold signature schemes made simple to use*. Accessed: 2024-11-10. 2021. URL: https://github.com/IBM/TSS.

[IKN+03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. "Extending Oblivious Transfers Efficiently". In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 145–161. DOI: 10.1007/978-3-540-45146-4_9.

[IKO+08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. "Cryptography with constant computational overhead". In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 433–442. DOI: 10.1145/1374376.1374438.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. "Indistinguishability obfuscation from well-founded assumptions". In: *53rd ACM STOC*. Ed. by Samir Khuller and Virginia Vassilevska Williams. ACM Press, June 2021, pp. 60–73. DOI: 10.1145/3406325.3451093.

[JJ21] Abhishek Jain and Zhengzhong Jin. "Non-interactive Zero Knowledge from Sub-exponential DDH". In: *EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. LNCS. Springer, Heidelberg, Oct. 2021, pp. 3–32. DOI: 10.1007/978-3-030-77870-5_1.

[JL10] Stanislaw Jarecki and Xiaomin Liu. "Fast Secure Computation of Set Intersection". In: *SCN 10*. Ed. by Juan A. Garay and Roberto De Prisco. Vol. 6280. LNCS. Springer, Heidelberg, Sept. 2010, pp. 418–435. DOI: 10.1007/978-3-642-15317-4_26.

[JMN23] Thomas Johansson, Willi Meier, and Vu Nguyen. "Differential cryptanalysis of Mod-2/Mod-3 constructions of binary weak PRFs". In: *2023 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2023, pp. 477–482.

[KNY+19] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. "Designated Verifier/Prover and Preprocessing NIZKs from Diffie-Hellman Assumptions". In: *EUROCRYPT 2019, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 622–651. DOI: 10.1007/978-3-030-17656-3_22.

[Kel20] Marcel Keller. "MP-SPDZ: A versatile framework for multi-party computation". In: *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 2020, pp. 1575–1590.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer". In: *ACM CCS 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 830–842. DOI: 10.1145/2976749.2978357.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. "Overdrive: Making SPDZ Great Again". In: *EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, Apr. 2018, pp. 158–189. DOI: 10.1007/978-3-319-78372-7_6.

[KW15] Eike Kiltz and Hoeteck Wee. "Quasi-Adaptive NIZK for Linear Subspaces Revisited". In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 101–128. DOI: 10.1007/978-3-662-46803-6_4.

[KS05] Lea Kissner and Dawn Xiaodong Song. "Privacy-Preserving Set Operations". In: *CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. LNCS. Springer, Heidelberg, Aug. 2005, pp. 241–257. DOI: 10.1007/11535218_15.

[KKR+16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. "Efficient Batched Oblivious PRF with Applications to Private Set Intersection". In: *ACM CCS 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 818–829. DOI: 10.1145/2976749.2978381.

[KRT+19] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. "Scalable Private Set Union from Symmetric-Key Techniques". In: *ASIACRYPT 2019, Part II*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11922. LNCS. Springer, Heidelberg, Dec. 2019, pp. 636–666. DOI: 10.1007/978-3-030-34621-8_23.

[KS08] Vladimir Kolesnikov and Thomas Schneider. "Improved garbled circuit: Free XOR gates and applications". In: *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35*. Springer. 2008, pp. 486–498.

[KS22] Abhiram Kothapalli and Srinath Setty. *SuperNova: Proving universal machine executions without universal circuits*. Cryptology ePrint Archive, Report 2022/1758. https://eprint.iacr.org/2022/1758. 2022.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes". In: *CRYPTO 2022, Part IV*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13510. LNCS. Springer, Heidelberg, Aug. 2022, pp. 359–388. DOI: 10.1007/978-3-031-15985-5_13.

[Kul24] Dalvinder Kular. *JPMorgan partners with fraud prevention platform*. 2024. URL: https://www.fstech.co.uk/fst/JPMorgan_Partners_With_Fraud_Prevention_Platform.php.

[KD08] Reinhard Kutzelnigg and Michael Drmota. *Random bipartite graphs and their application to Cuckoo Hashing*. na, 2008.

[Lip16] Helger Lipmaa. "Prover-Efficient Commit-and-Prove Zero-Knowledge SNARKs". In: *AFRICACRYPT 16*. Ed. by David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 9646. LNCS. Springer, Heidelberg, Apr. 2016, pp. 185–206. DOI: 10.1007/978-3-319-31517-1_10.

[LWY+22] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. *The Hardness of LPN over Any Integer Ring and Field for PCG Applications*. Cryptology ePrint Archive, Report 2022/712. https://eprint.iacr.org/2022/712. 2022.

[LQR+19] Alex Lombardi, Willy Quach, Ron D. Rothblum, Daniel Wichs, and David J. Wu. "New Constructions of Reusable Designated-Verifier NIZKs". In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 670–700. DOI: 10.1007/978-3-030-26954-8_22.

[Méa] P Méaux. *On the fast algebraic immunity of threshold functions. Crypt. Commun. 13 (5), 741–762 (2021).*

[Méa22] Pierrick Méaux. "On the algebraic immunity of direct sum constructions". In: *Discrete Applied Mathematics* 320 (2022), pp. 223–234.

[NR95] Moni Naor and Omer Reingold. "Synthesizers and Their Application to the Parallel Construction of Pseudo-Random Functions". In: *36th FOCS*. IEEE Computer Society Press, Oct. 1995, pp. 170–181. DOI: 10.1109/SFCS.1995.492474.

[NR97] Moni Naor and Omer Reingold. "Number-theoretic Constructions of Efficient Pseudo-random Functions". In: *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 458–467. DOI: 10.1109/SFCS.1997.646134.

[Nat+20] S. Nathan et al. *Enigma: Decentralized Computation Platform with Guaranteed Privacy*. Tech. rep. Initiative for Cryptocurrencies and Contracts (IC3), 2020. URL: https://web.media.mit.edu/~guyz/data/enigma_full.pdf.

[Obe07] Ulrich Oberst. "The fast Fourier transform". In: *SIAM journal on control and optimization* 46.2 (2007), pp. 496–540.

[Ope] OpenSSL Project. *OpenSSL Cryptography and SSL/TLS Toolkit*. https://www.openssl.org/. Accessed: 2024-02-12.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. "The Rise of Paillier: Homomorphic Secret Sharing and Public-Key Silent OT". In: *EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. LNCS. Springer, Heidelberg, Oct. 2021, pp. 678–708. DOI: 10.1007/978-3-030-77870-5_24.

[OOS17] Michele Orrù, Emmanuela Orsini, and Peter Scholl. "Actively Secure 1-out-of-N OT Extension with Application to Private Set Intersection". In: *CT-RSA 2017*. Ed. by Helena Handschuh. Vol. 10159. LNCS. Springer, Heidelberg, Feb. 2017, pp. 381–396. DOI: 10.1007/978-3-319-52153-4_22.

[OB22] Alex Ozdemir and Dan Boneh. "Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets". In: *USENIX Security 2022*. Ed. by Kevin R. B. Butler and Kurt Thomas. USENIX Association, Aug. 2022, pp. 4291–4308.

[PR04] Rasmus Pagh and Flemming Friche Rodler. "Cuckoo hashing". In: *Journal of Algorithms* 51.2 (2004), pp. 122–144.

[Pai99] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[PsV06] Rafael Pass, abhi shelat, and Vinod Vaikuntanathan. "Construction of a Non-malleable Encryption Scheme from Any Semantically Secure One". In: *CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. LNCS. Springer, Heidelberg, Aug. 2006, pp. 271–289. DOI: 10.1007/11818175_16.

[Pat04] Boaz Patt-Shamir. "A note on efficient aggregate queries in sensor networks". In: *23rd ACM PODC*. Ed. by Soma Chaudhuri and Shay Kutten. ACM, July 2004, pp. 283–289. DOI: 10.1145/1011767.1011809.

[Ped92] Torben P. Pedersen. “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing”. In: *CRYPTO’91.* Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 129–140. DOI: 10.1007/3-540-46766-1_9.

[PS19] Chris Peikert and Sina Shiehian. “Noninteractive Zero Knowledge for NP from (Plain) Learning with Errors”. In: *CRYPTO 2019, Part I.* Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. LNCS. Springer, Heidelberg, Aug. 2019, pp. 89–114. DOI: 10.1007/978-3-030-26948-7_4.

[PRT+19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. “SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension”. In: *CRYPTO 2019, Part III.* Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 401–431. DOI: 10.1007/978-3-030-26954-8_13.

[PRT+20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. “PSI from PaXoS: Fast, Malicious Private Set Intersection”. In: *EUROCRYPT 2020, Part II.* Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. LNCS. Springer, Heidelberg, May 2020, pp. 739–767. DOI: 10.1007/978-3-030-45724-2_25.

[PSS+15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. “Phasing: Private Set Intersection Using Permutation-based Hashing”. In: *USENIX Security 2015.* Ed. by Jaeyeon Jung and Thorsten Holz. USENIX Association, Aug. 2015, pp. 515–530.

[PSW+18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. “Efficient Circuit-Based PSI via Cuckoo Hashing”. In: *EUROCRYPT 2018, Part III.* Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, Apr. 2018, pp. 125–157. DOI: 10.1007/978-3-319-78372-7_5.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. “Faster Private Set Intersection Based on OT Extension”. In: *USENIX Security 2014.* Ed. by Kevin Fu and Jaeyeon Jung. USENIX Association, Aug. 2014, pp. 797–812.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. “Scalable private set intersection based on OT extension”. In: *ACM Transactions on Privacy and Security (TOPS)* 21.2 (2018), pp. 1–35.

[QRW19] Willy Quach, Ron D. Rothblum, and Daniel Wichs. “Reusable Designated-Verifier NIZKs for all NP from CDH”. In: *EUROCRYPT 2019, Part II.* Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 593–621. DOI: 10.1007/978-3-030-17656-3_21.

[Rab81] Michael Rabin. “How to exchange secrets by oblivious transfer”. In: *Technical Report TR-81, Harvard University,* (1981).

[RRT23] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. “Expand-Convolute Codes for Pseudorandom Correlation Generators from LPN”. In: *CRYPTO 2023, Part IV.* Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14084. LNCS. Springer, Heidelberg, Aug. 2023, pp. 602–632. DOI: 10.1007/978-3-031-38551-3_19.

[RR22] Peter Rindal and Srinivasan Raghuraman. “Blazing Fast PSI from Improved OKVS and Subfield VOLE”. In: *IACR Cryptol. ePrint Arch.* (2022), p. 320. URL: https://eprint.iacr.org/2022/320.

[RR17] Peter Rindal and Mike Rosulek. "Malicious-Secure Private Set Intersection via Dual Execution". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1229–1242. DOI: 10.1145/3133956.3134044.

[RR] Peter Rindal and Lawrence Roy. *libOTe: an efficient, portable, and easy to use Oblivious Transfer Library*. https://github.com/osu-crypto/libOTe.

[RS21] Peter Rindal and Phillipp Schoppmann. "VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE". In: *EUROCRYPT 2021, Part II*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. LNCS. Springer, Heidelberg, Oct. 2021, pp. 901–930. DOI: 10.1007/978-3-030-77886-6_31.

[RT21a] Mike Rosulek and Ni Trieu. *Compact and Malicious Private Set Intersection for Small Sets*. Cryptology ePrint Archive, Report 2021/1159. https://eprint.iacr.org/2021/1159. 2021.

[RT21b] Mike Rosulek and Ni Trieu. "Compact and Malicious Private Set Intersection for Small Sets". In: *ACM CCS 2021*. Ed. by Giovanni Vigna and Elaine Shi. ACM Press, Nov. 2021, pp. 1166–1181. DOI: 10.1145/3460120.3484778.

[Roy22] Lawrence Roy. "SoftSpokenOT: Quieter OT Extension from Small-Field Silent VOLE in the Minicrypt Model". In: *CRYPTO 2022, Part I*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13507. LNCS. Springer, Heidelberg, Aug. 2022, pp. 657–687. DOI: 10.1007/978-3-031-15802-5_23.

[Ste+24] W. A. Stein et al. *Sage Mathematics Software (Version 10.2)*. http://www.sagemath.org. The Sage Development Team. 2024.

[SVV16] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. "Trinocchio: Privacy-Preserving Outsourcing by Distributed Verifiable Computation". In: *ACNS 16*. Ed. by Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider. Vol. 9696. LNCS. Springer, Heidelberg, June 2016, pp. 346–366. DOI: 10.1007/978-3-319-39555-5_19.

[SGR+19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. "Distributed Vector-OLE: Improved Constructions and Implementation". In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 1055–1072. DOI: 10.1145/3319535.3363228.

[22] *Microsoft SEAL (release 4.0)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Mar. 2022.

[STW23] Srinath Setty, Justin Thaler, and Riad Wahby. *Customizable constraint systems for succinct arguments*. Cryptology ePrint Archive, Paper 2023/552. https://eprint.iacr.org/2023/552. 2023.

[SV14] Ankit Sharma and Jan Vondrák. "Multiway cut, pairwise realizable distributions, and descending thresholds". In: *46th ACM STOC*. Ed. by David B. Shmoys. ACM Press, May 2014, pp. 724–733. DOI: 10.1145/2591796.2591866.

[TLP+17] Sandeep Tamrakar, Jian Liu, Andrew Paverd, Jan-Erik Ekberg, Benny Pinkas, and N. Asokan. "The Circle Game: Scalable Private Membership Test Using Trusted Hardware". In: *ASIACCS 17*. Ed. by Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi. ACM Press, Apr. 2017, pp. 31–44.

[TE76] R. E. Twogood and M. P. Ekstrom. “An Extension of Eklundh’s Matrix Transposition Algorithm and Its Application in Digital Image Processing”. In: *IEEE Trans. Comput.* 25.9 (Sept. 1976), pp. 950–952. ISSN: 0018-9340. DOI: 10.1109/TC.1976.1674721. URL: https://doi.org/10.1109/TC.1976.1674721.

[Üna23a] Akin Ünal. “New Baselines for Local Pseudorandom Number Generators by Field Extensions”. In: *Cryptology ePrint Archive* (2023).

[Üna23b] Akin Ünal. “Worst-Case Subexponential Attacks on PRGs of Constant Degree or Constant Locality”. In: *EUROCRYPT 2023, Part I*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14004. LNCS. Springer, Heidelberg, Apr. 2023, pp. 25–54. DOI: 10.1007/978-3-031-30545-0_2.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. *EMP-toolkit: Efficient MultiParty computation toolkit*. https://github.com/emp-toolkit. 2016.

[WYK+21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. “Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits”. In: *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2021, pp. 1074–1091. DOI: 10.1109/SP40001.2021.00056.

[WYY+22] Chenkai Weng, Kang Yang, Zhaomin Yang, Xiang Xie, and Xiao Wang. “AntMan: Interactive Zero-Knowledge Proofs with Sublinear Communication”. In: *ACM CCS 2022*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. ACM Press, Nov. 2022, pp. 2901–2914. DOI: 10.1145/3548606.3560667.

[Wie+17] Udi Wieder et al. “Hashing, load balancing and multiple choice”. In: *Foundations and Trends® in Theoretical Computer Science* 12.3–4 (2017), pp. 275–379.

[WZC+18] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. “DIZK: A Distributed Zero Knowledge Proof System”. In: *USENIX Security 2018*. Ed. by William Enck and Adrienne Porter Felt. USENIX Association, Aug. 2018, pp. 675–692.

[YGJ+21] Jing Yang, Qian Guo, Thomas Johansson, and Michael Lentmaier. “Revisiting the concrete security of Goldreich’s pseudorandom generator”. In: *IEEE Transactions on Information Theory* 68.2 (2021), pp. 1329–1354.

[YSW+21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. “QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field”. In: *ACM CCS 2021*. Ed. by Giovanni Vigna and Elaine Shi. ACM Press, Nov. 2021, pp. 2986–3001. DOI: 10.1145/3460120.3484556.

[YW22] Kang Yang and Xiao Wang. “Non-interactive Zero-Knowledge Proofs to Multiple Verifiers”. In: *ASIACRYPT 2022, Part III*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13793. LNCS. Springer, Heidelberg, Dec. 2022, pp. 517–546. DOI: 10.1007/978-3-031-22969-5_18.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. “Ferret: Fast Extension for Correlated OT with Small Communication”. In: *ACM CCS 2020*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM Press, Nov. 2020, pp. 1607–1626. DOI: 10.1145/3372297.3417276.

[Yao86] Andrew Chi-Chih Yao. “How to Generate and Exchange Secrets (Extended Abstract)”. In: *27th FOCS*. IEEE Computer Society Press, Oct. 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25.