Better, Faster, Stronger

Improving Security, Efficiency, and Primitives for MPC

Nikolas Melissaris

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark

Better, Faster, Stronger

Improving Security, Efficiency, and Primitives for MPC

A Dissertation
Presented to the Faculty of Natural Sciences
of Aarhus University
in Partial Fulfillment of the Requirements
for the PhD Degree

by
Nikolas Melissaris
November 24, 2024

Abstract

Multiparty computation (MPC) enables parties to collaboratively compute functions on their inputs while preserving privacy. Evaluation of MPC protocols typically revolves around: security guarantees (is the output delivered to everyone, always?), communication complexity (how much information needs to be sent between the parties in order to carry out the desired computation?), and computational assumptions (how simple and/or well studied are the ones used?). This thesis advances MPC by developing tools that improve MPC protocols across multiple axes.

First, we study *identifiable abort*. In a protocol with dishonest majority we can't have strong guarantees like *guaranteed output delivery*. *Identifiable abort* ensures that if an adversary disrupts the computation, honest parties can identify at least one cheater and exclude them from future runs, something that can work as a deterrent to malicious behavior. We propose a new compiler that circumvents the limitations of previous approaches, offering identifiable abort without the need for adaptive security. Our approach leverages lightweight cryptographic tools to achieve an efficient preprocessing phase while maintaining a fast online phase.

Next, we investigate the novel *Friends and Foes (FaF)* security model, which extends traditional malicious adversary models. *FaF security* fixes a modeling problem security definitions: not imposing any limitations on what the honest parties might learn during the protocol. It accomplishes that by safeguarding private inputs from subsets of semi-honest parties in addition to the malicious parties. We present constructions achieving optimal round complexity and corruption thresholds, bridging gaps in prior work.

Pseudorandom Correlation Generators (PCGs) enable low-communication preprocessing by shrinking the correlated randomness needed by a protocol. Towards improving efficiency, we introduce *PCGs for pseudorandom permutations*. Our PCG construction for pseudorandom permutations is the first that outputs non-additive correlations without relying on heavy cryptographic primitives like indistinguishability obfuscation. Furthermore, it has practical applications in anonymous communication and distributed leader elections.

Lastly, we propose *structured-seed local pseudorandom generators (SSLPRGs)*, a weaker (in terms of complexity of assumptions) notion of pseudorandom generators. We use our construction of SSLPRGs in indistinguishability obfuscation (iO), constant-overhead secure computation, sublinear communication protocols, and hardness-of-learning results, pushing the boundaries of what weaker cryptographic assumptions can achieve.

Through these contributions, this thesis lays the groundwork for more secure, efficient, and versatile MPC protocols.

Resumé

Flerpartsberegning (MPC) muliggør, at flere parter kan samarbejde om at beregne funktioner på deres input, samtidig med at deres privatliv bevares. Vurdering af MPC-protokoller fokuserer typisk på: sikkerhedsgarantier (bliver output altid leveret til alle?), kommunikationskompleksitet (hvor meget information skal der udveksles mellem parterne for at gennemføre beregningen?), og beregningsmæssige antagelser (hvor simple og/eller velundersøgte er de antagelser, der anvendes?). Denne afhandling fremmer MPC ved at udvikle værktøjer, der forbedrer protokoller på flere måder.

Først undersøger vi *identificerbar afbrydelse*. I en protokol med et uærligt flertal kan vi ikke garantere, at alle ærlige parter modtager et output. Identificerbar afbrydelse sikrer, at hvis en angriber forstyrrer beregningen, kan de ærlige parter identificere mindst én snyder og udelukke denne fra fremtidige eksekveringer, hvilket kan virke som en afskrækkelse mod ondsindet adfærd. Vi foreslår en ny protokol, der overvinder begrænsningerne i tidligere tilgange ved at tilbyde identificerbar afbrydelse uden behov for adaptiv sikkerhed. Vores metode udnytter lette kryptografiske værktøjer for at opnå en effektiv præbehandlingsfase og samtidig bevare en hurtig onlinefase.

Dernæst undersøger vi den nye *Friends and Foes (FaF)* sikkerhedsmodel, som udvider traditionelle modeller for ondsindede modstandere. *FaF-sikkerhed* løser et modelleringsproblem i de eksisterende sikkerhedsdefinitioner, som ikke pålægger nogen begrænsninger på, hvad ærlige parter kan lære under protokollen. Modellen sikrer private input mod både delmængder af semi-ærlige parter og de ondsindede parter. Vi præsenterer konstruktioner, der opnår optimal runde-kompleksitet og korruptionsgrænser og udfylder huller i tidligere forskning.

Pseudorandom Correlation Generators (PCGs) muliggør lavkommunikations/præbehandling ved at reducere den mængde af korreleret tilfældighed, der er nødvendig i en protokol. For at forbedre effektiviteten introducerer vi *PCGs for pseudo-tilfældige permutationer*. Vores PCG-konstruktion for pseudo-tilfældige permutationer er den første, der genererer ikke-additive korrelationer uden at være afhængig af tunge kryptografiske primitive som uadskilleligheds-obfuskering (iO). Vores konstruktion har desuden praktiske anvendelser inden for anonym kommunikation og distribueret ledervalg.

Endelig foreslår vi *structured-seed local pseudorandom generators (SSLPRGs)*, en svagere version af pseudo-tilfældige generatorer (PRG), med hensyn til antagelsernes kompleksitet. Vores konstruktion af SSLPRGs finder anvendelse i iO, sikker beregning med konstant overhead, sublineære kommunikationsprotokoller og hårdhedsresultater for læring, hvilket skubber grænserne for, hvad svagere kryptografiske antagelser kan opnå.

Med disse bidrag skaber denne afhandling grundlaget for mere sikre, effektive og alsidige MPC-protokoller.

Acknowledgments

As I sit here, writing this section, the rest of my thesis safely tucked away in its technical armor, I find myself mere hours from submission. The protocols have been explained, the proofs are proven, and the LaTeX is... well, as good as it's going to get. Now, finally, I can put aside the formality. This is the part where I can be myself; a little sentimental, a little silly, and very grateful. It's strange, isn't it? I realize that writing a PhD thesis feels a lot like climbing a mountain, except the mountain keeps changing its shape, the summit is perpetually shrouded in fog, and halfway up, someone casually hands you a stack of research papers you're magically expected to read. But if there's one thing I've learned during this climb, it's that no one reaches the peak alone. This thesis is as much a testament to the incredible people who walked beside me, pushed me onward, or caught me when I stumbled, as it is to the work itself. Oh my God, what a strange trip this has been.

First, to the Aarhus Crypto Group, a community so vibrant, it's like stepping into an intellectual greenhouse, where ideas flourish and ambitions grow. You welcomed me with open arms and gave me the most stimulating environment I could have hoped for. Your passion and camaraderie set the gold standard for research groups everywhere. Everyone that visits wants to stay, and everyone that leaves comes back with a confession: "It's good, but it's not Aarhus".

To Claudio for willing to entertain an e-mail from an unknown student and then entertain it again when the first time didn't work out. Also, for making Kelsey feel like home from the very beginning. To Ivan, equal parts myth and man. His calm, steady demeanor across the hallway wasn't just a presence; it was a lesson, a reminder that being exceptional doesn't mean towering over others, it means lifting them up. To Sophia and Divya, for approaching me with a project when I was still clueless, fumbling through the basics. They are the co-authors on my very first published paper. That is a milestone that I'll cherish more than they'll ever know.

To Diego and Rahul: my co-conspirators in turning this PhD into a mix of intellectual pursuit, sitcom-level absurdity, and occasionally, deep Sunday philosophy sessions. Diego, you somehow manage to be the perfect mix of encyclopedic knowledge and unapologetic goofiness. Whether you were flexing your random trivia, dissecting sci-fi plots, or dropping some obscure theorem, you never failed to crack me up or make me think, sometimes both at once. And Rahul, my academic brother and king of the subtle (or not so subtle) roast, you were always there with the perfect quip to keep me grounded (or to mercilessly point out when I was missing something obvious). We've talked smack, solved problems, and somehow stumbled into meaningful conversations about life that I never saw coming. Together, you two made every day an adventure. We built this dynamic where every moment was a mix of trash-talking, unfiltered laughter, and the kind of camaraderie

you just can't fake. Honestly, the balance of chaos and support you both brought into my life made this PhD feel more like an inside joke that only the three of us could ever understand. I can't imagine this journey without you, and I wouldn't want to. Here's to all the memes, the debates, and those strangely profound moments I'll hold onto forever. You made these years much more fun than they had any right to be. Seriously, I owe you both more laughs than I can count.

To Mahak, my ultimate officemate and co-captain of our little ship for three unforgettable years. Sharing that office with you wasn't just about sharing a workspace, it was about creating a sanctuary where we could unload our insecurities, fears, hopes, and dreams. That office was where we talked about everything from life's big mysteries to why is it so gray outside again. You made days better, whether it was through a chat about what we ate, an inside joke, or just silently agreeing that neither of us was in the mood to deal with anything. It wasn't some regular office, it was our office, the one people stopped by because it felt a little lighter, a little warmer, and a whole lot more fun.

To Lennart, the undisputed LaTeX (see what I did there?) wizard whose magic saved me from more disasters than I care to admit. If there were an Olympic event for patience with someone else's "Undefined control sequence" meltdowns, you'd not only win gold but probably have a medal named after you. Truly, your ability to fix my chaos at 11 PM, sometimes with a sigh, most times with a smirk, has been nothing short of heroic. I promise I'll try to keep the late-night emergencies to a minimum. And now, we're heading to Paris together for our postdoc adventures. If these years were the warm-up, I can't wait for the days of genius and nights of baguettes and cheese. Here's to new beginnings, more laughs, and probably more "Lennart, can you fix this?" moments. Paris doesn't know what's about to hit it.

To Damiano and Mark, partners in adventure, and infamous pesto-less dinner. To Marius and Sebastian, who bravely joined me for their first surf lesson. And Sebastian, thank you, not just for the waves but for sharing your fancy coffee. To Simon, my Edinburgh pub buddy, whose commitment to one bar for a full week is the kind of loyalty every beer deserves. To Jakob, a fantastic human being, always ready for coffee breaks, a chat, a hike, or even festivals that he'll eventually realized he didn't like. To Hannah and Wesley, for making board game nights the perfect mix of strategy and laughter. To Katharina and her lattice course. To Mathias and his dancing through the hallways. To Matteo, for finding each other again halfway across the world. Thank you for showing me the best pizza place in Aarhus. To Lance, the oracle. The guy that will drive seasoned researchers to madness with his ability to solve problems in the blink of an eye. To Pierre and his vast knowledge of obscure facts. Here's to our upcoming sushi marathon. To Maciej, the mastermind of Friday bar mischief, for teaching me how great Bailey's with tea can be for the cold winter nights. A special shoutout to Nikolaj for making my Danish abstract not only grammatically correct but even vaguely comprehensible. Tak for alt! To Chris Schwiegelshohn and Christian Pedersen for being my PhD support group. I'm very sorry for the all times I stood you up and had to reschedule.

To Malene, the glue that keeps the Aarhus Crypto Group running like a finely tuned engine. You are the unsung hero who makes it all possible, and I'm endlessly grateful for

everything you do.

Then, there's Peter, my advisor and all-around academic compass. Peter, what can I possibly write here that captures everything you've been to me these past years? Your brilliance is undeniable, your patience superhuman, and your knack for answering even my dumbest questions with a smile? Legendary. Those casual pop-ins with a cheerful "How's it going?", just when I was deep in the weeds, were like lifebuoys keeping me afloat. You didn't just guide this thesis, you guided me. You taught me not just how to do research but how to think, how to question, and, most importantly, how to believe in myself when doubt was louder than reason. Your mentorship was a masterclass not just in academic rigor but in humanity. You've shown me that being a great scientist and being a genuinely good person aren't mutually exclusive, they're essential companions. Peter, if one day I can make even one student feel the way you've made me feel: supported, seen, and capable, I'll consider my academic career a success. Oh, and if I ever knock on a student's door with a "How's it going" I promise to say it with the same sincerity, even if they're halfway through reading sports news and pretending to work. Thank you, Peter, for everything.

To Antigoni. You didn't just help me find my footing when I was teetering on the edge, you built me a whole staircase. You patiently introduced me to this field, in which I wrote this thesis, breaking it down piece by piece. Despite your packed schedule, you carved out time to start projects with me, giving me a lifeline when I had no idea what to do next. Without you, I might never have made it to Aarhus, and for that, I owe you more than words can express. But your kindness didn't stop with me. You've been just as invaluable friend to Kelsey, guiding her on her own academic path with the same generosity and care. Antigoni, you've been a mentor, a lifeline, and a beacon of hope wrapped up in one amazing person. Thank you for everything, from your intellectual generosity to your unwavering support. I'll forever be grateful for the light you've brought to my life.

To Daniel, your dedication during my internship that summer went above and beyond anything I could have expected. You spent so much time guiding me and ensuring that every step of the process was a success. It's not just about the technical skills I gained, it was the sense of having someone in my corner, rooting for me and making sure I didn't falter. In those weeks, I gained a mentor and a friend. Thank you for being the kind of person who invests so much into others, it made all the difference for me.

To my co-authors, this thesis exists literally thanks to your brilliance and collaboration. Alex, Carsten, Divya, Dung, Geoffroy, Peter, Rahul, Sacha, Sophia thank you for your insights, your rigor, and for not muting me during my caffeine-fueled tangents.

A nod of gratitude to the committee who agreed to review my thesis. I promise I owe you all a coffee.

To Geoffroy, thank you for being both a guide and a friend during my time in Paris. You made sure I had meaningful projects to work on and generously made time in your busy schedule to support me. On top of that, you ensured I wasn't alone in the city, a gesture of kindness I'll never forget. Now, I'm super excited as I look forward to my postdoc years with you. Here's to the papers we'll write, the ideas we'll explore, and, of course, the quest for the best croissant in the land. Merci pour tout.

To Stathis Zachos. You've supported me through every jump, every blunder, and every wild hop I've taken in my career.

To my friends back home. There's something extraordinary about our friendship. One that has weathered, oh my God, so many years. We've gone from playgrounds and classrooms, to scattered lives across the globe, yet no matter where we are, you've remained my constant. In a world that changes so fast, you're the steady ground I can always lean on when life feels weird. Even as we've taken our own paths and built new lives, that connection we forged so long ago hasn't just survived, it's grown stronger. You're the ones who remind me who I am when things get overwhelming, the ones who bring me back to myself when I lose my footing. Whether it's through a message, a memory, or just the knowledge that you're out there, you've been my quiet strength. Thank you for being the people I can count on, no matter how far apart we are.

To my in-laws. Linda and Steve, you welcomed me into your home and into your hearts, providing warmth and support through all the ups and downs of this journey. Linda thank you for your kindness. Steve, if this work had a dedication page for culinary excellence, your grill would be front and center. Rick and Cheryl, thank you for your advice, humor, and support. You've all been there in your own ways, rooting for me, lifting me up, and reminding me that family doesn't just support, it nourishes. Thank you for being my cheerleaders. I couldn't have asked for a better extended family to lean on.

To my parents, everything I am, and everything I hope to become, starts with you. Mom, your caring for people is so extraordinary that it is sometimes baffling to me. Your unending energy and determination could power a small city. Dad, your boundless patience and quiet kindness taught me that strength doesn't need to shout; it listens, it endures, and it always knows how to fix things around the house. Whether it was cheering at my basketball games, organizing chaos into something resembling order, or somehow finding the time to be everywhere I needed you, you've always been a force of nature. From the countless hours you spent driving me to practice as a kid, to that bittersweet drive to the airport when I set off to build a life abroad, you've been with me every step of the way, often literally, with snacks packed for the journey. And through it all, no matter how far I've gone, you've never made me feel like I was too far from home. You've shaped me not just with your love but with the example you set every single day. I can only hope to pass on a fraction of your humor, generosity, and stubborn determination to the world. Thank you for being my foundation, my fans, and occasionally, my reality checks. I owe you everything.

To my brothers, you've been my greatest teachers, life coaches, and sometimes unwitting stand-up comedians. Through your wisdom -or whatever it is you call it- you've shown me how to navigate life with resilience, humor, and the occasional well-timed eye-roll. You've taught me lessons I couldn't have learned anywhere else: how to stand my ground, how to laugh at life's chaos, how to survive a wrestling match. You've been there for every stumble and every triumph, offering advice that ranged from "You've got this" to "That's what you get". You've been my reality slap when I was too serious, my hype squad when I doubted myself, and my lifelong allies in turning even the most mundane moments into hilarious memories. Whether it was shared jokes that only we'd understand, or simply

knowing you've got my back, you've shaped me in ways I'm endlessly grateful for. Thank you for keeping me grounded, making me laugh, and reminding me that no matter where life takes us, there's no bond quite like the one forged by growing up together. You'll always be my favorite teachers, just don't let it go to your heads.

Finally, to Kelsey, my anchor, my adventurer, and my greatest love. We've crisscrossed continents and crammed our lives into boxes more times than I can count, yet wherever we've landed, you've made it feel like home. Through all the chaos of moving, adapting, and figuring out life, you've been my constant; the person who insists I eat something other than cereal for dinner, who gently reminds me to sleep when I'm deep into a late-night spiral, and who knows exactly when to pull me out of my own head. You've stood beside me through every high and low, with a strength and grace that humbles me daily. And when I stare blankly out the window, you have this uncanny way of pulling me back, making me talk about the hard things, the real things, the things I'd otherwise try to ignore. But it's not just the serious moments that have defined us, it's everything in between that makes us, us. It's the quiet joy of Sunday mornings when the world feels still, when you bring pizza or a Kanelsnegle for breakfast or when you make coffee and the whole house smells like hazelnut. It's how you've turned even the messiest days into memories I'll treasure forever. When everything felt overwhelming, you were the calm at the center of the storm, reminding me with your steady presence that we'd figure it out together. You've turned the chaos of this journey into something beautiful. You've been my greatest companion, my brightest light, and the person who makes every challenge feel surmountable simply because you're by my side. This PhD, every (not so) late-night struggle, every fleeting moment of doubt, carries your fingerprints as much as mine. You didn't just support me through this; you carried me when I couldn't carry myself. I owe you more than words can ever express, and I will spend my life trying to show you how much you mean to me.

This PhD, every page of it, carries your invisible signature. For every reassuring word when I doubted myself, and for every time you reminded me that we were in this together, I can never thank you enough. I'm proud of this thesis, but I'm prouder still to call you my wife. You never once doubted me even when I doubted myself. Your sacrifices, your strength, and your humor have been my compass through every storm. Thank you for being my biggest fan, my grounding force, and for keeping me sane when I was ready to throw my laptop out the window. This PhD is as much yours as it is mine.

To everyone mentioned here and to those I might have inadvertently missed, you have my endless gratitude. This thesis is dedicated to you all; my village, my team, my people.

Nikolas Melissaris,
Aarhus, November 24, 2024

Contents

Part I.

Overview

1. Introduction

1.1. Multiparty Computation (MPC)

A Toy Example Imagine a group of friends. Let's assign them totally random names: Alice, Bob, and Carol. They will be recurring characters all throughout this thesis. This group of friends, for their own reasons, are in desperate need of computing the average of their salaries. Additionally, for reasons unlear to the outside observer, they are very reluctant to share their individual salary with each other. This might seem like an impossible task at first but if you think a little bit, a very simple idea arises. Alice, who makes a (in some currency) can pick a random number, say r and add it to her salary. Now she can pass this number $r + a$ to Bob who will add his salary b to the sum and pass it to Carol. Carol will add her own salary c and give the final sum $r + a + b + c$ to Alice. Now Alice can subtract r and retrieve the sum of all salaries, divide by 3, and announce the average to the group. You can notice that during this process, no information about the private inputs was leaked. When Bob receives $r + a$ he has no way of knowing a unless he correctly guesses r. Same goes for Carol. The value she sees is *masked* by a random value and thus seems random.

Why MPC Matters The motivation for multiparty computation (MPC) lies in its ability to achieve this seemingly difficult task of collaboration with privacy. In real-world applications, privacy concerns often prevent entities from pooling their data for mutual benefit. For example, hospitals may want to analyze shared medical data to detect trends in therapies and potential cures, but strict privacy regulations make direct data sharing impossible. Similarly, financial institutions may want to collaborate on fraud detection without exposing sensitive customer data. MPC provides a solution to these challenges by allowing such computation without requiring any party to reveal their private data.

MPC extends the basic idea which we talked about in the toy example above, to much more complex computations, from statistical analyses and machine learning, to the canonical example of a real world MPC application: the sugar beet auction by the Danish farmers [Bog+08]. All that while maintaining rigorous privacy guarantees. For computer scientists MPC represents an exciting intersection of algorithms, distributed computing, and cryptography. It opens the door to collaborative computations that were previously impossible due to privacy concerns and/or regulations, facilitating applications in finance, healthcare, and beyond.

Lastly, the MPC paradigm offers a way to think about secure computation in distributed systems. Understanding the underpinnings of MPC not only enhances a computer scientist's toolkit but also provides insights into how to design systems that respect both functionality and privacy.

In short, multiparty computation (MPC) enables a group of parties to jointly compute a function over their inputs while keeping those inputs private. To ease into formalism, let $P_1, P_2, \ldots, P_n$ denote n participants, each holding a private input x_i (for $i = 1, \ldots, n$). The goal is for the parties to jointly compute a function $f(x_1, x_2, \ldots, x_n)$ such that the result is output, and no additional information about each party's input is revealed. The protocol should ensure that no group of parties can learn anything more than $f(x_1, x_2, \ldots, x_n)$.

Security Models

In the MPC setting, security is defined with respect to some adversary $\mathcal{A}$ who controls some of the participating parties.

Semi-Honest v. Malicious Adversary Depending on how $\mathcal{A}$ behaves during the execution of a protocol we get the following two categories:

1. Semi-Honest Adversary: Parties follow the protocol honestly but may attempt to learn additional information from any intermediate values they observe.
2. Malicious Adversary: Parties can deviate arbitrarily from the protocol.

Security in these models can be formalized via simulation-based definitions. Let Π be an MPC protocol for computing f in the presence of adversaries. Security states that for any adversary corrupting a subset $C \subset \{P_1, \ldots, P_n\}$, there exists a simulator $\mathcal{S}$ such that the view of the adversary in the real execution of Π is indistinguishable from its view in an idealized execution where a trusted third party computes f and sends results to participants. This can be expressed as:

$$\mathsf{View}_{\mathcal{A}}^{\text{real}}(x_1, \ldots, x_n) \approx \mathsf{View}_{\mathcal{A}}^{\text{ideal}}(f(x_1, \ldots, x_n)).$$

Static v. Adaptive Adversary Depending on when $\mathcal{A}$ chooses to corrupt parties we get the following two categories:

1. Static Adversary. A static adversary selects the set of corrupted players at the very beginning, before the protocol starts, and cannot change this set afterward.
2. Adaptive Adversaries. An adaptive adversary dynamically selects which participants to corrupt during the execution of the protocol.

Security Guarantees

In order to define security in MPC we imagine an "ideal-world" in which our protocol behaves exactly how we want it. It's a secure "box" (which we call an *ideal functionality*) that receives inputs from the parties and gives the correct output to each party and nothing more. The way we define security is by assuring the parties that their interaction real-world protocol is pretty much the same as if they had interacted with an ideal functionality.

However, depending on adversarial power, the notion of an ideal functionality *revealing nothing more than the output* can be satisfied with any of the following guarantees:

1. Guaranteed Output Delivery. It ensures that, regardless of how the adversary deviates from the protocol, all honest parties are guaranteed to receive the output. This is the best guarantee that a protocol can offer.

2. Fairness. Fairness ensures that if the adversary can compute the output, then all of the honest parties will also receive it. In simpler terms, fairness means that either everyone gets the outputs or no one does.

3. Security with abort. The adversary has the ability to halt the protocol at any stage, including before or during the output phase. This allows the adversary to obtain the output and force the honest parties to abort without a result.

Each of these guarantees provides a different tradeoff between practicality and resilience against adversarial behavior.

Corruption Threshold

In multiparty computation (MPC), feasibility results vary significantly depending on the fraction of parties that can be controlled by the adversary $\mathcal{A}$. In protocols with an honest majority, security guarantees can be achieved with information-theoretic techniques. For example, in the semi-honest model a protocol can securely compute any function as long as fewer than half of the participants are corrupted ($t < n/2$, where n is the total number of parties). The same result in the malicious model, requires that fewer than one-third of the participants are corrupted ($t < n/3$). Under these thresholds, MPC protocols can achieve unconditional security, meaning that security does not rely on any computational assumptions.

With a dishonest majority ($t \geq n/2$), MPC protocols rely on computational assumptions to achieve security. Dishonest majority protocols often assume the existence of a public-key infrastructure (PKI) and a broadcast channel in order for the protocol to succeed. Protocols designed for a dishonest majority tend to have higher computational and communication complexity due to the need for "heavier" cryptography, such as zero-knowledge proofs, which can help to ensure that exchanges during the protocol execution happen in the way they are supposed to. The computational security that MPC protocols provide in this case relies on the hardness of certain problems, such as the discrete logarithm or factoring problems.

Key Techniques in MPC

We will now mention some of the building blocks of MPC protocols. Although it's impossible to make an exhaustive list of all of them, we will mention some of the very widely used ones and some specialized ones that appear in this thesis. Some others that are more complex and appear only in specific upcoming chapters will be introduced at

the appropriate section to avoid clutter and to improve readability in those particular sections.

Secret Sharing One of the foundational techniques in MPC is secret sharing and most well-known secret sharing schemes are *Shamir's secret sharing* and *additive secret sharing*. A secret $s \in \mathbb{F}_q$ is divided into n shares $s_1, s_2, \ldots, s_n$ such that any t-subset of shares can reconstruct s, but any subset of fewer than t shares reveals no information about s.

In additive secret sharing, a secret s is split into n random shares $s_1, s_2, \ldots, s_n$ such that:

$$s = s_1 + s_2 + \cdots + s_n \pmod{p},$$

where p is a large prime number. Each party receives one share, and any subset of $n-1$ or fewer participants learns nothing about s. To reconstruct s, all n shares are summed together.

Shamir's scheme is based on polynomial interpolation. This is done by choosing a random polynomial $p(x)$ of degree $t-1$ such that $p(0) = s$ and setting $s_i = p(i)$ for $i = 1, \ldots, n$. The sharing algorithm can be formalized as:

$$s_i = p(i) = s + a_1 i + a_2 i^2 + \cdots + a_{t-1} i^{t-1},$$

where $a_1, \ldots, a_{t-1}$ are randomly chosen from $\mathbb{F}_q$.

Homomorphic Encryption A technique frequently employed in MPC is homomorphic encryption. A homomorphic encryption scheme allows computation on ciphertexts such that the decrypted result is equivalent to the computation on the plaintexts. For example, given an encryption function E and messages m_1, m_2, additively homomorphic encryption ensures that:

$$E(m_1) \cdot E(m_2) = E(m_1 + m_2).$$

Such schemes are particularly useful for MPC as they allow parties to perform operations on encrypted data without revealing the underlying values.

Oblivious Transfer Oblivious transfer (OT) is another essential primitive in MPC. It is a very powerful tool and it is sufficient to realize any secure computation functionality [Kil88; IPS08]. In a 1-out-of-2 OT protocol, a sender has two messages, m_0 and m_1, and a receiver has a choice bit $b \in \{0, 1\}$. The protocol allows the receiver to obtain m_b without learning m_{1-b}, and the sender learns nothing about b. Take, for example, the famous GMW protocol for boolean circuits [GMW87a]. We'll talk about the two party case for simplicity. Two parties P_1 and P_2 have inputs x and y. They secret share said inputs so that $x = x_1 \oplus x_2$ and $y = y_1 \oplus y_2$, where x_i and y_i are the shares of P_i. Now the parties proceed to evaluate the circuit gate by gate. XOR gates are computed locally: if $z = x \oplus y$ parties can compute shares of z by having every party set $z_i := x_i \oplus y_i$. For AND gates, things are not as easy unfortunately. We have $z = x \wedge y$ and now the parties cannot compute shares of z with local computation alone. Instead, they have to interact.

In order to do this securely they will use a *1-out-of-4 oblivious transfer* scheme. P_1 will pick a uniformly random bit z_1 to be its share of z and now needs to determine P_2's share. Since $z_2 = z \oplus z_1$ then we have $z_2 = (x \wedge y) \oplus z_1 = ((x_1 \oplus x_2) \wedge (y_1 \oplus y_2)) \oplus z_1$ and for all possible values of x_2y_2 P_1 can compute $z_2^{x_2y_2} = ((x_1 \oplus x_2) \wedge (y_1 \oplus y_2)) \oplus z_1$. P_1 will play the role of the sender with input $z_2^{01}, z_2^{01}, z^{10}, z^{11}$ and P_2 will be the receiver with inputs x_2y_2. The OT will give the correct $z_2^{x_2y_2}$ to P_2 and now both parties are holding correct shares.

It is important to note that we can build passively secure OT given any public key encryption scheme. But can we do better? Enter *OT extension*. OT extension allows parties to use a small number of base OTs to get a larger number of OTs from more efficient primitives. While the first construction in [Bea96] was inefficient, [IKNP03] presented a very efficient extension with small overhead. Recently, with a technical innovation called a *pseudorandom correlation generator (PCG)* (which we define in a little bit) Boyle et al. [BCGIKS19] present the first concretely efficient construction for extending OTs.

Pseudorandom Correlation Generators Recent developments in MPC have focused on reducing the communication complexity of protocols. One approach involves using pseudorandom correlation generators (PCGs) [BCGI18; BCGIKS19], which enable parties to pre-generate correlated randomness that can be used to mask intermediate computations. Given a correlated seed σ, each party can locally expand this seed to get l correlated values $(r_1, r_2, \ldots, r_n)$, reducing the need for extensive interaction during the protocol execution. Formally, let G be a pseudorandom generator that expands σ such that:

$$(r_1, r_2, \ldots, r_l) = G(\sigma),$$

where the r_i satisfy a specific correlation required by the MPC protocol (e.g., additive shares summing to zero).

The preprocessing paradigm A lot of MPC protocols rely on correlated randomness. But correlated randomness doesn't rely on the private inputs of the parties so what we can do is split the protocol in two phases: A *preprocessing* (or *offline*) phase is used to generate the necessary correlated randomness and the *online* phase which is when the output is computed. Typically, the processes run in preprocessing are computationally expensive but the benefit is that then we can have a very efficient online phase. Furthermore, PCGs can compress the preprocessing phase even futher by allowing two parties to take short, correlated seeds, and expand them to a large amout of correlated randomness. Using PCGs we can achieve sublinear communication complexity but also get better space efficiency since we only need to store the initial short seed.

Advances in MPC have made significant strides in improving both efficiency and scalability, transforming MPC from a theoretical construct to a practical solution for secure collaborative computation. Innovations such as PCGs and low-communication secret-sharing schemes have reduced the communication complexity of traditional MPC protocols, addressing bandwidth limitations that have hindered large-scale deployments. Combining

MPC with techniques like homomorphic encryption and zero-knowledge proofs further enhances performance, particularly for computationally intensive applications. In terms of applications, MPC is now successfully implemented in fields like privacy-preserving machine learning and secure data analytics, enabling collaborative computations on sensitive data without sacrificing privacy.

1.2. Making MPC more robust - What is this thesis about?

In this thesis we set out to make more *robust* MPC. But we haven't defined what this means. An MPC protocol can be evaluated across multiple axes: What is its round complexity? What are the security guarantees that it offers? What corruption threshold does it support? In addition we care about what assumptions are being used or how much computation is needed. In the preprocessing paradigm we care about how much computation was pushed into the offline phase for an efficient online phase. There are more parameters that we study in order to talk about the efficiency of a protocol but these are the most common ones.

So what do we mean when we talk about making MPC more *robust*? In the bibliography we can find the term *robust* to mean *guaranteed output delivery* [PR19], providing security against t *malicious adversaries* (t-robustness) [HN06], or producing a correct output in a *constant number of rounds* [UR24]. There seems to be no consensus. Because *robust* seems a bit overloaded we'll avoid specifying it precisely. In this thesis we use the term more freely, as a colloquiallism rather than a formal term. By making MPC more *robust* we mean to improve protocols in any of the aforementioned axis:

1. In terms of security guarantees, we know from results due to Cleve [Cle86] that with a dishonest majority a protocol can't achieve guaranteed output delivery (or even fairness). In our paper [BMRS24] we build on a line of work that asks "What can we do better in the presence of a dishonest majority?". A satisfactory solution comes from *identifiable abort*. At the end of a protocol that the adversary has aborted, the honest parties will either learn the output, or learn the identity of one or more cheaters. The honest parties can then use this information to re-run the protocol excluding corrupt parties. For an overview see Section 2.1 and for the full paper see Chapter 6.

2. In terms of security modeling we continue a line of work around a new security model "Security with *Friends and Foes*" [AOP20]. The authors ask "Can we extend the standard notion of security against malicious adversaries to prevent leakage of private information to (possibly colluding) subsets of (semi)-honest parties?". In other words (making obvious what gave rise to the name), can we protect our input from our *foes* but also from our *friends*? In our work [MRY23] we study this new notion and provide protocols that match the optimal parameters (round complexity and corruption threshold) that were introduced in [AOP20]. For an overview see Section 2.2 and for the full paper see Chapter 7.

3. In terms of efficient communication, in our work [CKMSS24] we introduce a PCG for pseudorandom permutations. This means that the expansion of a short seed will give each party a distinct random position in a permutation. Though limited to the three party setting, ours is the first PCG for non-additive correlations (meaning that the parties obtain something other than an additive share of the target correlation) which doesn't assume indistinguishability obfuscation. For an overview see Chapter 3 and for the full paper see Chapter 8.

4. In terms of using weaker primitives in MPC protocols, in our work [BCM24] we introduce and construct *structured-seed local PRGs* (SSLPRGs). Pseudorandom generators (PRGs) are functions whose output can't be distinguished from a random string by a polynomial-time algorithm. Local PRGs have each output bit depend only on a constant number of input bits. Finally, *structured-seed* means that in contrast to regular PRGs where the seed is sampled uniformly at random, the seed is sampled from a different distribution. Sampling from the uniform distribution is expensive so removing this necessity makes our object simpler. We show how our SSLPRG can be used to prove results in a variety of areas including constant overhead secure computation [IKOS08] and sublinear communication [BCM23]. For an overview see Chapter 4 and for the full paper see Chapter 9.

2. Better MPC - Security

We already talked about the different ways the adversary can behave in an MPC protocol. The dishonest majority setting is a challenging one for MPC as the strong guarantees of guaranteed output delivery and fairness cannot be achieved. To this end, in the following sections we talk about two notions that will help us towards our goal of making MPC more robust. In Section 2.1 we talk about *identifiable abort* and how it strengthens security-with-abort protocols. In Section 2.2 we talk about *security with friends and foes* and how it strengthens regular malicious security.

2.1. MPC with Identifiable Abort

One of the major parameters that we want to study in an MPC protocol is the corruption threshold, meaning the maximum number of parties that can act maliciously or share information without compromising the protocol's guarantees. Ideally we would like to preserve said guarantees even when all but one party in the protocol are corrupt.

We have seen the notion of *fairness* and how it ensures that if any dishonest participants see the output, so do the honest ones. It's known however, that when we are in the dishonest majority setting, fairness and the even stronger guarantee of guaranteed output delivery are impossible [Cle86]. Consequently, protocols instead aim for a weaker standard which is security with abort. This means that a corrupt participant can force the protocol to terminate prematurely, preventing some or all honest participants from obtaining the correct result. We can see how this can be a trivial form of denial of service where the adversary can abort over and over again, not allowing the honest parties to get anything meaningful from the protocol.

2.1.1. The Notion of Identifiable Abort

In dishonest-majority scenarios where fairness is impossible we would like to at least be able to allow honest parties to agree on the protocol's termination and, even better, to identify at least one of the cheating parties if an abort occurs. This property, called *identifiable abort*, can discourage malicious behavior since any cheater exposed can be excluded from further rounds of computation. Identifiable abort and unanimous abort were first established as achievable by Goldreich, Micali, and Wigderson [GMW87a]. Their well-known GMW compiler provides a method to convert any protocol secure against semi-honest adversaries into one that achieves identifiable abort security against malicious adversaries. The core idea of the GMW approach is straightforward: parties begin by committing to their inputs and then collaboratively execute an augmented coin-tossing protocol to generate a committed random tape for each participant. Following

this setup, the original protocol is executed over a broadcast channel, with an additional requirement—each message is accompanied by a zero-knowledge proof to demonstrate that it was computed correctly.

While conceptually elegant, this method has significant efficiency drawbacks. Its reliance on cryptographic primitives in a non-black-box manner introduces substantial computational overhead, making it impractical for many real-world applications.

Identifiable abort in dishonest-majority MPC was first systematically explored by Ishai, Ostrovsky, and Seyalioglu [IOS12]. They found that it's impossible to construct unconditionally secure ID-MPC using only broadcast channels and pairwise ideal functionalities (such as oblivious transfer (OT)), which contrasts the secure-with-abort model, where pairwise OT alone is sufficient. Later, Ishai, Ostrovsky, and Zikas [IOZ14] introduced a compiler that converts any semi-honest protocol using correlated randomness into one that can withstand malicious behavior and provides identifiable abort in the correlated randomness model. The high level idea is pretty simple and it follows what we saw before with the GMW compiler: each party commits to their inputs and randomness for the semi-honest protocol, broadcasts their messages each round, and uses zero-knowledge proofs to show that each message is the one that it's supposed to be. Additionally, their work [IOZ14] presented a compiler that could transform any cryptographic preprocessing phase that is secure with abort into one supporting identifiable abort. The reader might be familiar with the impossibility result in [IOS12] and find this result impossible, but this approach manages to sidestep this barrier by depending on black-box OT protocol usage instead of an ideal OT functionality. So this is the first identifiable abort construction that only requires black-box access to cryptographic primitives, particularly an adaptively secure OT protocol and a broadcast channel. However, the construction relies on adaptively secure OT in the preprocessing phase and on computationally expensive zero knowledge used to prove correct protocol execution.

Building on [IOZ14], there was a surge in trying to improve ID-MPC and bring it closer to "practical" efficiency. Baum et al. [BOS16] introduced an identifiable abort protocol for arithmetic circuits in the preprocessing model where the online phase is a variant of BDOZ [BDOZ11] that can identify cheaters. Although their solution avoids adaptively secure OTs, the preprocessing phase overhead is about n (which means that their protocol takes about n times more than the the computation of non-identifiable protocols) and uses cheater identification for lattice-based cryptography, which is not efficient in practice. Spini and Feher [SF16], modify the SPDZ protocol to allow for cheater identification by ensuring that only correct shares are opened. Their identifiable preprocessing approach, however, also relies on the same costly methods as [BOS16], including verifiable decryption. Cunningham et al. [CFY17] use Pedersen commitments to detect cheaters in the online phase, which restricts the computation to specific finite fields and makes preprocessing resource-intensive, as these commitments must be generated during preprocessing. Finally, Baum et al. [BOSS20] present an ID-MPC protocol for boolean circuits that operates in a constant number of rounds and employs cryptographic primitives in a black-box manner. Their method avoids public-key operations post-setup and complex zero-knowledge tools but is restricted to boolean circuits and incurs significant overhead from reconstructing a large garbled circuit via multiparty BMR [BMR90].

2.1.2. Our Contributions

Our paper [BMRS24] introduces a simpler approach to identifiable abort. We construct an efficient MPC protocol that supports identifiable abort for arithmetic circuits over large fields. The standout feature of the protocol lies in its simple yet effective online phase, which leverages pairwise information-theoretic MACs. This design follows the approach used in the BDOZ protocol [BDOZ11] for secure-with-abort protocols.

The simplicity of the online phase has practical implications: the preprocessing step only needs to generate standard authenticated multiplication triples. This, in turn, minimizes the overhead typically associated with adding identifiable abort functionality.

The key idea enabling identifiable abort is a novel compiler that transforms specific sender-receiver protocols -where one party has private inputs- into protocols that support cheater identification. Unlike the compiler from [IOZ14], which is limited to preprocessing protocols and requires them to be adaptively secure, our new compiler lifts these restrictions, making it more versatile and applicable in a broader range of settings. In summary, our protocol combines the simplicity and efficiency of BDOZ with the added features for cheater identification, supported by lightweight preprocessing making it a step forward in practical MPC.

Related Work

Having already mentioned works that seem foundational to the notion of identifiable abort, in this section we will mention constructions that offer identifiable abort but usually in more restricting settings. Brandt et al. [BMMM20] and Simkin et al. [SSY22] independently explored methods to achieve dishonest-majority MPC with identifiable abort, relying on correlations shared by fewer than all n parties. Cohen et al. [CGZ20] investigated two-round MPC in a dishonest-majority setting with broadcast, analyzing when identifiable abort can be achieved under different broadcast scenarios. Damgård et al. [DMRSY21] extended this analysis to honest-majority settings. Later, Damgård et al. [DRSY23] examined necessary setups for two-round identifiable abort in the plain model, while Ciampi et al. [CRSW22] presented a four-round solution for ID-MPC under the same conditions.

If we relax malicious security to covert security, Faust et al. [FHKS21] and Scholl et al. [SSS22] proposed compilers that lift passively secure MPC to covertly secure MPC with identifiable abort (with dishonest majority) using time-lock puzzles and Attema et al. [ADEL22] removed the need for time-lock puzzles in the honest majority setting. These methods actually support a stronger model, called publicly verifiable MPC, which implies identifiable abort.

In the honest majority setting, using framing-free designated-verifier zero-knowledge proofs, Hazay et al. [HVW22] presented an alternative to the IOZ14 compiler.

Chen et al. [Che+21] developed an efficient identifiable abort protocol for RSA key generation with dishonest-majority security. Their efficiency savings come mainly from the model they use to circumvent the pairwise interaction barrier, by using a party called a coordinator, who is able to aggregate-and-broadcast.

Recently, Cohen et al. [CDKs24] proposed another identifiable abort approach that, like our work, avoids the adaptive OT requirement in [IOZ14]. Their method focuses on revealing committed inputs if cheating occurs, rather than using random tape openings to verify protocol messages as in our approach. Cohen et al. [CDKs24] rely on a special committed OT functionality instead of direct use of protocol messages.

Challenge of Adaptive Security and Identifiable Abort

The compiler from [IOZ14] offers a simple way to achieve identifiable abort in a preprocessing protocol by having each participant commit to a random tape and then running a secure-with-abort protocol. If the protocol aborts, participants open their committed random tapes, enabling others to identify the cheater by re-running a local copy of the protocol. Since preprocessing is input-independent, this approach generally does not compromise privacy.

A challenge here is simulating the view for corrupted parties. If the protocol aborts, the simulator must open honest parties' tapes that are consistent with what the adversary has already seen. This is trivially done using adaptive primitives and this is where this seemingly natural requirement for adaptive security comes from. Some attempts [BDD20; BOSS20] have avoided adaptive security but didn't manage to give a construction for arbitrary fields.

2.1.3. Some Preliminaries

In this section we will briefly talk about about some of the terms that are mentioned to improve readability. Full preliminaries can be found in Section 6.2.

A Message Authentication Code (MAC) ensures both the integrity and authenticity of a message exchanged between two parties. Given a message m and a secret key k a MAC scheme generates a short, fixed-length tag t that is unique to the message-key pair. The recipient can verify the tag to confirm that the message was neither altered nor forged.

A Vector Oblivious Linear Evaluation (VOLE) correlation consists of two vectors $\mathbf{u}, \mathbf{w}$ held by P_A, and a value Δ and a vector $\mathbf{v}$ held by P_B, such that $\mathbf{w} = \mathbf{u} \cdot \Delta + \mathbf{v}$. The value Δ and $\mathbf{v}$ are chosen uniformly at random.

One can view a VOLE correlation as an information-theoretic MAC on the vector $\mathbf{u}$ of P_A. Assume that P_A could produce a pair of vectors $\mathbf{u}', \mathbf{w}'$ with $\mathbf{u}' \neq \mathbf{u}$ such that the VOLE correlation holds, along with the original $\mathbf{u}, \mathbf{w}$ which means that we would have:

$$\mathbf{w} = \mathbf{u} \cdot \Delta + \mathbf{v}, \qquad \mathbf{w}' = \mathbf{u}' \cdot \Delta + \mathbf{v}$$

In order for this to hold we need $(\mathbf{w}[i] - \mathbf{w}'[i])/(\mathbf{u}[i] - \mathbf{u}'[i]) = \Delta$ (where i is the index in which $\mathbf{u}$ and $\mathbf{u}'$ differ). This can be achieved only if P_A knows the secret Δ so it can forge a MAC on a different value $\mathbf{u}'$. However, Δ is chosen randomly from a set of large size so the probability of a correct guess is negligible if we pick the set properly.

The important thing to know about the BDOZ protocol from [BDOZ11] is that it uses pairwise information-theoretic MACs, overcoming previous works that only offered computational security in the online phase. Every party P_i has a set of MACs for its

share x_i of a shared value x under the keys for every other party as well its own keys to verify the MACs of the other parties.

2.1.4. Informal Technical Overview

Online Phase To get to identifiable abort, we utilize *linear secret sharing* where participants are committed to their shares using linearly homomorphic commitments. In [BOS16] we see how having commitments that support multiple receivers and identifiable abort can be used to open secret shared values by verifying each share's commitment. Also, using a preprocessing phase to generate authenticated multiplication triples where each share is committed using homomorphic commitments, we can get an MPC protocol that uses the linearity of the commitments and Beaver multiplication to compute the desired function. The way that they instantiated their information-theoretic commitments was pretty complex, resulting in a more expensive preprocessing phase than the BDOZ scheme [BDOZ11; DPSZ12].

Our online phase follows this approach, using preprocessed triples and identifiable commitments. However, our protocol manages to have drastically better efficiency because of how we generate identifiable abort-compatible multiplication triples during preprocessing and how we construct identifiable, linearly homomorphic commitments.

Preprocessing Phase In the preprocessing phase, our goal is to create additive secret shares of random multiplication triples over a large field, committed through linearly homomorphic commitments. To this end, parties first run a secure-with-abort protocol, Π_{Trip}, to generate unauthenticated triples and then commit to each share with homomorphic commitments. To verify that the shares are correct we apply a sacrifice-based check, where one triple is "sacrificed" to validate another.

Now, in order to get identifiable abort in the preprocessing phase we proceed as follows. First, just as in the IOZ14 compiler [IOZ14], each party commits to their random tape for the secure-with-abort protocol Π_{Trip} before running it. If Π_{Trip} aborts, parties open their tapes and are able to run the protocol to find out who cheated. This process requires that the protocol has verifiable transcripts, allowing consistent identification of cheaters based on everyone's view of the randomness. This property is known as *identifiable cheating*. We show that this property can be added easily on any secure-with-abort protocol by adding digital signatures to all pairwise messages, ensuring that a record exists to verify any protocol deviations.

We also tackle the challenge of simulating Π_{Trip} while bypassing the need for adaptive security. In [BOSS20], we see how the simulator runs a local honest execution of Π_{Trip} in order to produce the messages of the honest parties. Now, although it can't extract the adversary's inputs (from the protocol execution), if there is an abort the simulator can extract the shares of the multiplication triples from the commitment functionality. This simple idea is the driving force behind why the simulator doesn't need to adaptively corrupt honest participants since the honest execution is sufficient.

Finally, if Π_{Trip} itself is successful but the triple sacrifice check fails (because a corrupt party committed to the wrong share), parties can open their random tapes from Π_{Trip} to

recover all shares and verify them by comparing the committed shares in the homomorphic commitment scheme.

Building Identifiable, Homomorphic Commitments We use pairwise information-theoretic MACs to implement homomorphic commitments. Each party's commitment is authenticated by a MAC shared with every other party like in the BDOZ protocol. These MACs can be efficiently generated with vector oblivious linear evaluation (VOLE) protocols that rely on learning parity with noise assumptions [BCGI18; WYKW21; BCGIKRS19]. However, going back to the impossibility result due to Ishai, Ostrovsky, and Seyalioglu [IOS12], identifiable abort is impossible using pairwise information theoretic MACs in a black box way.

Ideally we want to use the same commit-and-open technique that we used for the triple generation phase. If there is an opening that fails then the parties will open the randomness that was used to generate the MACs in the VOLE protocol and from those, they can re-run the execution and find who cheated.

Unfortunately it's not going to be that simple. The commit-and-open technique only works in preprocessing protocols; opening all parties' random tapes in the online phase would leak private inputs. Additionally, we still have to deal with the issue of adaptivity that was mentioned before.

Compiling Sender-Receiver Protocols to Identifiable Abort To manage to get around the issues with adaptive security in the online phase, we restrict our compiler to a class of *sender-receiver protocols* where only one participant (the sender) holds private input. We will see how this will not pose a problem in constructing MPC. Our compiler makes use of the underlying secure-with-abort protocol but doesn't face the limitation of IOZ compiler which was only for preprocessing protocols. This makes it suitable for use with any linearly homomorphic commitment scheme, which we instantiate using VOLE to set up the MACs.

How do we make use of just one party having input? In this case we are able to follow the same strategy as in the preprocessing phase but now if we only require the receivers to open their random tapes, then no private input is revealed. By making the assumption that receivers don't communicate with each other, receivers can still detect a dishonest sender. By this we don't mean that the expectation is that corrupt receivers don't communicate to gain security, rather that it is a protocol specification.

There is a problem though. Even if the receivers don't have private inputs, they might have private outputs which can't be revealed. Because of this we need to separate how we handle aborts in the case where a sender aborts versus when a receiver aborts. In the case where a sender aborts, receivers send evidence to the sender in private, who selects and publishes a proof. If a receiver is responsible for an abort, they open their view which shows that the sender cheated (because of the restriction we imposed on the communication between receivers). Since the private output of the receiver depends on the corrupt sender's input, it's ok to reveal it.

Avoiding Adaptive Security with Online Extractability When talking about using an honest execution of the protocol to extract adversarial inputs, that will in turn help in the simulation of all the identification stages, we hinted towards the idea that the full power of adaptivity is not needed in order to achieve this. Based on this observation, we define *online extractability*, a weaker but sufficient property that allows inputs from corrupted parties to be extracted without adaptive simulation by making imperceptible adjustments to the common reference string (CRS) or other hybrid functionalities. This actually is how many UC simulators work in practice. Online extractability thus allows honest execution simulation without needing adaptive corruption.

2.1.5. Efficiency Analysis

Efficiency Compared with MPC with Abort Identifiable abort is an added property so in order to evaluate its efficiency we study how much overhead is added to a secure-with-abort protocl when we transform it. We compare our protocol to the two versions of the Le Mans protocol [RS22]. In the first version, called Le Mans 1 in Table 2.1 (from [BMRS24]), the protocol generates what are called "partial triples" and only authenticates these triples during the online phase. This version has an asymptotic preprocessing cost of $O(n^2 \log |C|)$ communication (where $|C|$ represents the circuit size) if pseudorandom correlation generators are used for OLE and VOLE correlations. Depsite the use of PCGs, the the local computation is still $O(n^2|C|)$. In comparison, if non-silent OLE or VOLE protocols (like those using homomorphic encryption or OT) are used, communication costs would also be $O(n^2|C|)$. The online phase then costs each party $12n$ field elements.

Le Mans' second version authenticates and verifies partial triples during preprocessing, which increases the preprocessing cost to $O(n^2|C|)$ field elements, but it reduces the online phase cost to $4n$ elements per party.

Our protocol's preprocessing phase has the same base cost as Le Mans 1, plus an extra $2(n-1)|C|$ field elements per party sent through point-to-point channels. For the online phase, we use a standard BDOZ phase with authenticated triples and signatures, raising the cost by $O(n)$ which will ultimately set the online communication per party at $2(n-1)|C|$ elements in an honest execution.

A corrupted adversary can always force complaint procedures, which will in turn force messages to go through secure broadcast channels rather than point-to-point links, effectively doubling the round complexity. Additionally, although an adversary can abort the protocol requiring parties to reveal their views, the cost of resolving an abort in our protocol remains cheap. It requires only local computations to compute the messages that should have been sent, avoiding zero-knowledge proofs which incur significant computational costs.

Efficiency Compared to Other ID-MPC Protocols We also compare our protocol with previous identifiable abort constructions, the closest to our work being by Baum et al. [BOS16] and Baum et al. [BOSS20].

[BOS16] uses an $O(n^3)$ broadcast complexity per multiplication gate in preprocessing since $O(n^2)$ verifiable decryptions of RLWE ciphertext decryptions are required. By

Protocol	Building blocks	IA	Preprocessing cost	Online cost
Le Mans, v1	(V)OLE	✗	$n^2 \times$ OLE*	$12n$
Le Mans, v2	(V)OLE	✗	$n^2 \times$ OLE*+ $O(n)$	$4n$
[BOS16]	depth-1 HE	✓	$O(n^3)^\dagger$	$O(n^2)^\ddagger$
Ours	(V)OLE	✓	$n^2 \times$ OLE*+ $O(n^2)$ ‡	$O(n^2)^\ddagger$

* Random, pairwise OLE and VOLE correlations. Can be generated with amortized $o(1)$ communication using variants of LPN [BCGIKRS19; BCGIKS20b].
† Must be broadcast
‡ Corrupted party can force to be broadcast

Table 2.1.: Comparing efficient MPC protocols with and without identifiable abort. Preprocessing cost reflects the cost per multiplication in the preprocessing phase, in terms of building blocks (OLE/VOLE) plus total communication in field elements.

contrast, even if all messages in our protocol are forced to be broadcast, the broadcast cost is only $O(n^2)$ per multiplication. This improvement stems from the use of our information theoretic MACs that are cheaper to set up than the ones in [BOS16]. For the online phase, both our protocol and [BOS16] have an $O(n^2)$ complexity.

We expect our protocol to be significantly faster than [BOS16] (although there is no concrete implementation) due to the ability to use Pseudorandom Correlation Generator (PCG) techniques for VOLE and OLE correlations, which are known to be much more communication-efficient than homomorphic encryption (HE)-based methods [BCGIKS20b]. The complex preprocessing requirements in [BOS16] make it challenging to use practical PCG techniques over HE and result in an $O(n^3)$ asymptotic overhead.

The work of [BOSS20] is not directly comparable to ours (in terms of setting) since it uses garbled circuits for Boolean circuits, managing to have a constant-round online phase. In contrast, our protocol supports computations over large fields $\mathbb{F}_p$ with a round complexity depending on the circuit depth. Both protocols leverage homomorphic commitments in the offline phase: while [BOSS20] requires each party to commit to garbled circuit keys, our approach commits to the shares. The commitment in [BOSS20] uses a non-interactive vector commitment whereas we use VOLE-based commitments. Extending our commitments to their setting is an interesting idea for future work.

2.2. MPC with Friends and Foes

We talked about corruption thresholds in MPC. We saw how, generally, MPC protocols are designed with a security threshold t, such that as long as no more than t participants collude, the protocol's guarantees of privacy and correctness are maintained. If, however, more than t parties deviate from the protocol, it may compromise the privacy of the remaining $n - t$ participants.

In practice, however, we would prefer that even our honest counterparts, who aren't colluding with an adversary don't gain access to our private inputs. Alon et al. [AOP20]

introduced MPC with Friends and Foes (FaF security), a model that precisely incorporates this notion. It is important to recognize that an honest party may not remain honest indefinitely. If such a party were to become corrupted later, any sensitive information it acquired during the protocol (for instance, another party's bank account password) could still be exploited. Moreover, most people would hesitate to share something as personal as their bank account password even with their closest friends. This naturally leads to the consideration of a model where each honest party is treated as semi-honest, while a malicious adversary operates simultaneously in the system.

One way honest parties can see the private information of others is if the protocol instructs them to do so. There are MPC protocols (for example, [IKP10; IKKP15; PR18]) where if cheating is detected, any parties identified as honest (through some mechanism) they are instructed to reveal their private inputs to the other honest parties.

Alternately, honest parties might receive the view of the adversary, thus exposing them to infomartion about the other parties. Take the following example. It's very common in MPC protocols to rely on some $(t+1)$-out-of-n secret sharing scheme (meaning that it can withstand t corruptions). If the adversary who controls t shares sends these shares to an honest party, with the newly acquired information it now has the power to reconstruct the inputs of all the other parties. This seems like a more realistic model, one in which honest parties might be tempted to not stay honest if the opportunity arises.

Informally, a protocol achieves (t, h^*)-FaF security if it behaves like standard MPC with respect to any adversary $\mathcal{A}$, meaning there exists a simulator $\mathcal{S}_{\mathcal{A}}$ capable of producing a view indistinguishable from that of the t corrupt participants without needing to know the inputs of the honest parties. Additionally, for FaF security, there should also exist a separate simulator $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ for every subset of up to h^* honest participants, able to create a view indistinguishable from the honest participants without having access to the inputs of the remaining honest parties. This ensures that no messages from corrupt parties can help any group of h^* honest participants to learn more about the other honest parties' inputs.

Alon et al. defined two versions of FaF security:

1. **Weak FaF.** In this model, while $\mathcal{S}_{\mathcal{A}}$'s output must be indistinguishable from the real view of the t corrupt participants, and $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$'s output must be indistinguishable from the real view of the h^* honest participants, these two views are not required to be mutually consistent. That is, the simulated views may differ when considered jointly.

2. **Strong FaF.** Here, $\mathcal{S}_{\mathcal{A}}$ and $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$'s outputs must be jointly indistinguishable from the combined real views of the t corrupt and h^* honest parties.

These two notions of FaF security can be viewed in terms of adversarial influence: *strong FaF* accounts for cases where adversaries receive feedback on honest parties' knowledge, while *weak FaF* assumes no feedback. The main difference is that in the strong notion of FaF security we want the simulated views of $\mathcal{A}$ and $\mathcal{A}_{\mathcal{H}*}$ to be simulatable *together*.

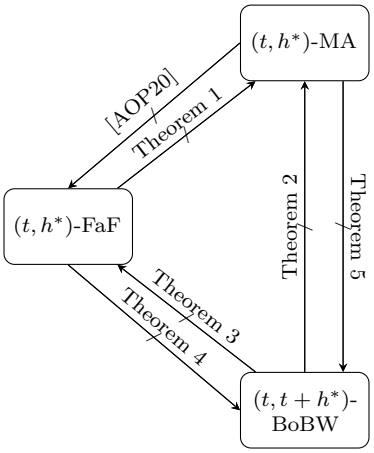


Figure 2.1.: Relationships of FaF to other notions. MA denotes security against mixed adversaries; BoBW denotes (active / passive) best of both worlds security.

2.2.1. Our Contributions

In this paper, we address two gaps identified in Alon et al.'s constructions and further examine the relationships between FaF security and other security models. Our primary focus is on FaF security with guaranteed output delivery (GOD).

First, we present a three-round construction that achieves weak FaF security with corruption threshold $2t + h^* < n$ in the CRS (common reference string) model. This is the first construction that is both round-optimal and threshold-optimal (but it's not strong FaF). Secondly, we introduce a construction achieving strong FaF security for the same threshold, $2t + h^* < n$. This is the first strong FaF construction to achieve optimal threshold, though the round complexity depends on the function's multiplicative depth and also the construction relies on correlated randomness.

Lastly, we study FaF security's connections with other security models. While Alon et al. showed that mixed adversary security (where an adversary can perform t active and h^* passive corruptions) does not imply FaF security in the computational setting, we prove the reverse: FaF security does not imply mixed adversary security. Additionally, we consider *Best of Both Worlds (BoBW)* security [IKLP06; Kat07], which allows an adversary either t active corruptions or $t + h^*$ passive corruptions, but not both. We demonstrate that FaF security neither implies nor is implied by BoBW security. These findings are summarized in Figure 2.1 from [MRY23][1].

Prior Work

Alon et al. explored the foundational aspects of FaF-secure MPC by identifying certain limitations and offering initial constructions. Their work is mostly in the guaranteed output delivery setting.

[1]The theorem numbers on the figure correspond to the theorem numbers in [MRY23].

Construction	FaF Level	Security	Threshold	Rounds	Assumptions	Preprocessing
[AOP20]						
GMW-based	Weak	Comp	$2t + h^* < n$	$\mathsf{poly}(\kappa)$	OT & OWP	no
DI-based	Strong	Comp	$5t + 3h^* < n$	3	PRG	no
BGW-based	Strong	Stat IT	$2t + 2h^* < n$	$\mathsf{poly}(\kappa)$	Broadcast	no
BGW-based	Strong	Perf IT	$3t + 2h^* < n$	$\mathsf{poly}(\kappa)$	None	no
This Work						
TFHE-FaF	Weak	Comp	$2t + h^* < n$	3	Lattices & Broadcast	no
BGW-BT-Comp	Strong	Comp	$2t + h^* < n$	$O(\kappa)$	ETP	Beaver triples

Figure 2.2.: Our constructions compared to those of [AOP20]. Notation: n denotes the total number of participants, t denotes the bound on the number of corruptions (foes), h^* denotes the bound on the number of honest parties against whom we want privacy (friends), and κ denotes the multiplicative depth of the circuit being evaluated.

Limitations The MPC parameters that were studied by Alon et al. are: round complexity and corruption thresholds. They demonstrated that two-round MPC with weak FaF security (and consequently, strong FaF security) and GOD is impossible by the simple observation regarding the impossibility of computing the AND functionality in two rounds against 2 corrupted parties in [GIKR02]. This two-round impossibility can be viewed as the reason why the leakage of honest parties' private inputs in the two-round protocol in [IKP10] in necessary.

The 2-round impossibility was shown even at minimal thresholds ($t = h^* = 1$), implying that three rounds is the best we can hope to achieve. Furthermore, they showed that certain threshold settings make even weak FaF security unattainable, regardless of the round complexity. Specifically, if n is the number of participants, t is the maximum number of corrupt parties, and h^* is the number of honest parties who should learn northing about other honest parties' inputs, then:

- Weak FaF security with GOD is infeasible if $2t + h^* \geq n$.
- Information-theoretic (statistical) weak FaF security with GOD is impossible if:
  - $2t + 2h^* \geq n$ (even with broadcast).
  - $2t + 2h^* \geq n$ or $3t \geq n$ (without broadcast).
- Information-theoretic (perfect) weak FaF security with GOD cannot be achieved when $3t + 2h^* \geq n$, even with broadcast.

Alon et al. present a three-round construction achieving strong FaF security for corruption threshold satisfying $5t+3h^* < n$. They also present a threshold-optimal construction (with $2t + h^* < n$) that only guarantees *weak* FaF security. These constructions are summarized in Figure 2.2, which is modified slightly from [MRY23].

Research on FaF security falls within the broader study of security model robustness[2]. Robust security is a valuable feature as it mitigates denial-of-service risks. In this area,

[2]This time we mean it in slightly more technical terms. We mean that the model is able to maintain its guarantees even in the face of changing adversarial conditions.

Koti et al. [KPPS21] proposed a robust privacy-preserving machine learning (PPML) framework for multiple ML tasks, Dalskov et al. [DEK21] introduced a novel four-party honest-majority MPC protocol with active security and guaranteed output delivery, and in [KPRS21], the authors presented an actively secure 4-party protocol for secure training and inference.

Although research on FaF security is relatively new, concurrent work has yielded encouraging results. For example, Koti et al. [KKPG22] proposed an efficient (1,1)-FaF secure five-party computation (5PC) protocol, while Hedge et al. [HKKPPP22] established the necessity of semi-honest oblivious transfer for FaF-secure protocols with optimal resiliency and introduced a ring-based 4PC protocol that guarantees fairness and GOD with one semi-honest and one malicious adversary.

2.2.2. Informal Technical Overview

Three-Round Weak FaF Construction Our three-round protocol uses decentralized threshold fully homomorphic encryption (dTFHE), following Gordon et al.'s approach [GLS15]. In the first round, participants exchange public keys. In round two encrypt their inputs under all the public keys, and broadcast the ciphertexts. In the third round, after each party has received the ciphertexts, they compute the function homomorphically and broadcast partial decryptions, allowing everyone to locally combine these partial decryptions to obtain the output. Gordon et al. showed that this protocol provides guaranteed output delivery with a dishonest minority. We prove that it also achieves weak FaF security when $2t + h^* < n$. Intuitively, security of such a dTFHE scheme ensures that the joint view of the active and passively corrupt parties comprising of $(t+h^*)$ secret keys is not enough to leak information about the inputs of the honest parties. Correctness of such a scheme ensures that even if up to t parties abort, guaranteed output delivery is achieved because the partial decryptions sent by the remaining $n - t > t + h^*$ parties suffice to compute the output.

Strong FaF Construction Our strong FaF construction is based on the BGW protocol [BGW88]. First, we show that BGW with Beaver triple preprocessing [Bea92] achieves guaranteed output delivery with an adaptive mixed adversary that can perform t fail-stop corruptions (similar to passive corruptions, except participants may also abort at any step) and h^* passive corruptions, provided $2t + h^* < n$. We then use Canetti et al.'s compiler [CLOS02], which involves adaptively secure commitments and zero-knowledge proofs, to allow t active corruptions and h^* passive corruptions. Finally, building on Alon et al.'s insight that adaptive security implies strong FaF, we obtain our result.

Relation of FaF to Other Notions We analyze the relationships between FaF, Best of Borth Worlds, and Mixed Adversary security. These notions seem very close but suprisingly they are incomparable. In Best of Both Worlds (BoBW) the adversary can make t active or $t + h^*$ passive corruptions but not both, and in Mixed Adversaries, the adversary can make t active and h^* passive corruptions. We present protocols that achieve one notion but not the others. Together with the findings of Alon et al., our

results illustrate that these security models are fundamentally distinct. We summarize these relationships in Figure 2.1 from [MRY23].

3. Faster MPC - Efficiency

3.1. PCGs, PCFs, and the Preprocessing Paradigm

In the introduction (Section 1.1) we talked about how modern secure multi-party computation (MPC) protocols are generally divided into two key phases: an offline phase that's independent of the function being computed and an online phase. In the offline phase the protocol participants work together to generate correlated randomness, which will be used later in the online phase. This correlated randomness enables efficient computation of the function of interest. The tradeoff that is being made is that this setup achieves a more efficient online phase at the cost of a resource-intensive preprocessing stage.

Building on the work by Boyle et al. [BCGI18; BCGIKS20b; BCGIKS19], major improvements have been made in the efficiency of generating correlated randomness, based on the inception of *pseudorandom correlation generators (PCGs)* and *pseudorandom correlation functions (PCFs)* [BCGIKS20a]. These tools enable parties to expand a short seed into larger pseudorandom correlations locally, boosting efficiency in MPC protocols (in the preprocessing model) by reducing the interaction cost during correlated randomness generation.

Recent years have seen numerous PCG and PCF constructions for various types of useful pseudorandom correlations, including oblivious transfer (OT) correlations [BCGIKS19; BCGIKRS19] and Beaver triple correlations [BCGIKS20b; BCCD23]. Generic frameworks for constructing PCGs and PCFs for a broad range of correlations rely on multi-key fully homomorphic encryption (FHE) or homomorphic secret sharing over circuits [DHRW16; BCGIKS19], and even support any efficiently computable correlations using indistinguishability obfuscation (iO) [DHRW16; ASY22]. However, these approaches tend not to yield practical efficiency. On the other hand, concrete efficiency can be achieved by constructing PCGs from variants of the Learning Parity with Noise (LPN) assumption or group-based assumptions. These PCGs, and even some PCFs [OSY21; BCGIKS20a; BCGIKRS22], are highly efficient but only cover a limited range of correlations, such as OT and Beaver triples.

Most PCG and PCF designs are restricted to generating correlations as *additive secret shares*, where each participant receives an additive share of the pseudorandom correlation. This restriction poses a barrier to generating certain complex correlations that cannot be naturally represented with additive secret shares. Examples include cases where pseudorandom permutation correlations or Shamir secret shares are required. Despite being natural and useful goals, even high-level primitives like multi-key FHE fall short of producing these non-additive correlations. The only known method for achieving such "non-additive" correlations involves indistinguishability obfuscation ($i\mathcal{O}$) paired with other cryptographic assumptions [DHRW16; ASY22].

Additionally, a PCF for pseudorandom permutations enables a non-interactive single secret leader election (SSLE) protocol, wherein the elected leader is the party that was assigned the lowest permutation value. This application can be valuable for proof-of-stake cryptocurrencies and other distributed systems [BEHG20].

3.2. Our Contributions

Our paper [CKMSS24] initiates the study of PCGs and PCFs specifically for pseudorandom permutation correlations.

Using an n-party pseudorandom permutation correlation, each party independently generates part of a pseudorandom permutation over the set $\{1, \ldots, n\}$, ensuring that no subset of $n-2$ corrupt parties can determine the entire permutation. A PCG (or PCF) for pseudorandom permutations can produce a large number, ℓ, of such correlations while keeping communication complexity low, e.g., scaling with $\mathsf{polylog}(\ell)$ after an initial setup.

Our contributions include designing three-party ($n = 3$) PCGs and PCFs using assumptions that are not known to imply $i\mathcal{O}$. Specifically, we construct a PCG for permutations using Quasi-Abelian Syndrome Decoding (a well-known LPN variant) and a PCF for permutations using Homomorphic Secret Sharing (HSS) [BGI16a; BCGIO17] for branching programs combined with a PRF in NC^1, which can be instantiated under the DCR [OSY21; RS21], LWE [BKS19], or class group assumptions [ADOS22]. While our HSS-based construction remains primarily theoretical, our LPN-based PCG offers practical efficiency which we demonstrate through benchmarking.

This work establishes the first concrete feasibility result for constructing PCGs/PCFs for non-additive, reverse-sampleable correlations without having to rely on heavy primitives such as $i\mathcal{O}$. Beyond the theoretical contributions, our constructions have practical applications in areas such as anonymous communication protocols and single secret leader election (SSLE). For instance, we demonstrate that a PCG can replace the costly "offline" preprocessing protocol used by Studholme and Blake [SB07] to generate pseudorandom permutations, enabling an efficient implementation of a Dining-Cryptography network for anonymous broadcast with optimal communication costs. Here, each participant uses a PCG to locally decide where to place their message on a shared bulletin board, ensuring that message origins remain anonymous. This pseudorandom permutation guarantees that each participant writes to a unique slot, eliminating the redundancy found in previous works [CBM15; GJ04].

Our results are summarized in Table 3.1 (from [CKMSS24]), where we compare our methods with prior approaches.

3.3. Informal Technical Overview

Background

To set the stage for our overview, we first provide an introduction to PCGs and PCFs. Formal definitions can be found in the detailed discussion in Section 8.3.

	Assumption	Target Correlation	# parties
PCF [DHRW16]	$i\mathcal{O}$	Any reverse-samplable	2
PCF [ASY22]	multi-key FHE + $i\mathcal{O}$	Any reverse-samplable	Any
PCF [ASY22]	multi-key FHE + $i\mathcal{O}$ + ROM	Any	Any
PCG (Section 8.4)	Quasi-Abelian SD	Permutations (biased)	3
PCF (Section 8.4)	HSS+PRF in NC^1	Permutations	3

Table 3.1.: Summary of PCGs and PCFs for non-additive correlations. "Reverse-samplable" refers to cases where, given the outputs of a sample's corrupt participants, the honest participants' outputs can be efficiently simulated to appear indistinguishable from the true sample.

A PCG for a two-party correlation $\mathcal{C}$ consists of two algorithms:

1. $\mathsf{Gen}(1^\lambda) \to (\mathsf{k}_0, \mathsf{k}_1)$: A randomized algorithm that, given a security parameter λ, generates a pair of short, correlated seeds $(\mathsf{k}_0, \mathsf{k}_1)$.

2. $\mathsf{Expand}(\mathsf{k}_\sigma) \to R_\sigma$: A deterministic algorithm that expands a seed k_σ into a long output R_σ.

For PCFs, the expansion algorithm Expand is replaced by an evaluation algorithm Eval, which takes an additional input x to produce an instance of the correlation.

The security requirement for PCGs is that the joint outputs (R_0, R_1) must be indistinguishable from the desired correlation $\mathcal{C}$, both to external observers and to each party that holds only one of the seeds. Access to a short seed should not allow an adversary to infer any additional information about the other party's pseudorandom string (other than what is inherently revealed by their own pseudorandom string). Slightly more formally, we require that an adversary, when given one of the short seeds k_σ, is unable to distinguish $R_{1-\sigma}$ from a randomly chosen pseudorandom string conditioned on (R_0, R_1) being correlated.

This security definition requires the target correlation to be *reverse sampleable*. This roughly says that it is possible to efficiently sample from the conditional distribution of $R_{1-\sigma}$ given R_σ. PCFs have a comparable security requirement, with the addition of needing the evaluation of R_0 and R_1 on-the-fly.

Examples of useful *additive*[1] correlations that have been studied in previous works are: random oblivious transfer (OT), oblivious linear function evaluation (OLE), and Beaver triples.

An example of a *non reverse-sampleable* correlation that is useful in preprocessing secure two party computation is the following: one party has a garbled circuit, and the other has the labels for the wires. The security guarantees of the garbling scheme do not allow for the labels to be reverse-sampled from the circuit meaning that this correlation is not reverse-sampleable.

[1]This means that the outputs (R_0, R_1) form additive secret shares of a sample from a distribution.

Main Ideas and Approach

Our three party construction, has the following parties: Alice, Bob, and Carol. We think that it will be beneficial to the reader if we first go over our initial attempt which ended up being unsuccessful. We will then mention the fixes that ultimately gave us our PCG and PCF construction.

First Try To illustrate a three-party permutation, we can start by interpreting the permutation over the set $\{0, 1, 2\}$ as providing pseudorandom shares of zero, given that the sum of a valid permutation's values is always 3, which is equivalent to zero in the field $\mathbb{F}_3$. However, while this can be correct, it doesn't ensure that what we have is indeed a permutation. For example, in the three-party case, if all zero shares are the same in $\mathbb{F}_3$, each party would hold the same output, which would not constitute a valid permutation. Thus, the first roadblock that we meet is that we need each party's share to be unique. We note that it's sufficient that the parties' shares $a + b + c = 0$ with $a \neq b$. This ensures that a, b, and c are pairwise distinct.

Ensuring Unique Shares Our initial insight is that, for two parties, say Alice and Bob, distinct shares can be generated by creating shares of a pseudorandom bit $\mu \in \{0, 1\}$ over $\mathbb{F}_3$. Now we can have Alice get $a \in \mathbb{F}_3$ and Bob get $b \in \mathbb{F}_3$, such that $a + b \pmod 3 = \mu$ and their output is a and $2 - b$ respectively then their values will be distinct because $2 - b = a + 2 - \mu \pmod 3$. Note that privacy is achieved because Alice has no way of knowing if Bob's output is $a + 1$ or $a + 2$.

With this first obstacle out of the way we can focus on Carol's share. Unfortunately, following the same approach wouldn't work out of the box. If we could use the same idea for Alice and Carol, ensuring Carol's output is distinct from both Bob's and Alice's without changing Alice's share, how would Carol's share correlate with Bob's to maintain uniqueness while keeping Alice's share consistent? This problem of coordinating the shares among three parties forms the core of our solution, and we provide two different methods for resolving it in our PCG and PCF constructions for permutations.

General Template for Construction Assuming we could coordinate the shares among Alice, Bob, and Carol as described, a general template for the three-party construction emerges. If Alice and Bob, as well as Alice and Carol, could each generate a pseudorandom *subtractive* share of a bit μ and $1 - \mu$, respectively, with Alice keeping the same share a in both cases, then Carol's share c would automatically be distinct from both Alice's and Bob's values. Expanding the parties' outputs, we have:

$$\textbf{Alice's output:} \quad a := x$$

$$\textbf{Bob's output:} \quad b := x + 2 - \mu$$

$$\textbf{Carol's output:} \quad c := x + 1 + \mu$$

Here, a, b, and c are pairwise distinct because:

- If $a = b$, then $\mu = 2$,
- If $a = c$, then $\mu + 1 = 0$,
- If $b = c$, then $2\mu = 1 \pmod 3$.

Each of these statements is false for $\mu \in \{0, 1\}$. In essence, the solution requires Bob and Carol to hold shares of pseudorandom bits μ and $1 - \mu$, respectively, with Alice's share remaining consistent across both μ and $1 - \mu$.

Overcoming Technical Challenges

Fixing Alice's Share Across Correlations In order for Alice's share to remain consistent across both instances of μ and $1 - \mu$ we need "programmability". Existing PCG and PCF constructions that allow for programmable correlations [BCGIKS19; BCGIKS20b] can ensure that μ remains consistent across both shares but even with programmability, Alice would receive two independent shares -one for each correlation- so the approach described before doesn't work. To resolve this, we introduce the concept of "doubly-programmable" PCGs and PCFs.

Doubly-Programmable PCGs and PCFs We extend the notion of programmability by creating a doubly-programmable PCG (or PCF) that allows one party's seed to be generated independently of the correlation itself. This enables us to fix Alice's seed k_0 while generating two separate keys, k_1 and k_2, corresponding to distinct correlations $\mathcal{C}_1$ and $\mathcal{C}_2$. Specifically, in our example, Alice can hold k_0 to generate a pseudorandom share x, while Bob and Carol hold k_1 and k_2 so that:

$$\mathsf{Expand}(\mathsf{k}_0) - \mathsf{Expand}(\mathsf{k}_1) = \mu \quad \text{and} \quad \mathsf{Expand}(\mathsf{k}_0) - \mathsf{Expand}(\mathsf{k}_2) = 1 - \mu.$$

We formalize this doubly-programmable feature by defining the PCGs and PCFs as $\mathcal{F}$-programmable, where one seed/key is generated independently of the target correlation and where the correlation can be expressed as a function $f(\cdot)$ of the pseudorandomness where $f \in \mathcal{F}$. In the above example we see how the programmed function in k_2 maps $x \mapsto 1 - x$.

Generating Pseudorandom Bits over Non-Binary Fields

The ability to generate pseudorandom bits in larger fields has significant applications in MPC, particularly for protocols that require converting shares from one field to another [IKNZ23; CS10; DFKNT06]. However, generating pseudrorandom bits in $\mathbb{F}_3$ poses unique challenges, as efficient protocols typically rely on degree-2 correlations meanwhile the correlation we need is computed by high degree polynomials.

In our work, we show that in $\mathbb{F}_3$, shares of a pseudorandom bit can be obtained by evaluating a degree-2 polynomial, albeit with a slight bias. This insight is crucial for our PCG construction, as it allows us to leverage existing degree-2 PCG techniques for biased pseudorandom bits.

To construct a PCF for pseudorandom permutations, we use homomorphic secret sharing (HSS) [BGI16a] to evaluate a high-degree polynomial that computes a PRF that outputs a bit. Although low-degree PRFs can help reduce the number of HSS multiplications needed, this approach remains more theoretical compared to the PCG construction.

Overview of Our PCG Construction

Our PCG construction for permutations leverages an $\mathcal{F}$-programmable PCG for generating pseudorandom bits. Starting from existing PCGs for degree-2 correlations in $\mathbb{F}_3$, such as those by Bombar et al. [BCCD23; BBCCDS24a], we use a *programmable distributed point function (DPF)* [BGIK22] to achieve $\mathcal{F}$-programmability.

Using this framework, we convert a pseudorandom degree-2 correlation into a biased pseudorandom bit by defining a bit μ as a share of $z^2 \pmod 3$, where z is pseudorandom. Since z^2 results in the set $\{0, 1\}$ but with a bias toward 1, we achieve an ε-biased PCG for permutations, suitable for multi-round anonymous broadcast.

Overview of Our PCF Construction

To avoid bias in our PCG construction, an $\mathcal{F}$-programmable PCF for generating unbiased pseudorandom bits in $\mathbb{F}_3$ would be ideal. For this, we combine HSS with an NC^1 PRF, enabling programmable PCFs for unbiased bits in $\mathbb{F}_3$ [2]. This technique, adapted from Couteau et al. [CMPR23], ensures efficient generation of pseudorandom bits without introducing bias. When paired with a low-degree PRF, this approach could bring the construction closer to practical use, with applications such as single secret leader election (SSLE), where the leader is the party with the lowest permutation value in an anonymous, non-interactive setting.

[2] Languages in NC^1 can be computed by bounded fan-in circuits of polynomial size and logarithmic depth or (alternatively) we can say that there is an efficient Turing machine that on input 1^n generates the n-th circuit.

4. Stronger MPC - Weaker Primitives

4.1. Pseudorandom Generators (PRGs)

Pseudorandom generators (PRGs) are functions that expand an input, such that the resulting output is computationally indistinguishable from a random string by any polynomial-time algorithm. A specific type of PRG are the *local pseudorandom generators (local PRGs)*, where the output depends only on a constant number of the input bits. Cryan and Miltersen [CM01] first investigated the feasibility of such local PRGs, with Applebaum, Ishai, and Kushilevitz [AIK04; AIK08] later demonstrating that PRGs in NC^0 with sublinear stretch can be constructed under widely accepted cryptographic assumptions, such as those based on the hardness of factoring or the discrete logarithm [1]. They also showed that local PRGs with linear stretch can be obtained under a specific hardness assumption related to decoding sparsely generated linear codes.

Recently, local PRGs with *polynomial stretch* have proven useful in numerous cryptographic and non-cryptographic applications. Examples include their use in secure computation with constant computational overhead [IKOS08], indistinguishability obfuscation [JLS21; JLS22], pseudorandom correlation generators and functions [BCGIO17; BCMPR24], public-key encryption [BKR23], and sublinear secure computation [BCM23]. Outside of cryptography we find them applied to prove hardness-of-learning results [DV21]. As a result, both the construction of polynomial-stretch local PRGs and the cryptanalysis of existing candidates have been an active research focus [Gol00; MST03; BQ09; App12; OW14; CEMT14; App15; ABR16; AL16; LV17; CDMRR18; AK19; OST19; Méa; YGJL21; Méa22; Üna23b; DMR23; Üna23a]. All existing local PRG candidates stem from the proposal by Goldreich in [Gol00], which applies a specific predicate P to constant-size subsets of the input bits, where these subsets are formed as hyperedges of a sufficiently expanding uniform hypergraph.

In this paper, we re-evaluate the scope and use of local PRGs and make a key observation: many established applications of local PRGs don't require the full capabilities of local PRGs. In particular, a number of applications only need a local pseudorandom mapping from n-bit seeds to m-bit strings but *do not require that the seeds are sampled uniformly at random*. We formalize this observation by introducing a new concept we call structured-seed local pseudorandom generators.

[1] A language is in NC^0 if it can be decided by uniform boolean circuits with constant depth and bounded fan-in.

4.2. Our Contributions

The notion of *structured-seed local PRGs* extends local PRGs to cases where the seed is sampled from a specific distribution supported over $\{0,1\}^n$ (rather than being purely random). We provide examples where structured-seed local PRGs can seamlessly replace traditional local PRGs. Specifically, we illustrate the use of structured-seed local PRGs in:

1. Indistinguishability obfuscation based on well-established assumptions [JLS21];
2. Secure computation with constant overhead [IKOS08];
3. Compact homomorphic secret sharing [BCM23];
4. Learning hardness for disjunctive normal forms (DNFs) [DV21].

In addition to defining structured-seed local PRGs, we propose constructions based on well-known cryptographic assumptions that do not currently imply the existence of standard local PRGs. Specifically, we focus on the *sparse learning parity with noise (sparse-LPN)* assumption, originally introduced by Alekhnovich [Ale03], which is equivalent to the problem of decoding random low-density parity-check (LDPC) codes. We present a range of structured-seed local PRG constructions derived from different variants of this assumption, obtaining:

1. A direct structured-seed local PRG construction from the sparse-LPN assumption with a regular noise distribution (where noise is generated by concatenating random one-hot vectors).
2. A structured-seed local PRG construction with inverse-polynomial security based on the sparse-LPN assumption with more general noise distributions. This approach leverages advanced hashing schemes for balanced allocation and is more technically involved.

As a result, we demonstrate that, for the four applications listed above, assumptions about local PRGs can be replaced with assumptions regarding the hardness of sparse-LPN with regular noise (for indistinguishability obfuscation, this requires assuming the subexponential hardness of sparse-LPN). For the DNF learning application, where inverse-polynomial security suffices, we further derive hardness results from the sparse-LPN assumption without requiring regular noise.

4.3. Structured-Seed Local PRGs

We formally define *structured-seed local PRGs* and discuss the primitives that we use to build them. A *structured-seed* pseudorandom generator relaxes "traditional" pseudorandom generators to allow for more general distributions of seeds. Instead of sampling r uniformly over $\{0,1\}^n$, we sample it as $r \xleftarrow{\$} \mathsf{SampleSeed}$. The properties that we require are:

- Small size. The support $\mathsf{Supp}(\mathsf{SampleSeed})$ of $\mathsf{SampleSeed}$ is contained in $\{0,1\}^n$, and
- Efficiency. The running time of the sampler $\mathsf{SampleSeed}$ is much smaller than m.

We provide a formal definition (which is Definition 9.4 in [BCM24]) below.

Definition 4.1 (Structured-Seed Local Pseudorandom Generator)**.** A structured-seed pseudorandom generator with a stretch $m(\cdot)$ is a triple of uniform PPT algorithms $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$, with:

- $\mathsf{Setup}(1^\lambda)$. A probabilistic algorithm that on inputs 1^λ and outputs a public parameter pp.
- $\mathsf{SampleSeed}(\mathsf{pp})$. A probabilistic algorithm that on inputs pp and outputs a seed value $\mathsf{seed} \in \{0,1\}^n$.
- $\mathsf{PRG}(\mathsf{pp}, \mathsf{seed})$. A deterministic algorithm that on inputs seed, public parameter pp and outputs an evaluation value $y \in \{0,1\}^{m(n)}$.

A structured-seed pseudorandom generator is (T, ε, δ)-*secure* if for any *non-uniform* p.p.t adversary $\mathcal{A} = (\mathcal{A}_\lambda)_{\lambda\in\mathbb{N}}$ of size at most $T = T(\lambda)$, for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Adv}_{\mathcal{A}_\lambda}(\mathcal{D}_0, \mathcal{D}_1) > \varepsilon\right] \leq \delta,$$

where $\mathcal{D}_0 = \mathcal{D}_0^{\lambda,n}$ denotes the family of distributions

$$\{(\mathsf{pp}, \mathsf{PRG}(\mathsf{pp}, r)) \mid \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda), r \xleftarrow{\$} \mathsf{SampleSeed}(\mathsf{pp})\}$$

and $\mathcal{D}_1 = \mathcal{D}_1^{\lambda,n}$ denotes the family of distributions

$$\{(\mathsf{pp}, z) \mid \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda), z \xleftarrow{\$} \{0,1\}^{m(n)}\}.$$

Related Work There is one recent parallel work in which Ragavan, Vafa, and Vaikuntanathan [RVV24] also introduced the concept of structured-seed local PRGs. Our work is concurrent and developed independently from theirs, with considerable overlap in some findings: both works make the central observation that structured-seed local PRGs can substitute local PRGs in certain applications, and both define structured-seed local PRGs in a similar way. Some of the differences can be summarized as follows: (1) In [RVV24] the main goal is to apply structured-seed local PRGs to indistinguishability obfuscation (iO). Although we also discuss iO, their analysis is more comprehensive and achieves stronger results, specifically by replacing local PRGs with structured-seed local PRGs in [JLS22] rather than in [JLS21], meaning that they can bypass the need for the LWE assumption. (2) The other applications that we study, such as secure computation and hardness of learning, are not covered in [RVV24]. While our current exploration of these applications is preliminary, we do a much more thorough analysis of the hardness-of-learning application in future revisions, as it presents unique technical challenges. (3)

Finally, in [RVV24] the primary structured-seed local PRG construction uses sparse-LPN with Bernoulli noise. In contrast, we consider different noise distributions, such as regular noise and XOR noise. Consequently, while some of our constructions share underlying principles, they differ significantly in implementation and technical approach.

LPN Assumptions Informally, the Learning Parity with Noise (LPN) assumption over the binary field $\mathbb{F}_2$ states, that no efficient adversary can distinguish between the pairs $(A, A \cdot \mathbf{x} + \mathbf{e})$ and $(A, \mathbf{b})$, where A is sampled from a certain matrix distribution $\mathcal{M}$ over $\mathbb{F}_2^{n \times k}$. Here, $\mathbf{x}$ is drawn uniformly from $\mathbb{F}_2^k$, and $\mathbf{e}$ represents a noise vector sampled from a noise distribution $\mathcal{E}$ that typically generates sparse vectors over $\mathbb{F}_2$. In contrast, $\mathbf{b}$ is chosen as a uniform vector over $\mathbb{F}_2^n$. Formally, the LPN assumption over $\mathbb{F}_2$ with parameters k, n, and matrix and noise distributions $\mathcal{M}$ and $\mathcal{E}$ can be stated as follows:

Definition 4.2 (Learning Parity with Noise (LPN))**.** Let $k \in \mathbb{N}$ be an integer, and let $n = n(k)$ be a polynomial function of k. Let $\mathcal{M} = \mathcal{M}_{n,k}$ be a distribution over matrices in $\mathbb{F}_2^{n \times k}$, and let $\mathcal{E} = \mathcal{E}_n$ be a noise distribution over $\mathbb{F}_2^n$. We say that the $(\mathcal{M}, \mathcal{E})$-LPN problem is (T, ε, δ)-hard if, for any probabilistic adversary $\mathcal{A} = (\mathcal{A}_\lambda)_{\lambda \in \mathbb{N}}$ of size at most $T = T(\lambda)$, it holds that for sufficiently large k:

$$\Pr_{A \xleftarrow{\$} \mathcal{M}_{n,k}} \left[\mathsf{Adv}_{\mathcal{A}_k}(\mathcal{D}_0^A, \mathcal{D}_1^A) > \varepsilon \right] \leq \delta,$$

where $\mathcal{D}_0^A$ denotes the distribution $\{(A, A \cdot \mathbf{x} + \mathbf{e}) \mid \mathbf{x} \xleftarrow{\$} \mathbb{F}_2^k, \mathbf{e} \xleftarrow{\$} \mathcal{E}\}$ and $\mathcal{D}_1^A$ denotes $\{(A, \mathbf{b}) \mid \mathbf{b} \xleftarrow{\$} \mathbb{F}_2^n\}$.

Discussion on the Definition Definition 4.2 (which is Definition 9.1 in [BCM24]) is not the same as the traditional LPN definition. Typically, LPN is defined such that an adversary's advantage in distinguishing the distributions $\{(A, \mathbf{b}) \mid A \xleftarrow{\$} \mathcal{M}_{n,k}, \mathbf{b} \xleftarrow{\$} \mathcal{D}_0^A\}$ and $\{(A, \mathbf{b}) \mid A \xleftarrow{\$} \mathcal{M}_{n,k}, \mathbf{b} \xleftarrow{\$} \mathcal{D}_1^A\}$ is bounded by ε, where ε is negligible in the security parameter λ, and the probability of success is taken over both the adversary's random choices and the random selection of A. However, in Definition 4.2, we tweak the definition as follows: δ represents the probability that a matrix A is "good" for the LPN problem, while ε captures the probability that an adversary, given such a matrix A, can successfully distinguish LPN samples from random samples. By setting both (ε, δ) to be negligible, we immediately get the standard LPN definition but using this tweaked definition becomes very useful in analyzing the *sparse-LPN* assumption, which is discussed further in Section 9.4.

Types of Noise In the LPN context, the properties of the noise distribution play a significant role in determining the hardness of the problem. Here, w is a parameter that represents the expected density of nonzero entries in a typical noise vector $\mathbf{e}$. Several standard choices of noise distribution include:

- **Bernoulli noise:** Each entry of $\mathbf{e}$ is chosen independently according to a Bernoulli distribution with parameter w/n.

- **Exact noise:** The noise vector $\mathbf{e}$ is uniformly sampled from the set of vectors with a fixed Hamming weight w.
- **Regular noise:** The vector $\mathbf{e}$ is constructed by concatenating w randomly selected one-hot vectors.
- **XOR noise:** Here, $\mathbf{e}$ is formed as the XOR of w randomly selected one-hot vectors of length n. This distribution is particularly useful for security analyses since the one-hot vectors are mutually independent.

We formally denote these distributions as follows:

- $\mathcal{B}_{n,w}$: the distribution over $\mathbb{F}_2^n$ in which each bit is independently set to 1 with probability w/n.
- $\mathcal{S}_{n,w}$: the uniform distribution over vectors in $\mathbb{F}_2^n$ with exactly w non-zero bits.
- $\mathcal{R}_{n,w}$: generated by concatenating w one-hot vectors sampled over $\mathbb{F}_2^{n/w}$ (assuming w divides n).
- $\mathcal{X}_{n,w}$: generated by taking the XOR of w one-hot vectors of length n.

In the "low-noise" regime, where $w \ll \sqrt{n}$, the LPN variants with noise sampled from $\mathcal{S}_w$ and $\mathcal{X}_w$ are nearly equivalent. This is because a sample from $\mathcal{X}_w$ has a Hamming weight of exactly w with high probability. Moreover, it is known that LPN with $\mathcal{S}_w$ noise is equivalent to LPN with $\mathcal{B}_w$ noise [Pie12]. While there are also reductions between LPN with regular noise and other noise types, these typically come at a higher parameter cost [LWYY24].

4.4. Applications

In this section we look into works that utilize a PRG in order to achieve their results and we study the feasibility of replacing their PRG with our structured-seed local PRG construction. These applications we study are:

- Indistinguishability obfuscation (iO) [JLS21]
- Constant-overhead secure computation [IKOS08]
- Sublinear secure computation [BCM23]
- Hardness of learning [DV21]

The reason why our structured-seed local PRG can replace their PRG is that in the aforementioned works, the fact that the seed is sampled uniformly at random is not actually used. They only require a *short* seed that can be sampled *efficiently*.

In other works, for example ones that employ some GGM style composition where the output of the PRG is being used as input for the next invocation of the PRG, our work can't be applied since the PRG output doesn't have the structure that we need anymore.

4.4.1. Indistinguishability obfuscation

Indistinguishability obfuscation (iO) aims to obscure program code such that no polynomial-time adversary can distinguish which of two functionally identical programs has been obfuscated. This concept was formalized as a cryptographic building block in the early 2000s by Hada [Had00] and Barak *et al.* [BGIRSVY01], with both positive [Can97; LPS04; Wee05; HRsV07; HMS07] and negative [BGIRSVY01; GK05; Wee05] results emerging in the early literature. Recently, a series of breakthrough results, culminating in [JLS21] by Jain, Lin, and Sahai, demonstrated the construction of iO based on the subexponential hardness of four assumptions:

- The LWE assumption,
- Learning parity with noise over a general prime field $\mathbb{F}_p$,
- A boolean local PRG in NC^0,
- The Decision Linear assumption on symmetric bilinear groups of prime order.

This iO framework relies on a series of transformations, beginning with weaker forms of functional encryption and progressively advancing to full indistinguishability obfuscation. Notably, the local PRG in their construction is leveraged to build a *structured-seed PRG*. While the structured-seed PRG in [JLS21] is different from the structured-seed local PRG notion introduced in our work, their construction can proceed equivalently if the boolean local PRG is substituted with a structured-seed local PRG.

But it's not just plug-and-play. The local PRG in [JLS21] requires subexponential security. Our structured-seed PRG based on regular sparse-LPN can be conjectured to meet this need against subexponential-time adversaries if we keep ε inverse-subexponential, as suggested by an assumption we make on a concrete parametrized version of sparse-LPN. Our assumption, states that no attacks on sparse-LPN do better than linear tests. However, the value of δ remains non-negligible due to the inherent chance of sampling a matrix with low dual distance. By using the matrix distribution proposed in [AK19], we can make δ negligible but only *slightly*, which doesn't fully work because the construction needs δ to be subexponentially small.

"Traditional" local PRGs have similar issues, since they require an explicit hypergraph with strong expansion properties, while randomly generated hypergraphs meet the necessary condition only with probability $1 - 1/\mathsf{poly}$. Two potential solutions exist: the first was the solution that was essentially used in [JLS21], where a specific sparse matrix A is selected, assuming the subexponential hardness of sparse-LPN with respect to this matrix. Then, by the assumption we mentioned above, *most* matrix choices should yield feasible candidates, providing a non-uniform iO construction (which could become uniform if an efficiently sampleable distribution over sparse matrices with a subexponentially small dual distance probability is found).

The second approach involves sampling *multiple* parameter sets pp for the structured-seed local PRG, such that at least one parameter set will likely offer security against subexponential adversaries, except with subexponentially small probability. As shown

in [JLS22], this strategy produces a collection of functional encryption schemes from which at least one will likely have subexponential security. These schemes can be combined to create a robust functional encryption scheme using FE combiners. We obtain the following result:

Theorem 4.1 (informal)**.** *Assume sub-exponential security of the following assumptions:*

- *The LWE assumption,*
- *Learning parity with noise over a general prime field $\mathbb{F}_p$,*
- *The sparse-LPN assumption with regular noise,*
- *The Decision Linear assumption on symmetric bilinear groups of prime order.*

Then, there exists (subexponentially secure) indistinguishability obfuscation for all polynomial-size circuits. Assuming only polynomial security of the above assumptions, there exists collusion-resistant public-key functional encryption for all polynomial-size circuits.

Note that in [JLS22] the LWE assumption is removed by utilizing the local PRG in a more involved way. Direct replacement of this local PRG with a structured-seed local PRG is nontrivial, as it involves an affine randomized encoding that reuses the PRG's output as a seed, which does not align with structured-seed PRGs. While we initially considered exploring this more, recent independent work by [RVV24] has fully addressed this challenge, so we decided to not pursue this direction any further.

4.4.2. Constant-overhead secure computation

Ishai, Kushilevitz, Ostrovsky, and Sahai [IKOS08] demonstrated that under the assumption of polynomial-stretch local pseudorandom generators (along with oblivious transfers), any two-party functionality can be securely computed with *constant computational overhead* over the clear evaluation cost. In our work, we show that the local PRG in [IKOS08] can be replaced with a structured-seed local PRG, as summarized in Theorem 4.2.

Theorem 4.2. *Assuming the existence of a polynomial-stretch structured-seed local PRG in NC^0, denoted as $G : \{0,1\}^n \to \{0,1\}^m$, and of a standard OT protocol, for a circuit family $C = \{C_n\}$ with polynomial size $s(n)$ defining a two-party computation functionality f, there exists a two-party protocol π_f that realizes f in the semi-honest setting, with each party in π_f implemented by a circuit of size $O(s(n))$.*

Thus, secure two-party computation with constant overhead can be based on OT and the hardness of regular sparse-LPN expanding the assumptions supporting efficient secure computation.

Although this is an informal overview, we will mention a little bit of the proof of Theorem 4.2, because it makes it clear why our structured-seed local PRG can be used. The protocol in [IKOS08] involves (very roughly) these steps:

1. Constant-overhead secure computation reduces to constructing $O(s)$ OTs with constant overhead. In particular, for $p = O(s)$ they show how to construct p bit-OTs using a local PRG and $\sqrt{p}$ OT instances.

2. Let g be a parameterized parameterized by a local PRG G. With black-box access to g, a secure protocol to generate $O(s)$ bit-OTs follows. This involves using $G(\mathsf{seed})$ in place of random masks to hide selection bits, achieved in linear time with derandomization.

3. A key component is reducing g's implementation to $\sqrt{p}$ OTs on strings of total length $O(p)$ via decomposable randomized encodings.

The substitution of G with a structured-seed local PRG is straightforward in steps 1 and 3, as g's hardcoded stretching algorithm is NC^0 and independent of seed's sampling. But in step 2 the receiver must sample $\mathsf{seed} \xleftarrow{\$} \mathsf{SampleSeed}(\mathsf{pp})$ instead of a random seed, which is computationally trivial.

There are a couple of things to consider here. The protocol in [IKOS08] assumes a quadratic-stretch local PRG, which any polynomial-stretch PRG could achieve by composing with itself. Unfortunately, structured-seed local PRGs achieve only *near* quadratic stretch and do not self-compose, but we can tweak [IKOS08] to accommodate smaller polynomial stretch. Secondly, $O(p^{1/2+\varepsilon})$ string-OTs with constant overhead are realized by sampling ℓ-bit seeds for the local PRG and using string-OTs to send one seed to the receiver, who will then unmask a pair. This cost is polynomially small for large p, and we conclude that polynomial stretch structured-seed PRG suffice for the reduction.

4.4.3. Sublinear Secure Computation and Compact HSS

The notion of *Homomorphic Secret Sharing (HSS)*, introduced in [BGI16a], offers a compelling alternative to fully homomorphic encryption by enabling secure computation with sublinear communication overhead. The main idea behind an N-party HSS scheme for a class of functions $\mathcal{F}$ is the splitting of x into multiple shares $x_1, x_2, \ldots, x_N$, distributed among the parties. Each party can independently compute a share of the output y_i, such that the outputs $(y_1, \ldots, y_N)$ form additive shares of $y = f(x)$, for a function $f \in \mathcal{F}$. A *compact* HSS scheme achieves this with shares and a sharing algorithm of size $O(|x|) + \mathsf{poly}(\lambda)$, where λ is the security parameter. When paired with a generic MPC protocol that has linear communication overhead, compact HSS enables secure N-party protocols with optimal communication complexity $O(N \cdot (|x|+|y|)) + \mathsf{poly}(\lambda)$ for all $f \in \mathcal{F}$.

A popular strategy for constructing compact HSS schemes is the "hybrid encapsulation" approach. Instead of directly sharing the input x, parties share a shorter seed seed using HSS and for a pseudorandom generator (PRG) G, the value $u = x \oplus G(\mathsf{seed})$ is made public. If G runs in linear time, the size of the shares and the runtime of the sharing process are bounded by $O(|x|) + \mathsf{poly}(\lambda)$. To compute shares of $f(x)$, the parties evaluate $g_u(\mathsf{seed}) := f(u \oplus G(\mathsf{seed})) = f(x)$ homomorphically. This approach works as long as

$g_u \in \mathcal{F}$, meaning that the PRG G must also be efficient and belong to a low complexity class when the HSS scheme supports a minimal function class $\mathcal{F}$.

Recent work by [BCM23] has shown that sublinear secure computation and compact HSS can be constructed under new assumptions. By new we mean *new to the specific application* and not new to cryptography in general. They show:

Theorem 4.3 (Theorem 32 in [BCM23])**.** *Assuming the superpolynomial hardness of the DCR problem and the existence of PRGs with constant locality, there exists a four-party HSS scheme for loglog-depth circuits with n inputs, where the share size is $n \cdot (1 + o(1))$. Moreover, there is a protocol with communication complexity $n \cdot (4 + o(1))$ (for sufficiently large n) to securely realize the four-party functionality generating HSS shares of the concatenated party inputs.*

In a seamless way, Theorem 4.3 can be adapted to using a *structured-seed* constant-locality PRG with any small polynomial stretch. Specifically, each participant independently generates a short seed seed_i, and a secure computation protocol processes their combined input $(\mathsf{seed}_1||\mathsf{seed}_2||\mathsf{seed}_3||\mathsf{seed}_4)$. The difference is that parties need to run $\mathsf{SampleSeed}$ instead of uniformly sampling a seed. Importantly, this adjustment preserves the protocol's correctness, security properties, and communication efficiency which means that we can have a four-party HSS scheme for loglog-depth circuits with n inputs, where the small share size. Furthermore, a protocol with communication complexity $n \cdot (4 + o(1))$ (for sufficiently large n) can securely realize the four-party functionality generating HSS shares of the concatenated inputs.

Combining this with the compiler from [BCM23] for N-party compact HSS to $(N + 1)$-party secure computation with sublinear communication we get a five-party protocol with sublinear communication complexity $O(s/\log\log s)$ for layered circuits of size s under the assumptions above.

4.4.4. Hardness of Learning

Probably Approximately Correct (PAC) learning [Val84] is the process of identifying a hypothesis that can accurately predict the output of an unknown function class with high probability. In this model, a learner aims to approximate an unknown target function f from a class of functions $\mathcal{H}$, given access to labeled examples $(x, f(x))$, where x is drawn from an unknown distribution $\mathcal{D}$. A hypothesis $h \in \mathcal{H}$ is said to be a PAC solution if, for any distribution D over the input space and any $\varepsilon > 0$ and $\delta > 0$, the learner outputs h such that the probability of h having an error greater than ε (i.e., $\Pr_{x\sim D}[h(x) \neq f(x)] > \varepsilon$) is at most δ. Formally,

Definition 4.3 (PAC Learning)**.** A class of functions $\mathcal{H}$ is PAC-learnable if there exists an algorithm $\mathcal{A}$ such that, for every distribution D, $\varepsilon > 0$, and $\delta > 0$, $\mathcal{A}$ outputs a hypothesis h satisfying:

$$\Pr[\Pr_{x\sim D}[h(x) \neq f(x)] \leq \varepsilon] \geq 1 - \delta,$$

given a number of labeled examples that is polynomial in $1/\epsilon$, $1/\delta$, and the complexity of $\mathcal{H}$.

Hardness of learning establishes the difficulty for learning algorithms to generate such hypotheses. In [DV21], Daniely and Vardi provide several hardness-of-learning results based on the existence of local PRGs with polynomial stretch and constant distinguishing advantage. We show how our structured-seed local PRG can substitute the PRG in their results, leading to hardness-of-learning conclusions from the sparse-LPN assumption.

Definition 4.4 (Predicate)**.** Given a structured-seed ℓ-local PRG $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ with input size k and stretch n, we let P denote the predicate such that for each $i \in [n]$, there is a subset $S_i \subset [k]$ of size $|S_i| \leq \ell$ such that for all $x \in \mathsf{Supp}(\mathsf{SampleSeed}(\mathsf{pp}))$, defining $y = \mathsf{PRG}(x)$, we have $y_i = P(x[S_i])$.

Generally, for ℓ-local PRG we have that each output bit y_i depends on a predicate P_i and a subset S_i of size ℓ of the bits of the seed x, where $y_i = P_i(x[S_i])$. However, when the PRG has polynomial stretch then we can just assume a single predicate P. Because there are at most 2^{2^ℓ} possible predicates P_i for ℓ-bit inputs, and ℓ is constant, choosing P as the most frequent P_i and retaining only bits generated with P yields an ℓ-local PRG with polynomial stretch, reduced by at most 2^{2^ℓ}. Additionally, this is works perfect for our construction because it uses a single global predicate.

DNFs We prove that DNF formulas with $\omega(1)$ terms cannot be efficiently PAC-learned, assuming the sparse-LPN assumption. Informally this says that for any $q(n) = \omega(1)$, there is no efficient algorithm that PAC-learns DNF formulas with n variables and $q(n)$ terms.

We note that unlike other applications discussed here, we only require a structured-seed local PRG with constant (ε, δ). Therefore, we can directly rely on sparse-LPN instead of regular sparse-LPN. However, the result critically requires a local PRG with *arbitrary polynomial stretch*, achievable under sparse-LPN using the construction in Section 9.5.11.

The proof of Theorem 3.1 in [DV21] hinges on a clever insight: let ℓ be constant, and let $P : \{0,1\}^\ell \to \{0,1\}$ be an ℓ-local predicate. For any subset $S = \{s_1, \ldots, s_\ell\} \subset [n]$, set $\mathbf{v_i} := \mathsf{unit}_n(s_i) \in \{0,1\}^n$ for $i = 1$ to ℓ. Then, define the formula ψ as follows:

$$\psi(\mathbf{v_1}, \cdots, \mathbf{v}_\ell) = \bigvee_{\mathbf{x}: P(\mathbf{x})=1} \bigwedge_{i \leq \ell} \bigwedge_{j: \mathsf{seed}_j \neq x_i} \bar{v}_{i,j}.$$

We have

$$\begin{aligned}
\psi(\mathbf{v_1}, \cdots, \mathbf{v}_\ell) = 1 &\iff \exists \mathbf{x} \in P^{-1}(1), \forall i \leq \ell, \forall \mathsf{seed}_j \neq x_i, \bar{v}_{i,j} = 1 \\
&\iff \exists \mathbf{x} \in P^{-1}(1), \forall i \leq \ell, \forall \mathsf{seed}_j \neq x_i, \mathsf{unit}_n(s_i)_j = 0 \\
&\iff \exists \mathbf{x} \in P^{-1}(1), \forall i \leq \ell, \forall \mathsf{seed}_j \neq x_i, s_i \neq j \\
&\iff \exists \mathbf{x} \in P^{-1}(1), \forall i \leq \ell, \mathsf{seed}_{s_i} = x_i \\
&\iff \exists \mathbf{x} \in P^{-1}(1), \mathsf{seed}_S = \mathbf{x} \\
&\iff P(\mathsf{seed}_S) = 1.
\end{aligned}$$

This shows that, given a predicate P and a seed $\mathbf{x}$, $(P, \mathbf{x})$ can be hardcoded into a DNF ϕ such that for any size-ℓ subset S, there is an encoding $\mathsf{Encode}(S) = (\mathbf{v_1}, \cdots, \mathbf{v}_\ell)$ with $\psi(\mathsf{Encode}(S)) = P(\mathbf{x}[S])$.

To adapt the proof of Theorem 3.1 from [DV21] to structured-seed local PRGs, we make the following assumption:

For every constant $s > 1$, there exists a constant ℓ such that a $(T, 1/6, 1/6)$-secure structured-seed ℓ-local PRG $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ with predicate P exists, mapping k bits to k^s bits, for any $T = \mathsf{poly}(\lambda)$.

By Theorem 9.5.11 (which roughly says that the stretch of our construction can be extended to an arbitrary polynomial via self-composition), the assumption above is implied by the sparse-LPN assumption. Let $\mathcal{A}$ be a PPT adversary that PAC-learns DNF formulas with k variables and $q = \omega(1)$ terms. Define Q as the number of DNF oracle queries by $\mathcal{A}$, and set s such that $k^s > 100Q^2$. Define the distribution $\mathcal{D}$ as follows:

- Sample $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
- For any $i \leq k^s$, let S_i denote the size-ℓ subset of bits from seed used by $\mathsf{PRG}_{\mathsf{pp}}(\mathsf{seed})$ (noting that S_i is independent of seed, though it may depend on pp).
- Define $\mathcal{D} = \mathcal{D}_{\mathsf{pp}}$ as the distribution sampling $i \xleftarrow{\$} [k^s]$ and outputs $z = \mathsf{Encode}(S_i)$.

Sample $\mathsf{seed} \leftarrow \mathsf{SampleSeed}(\mathsf{pp})$, and let ψ encode the computation of $\mathsf{PRG}_{\mathsf{pp}}(\mathsf{seed})_i = P(\mathsf{seed}_{S_i}) = \psi(\mathsf{Encode}(S_i))$. ψ is a DNF formula with at most 2^ℓ terms. Given Q samples $(z_i, \psi(z_i))_{i \leq Q}$ (the training set), adversary $\mathcal{A}$ outputs a hypothesis h, with low error on the training set with small probability. With probability at most $1/100$, no query collisions occur, meaning that distinguishing the next sample $(z_{Q+1}, \psi(z_{Q+1}))$ from random happens with high probability.

5. This Thesis

5.1. Papers and Contributions

Parts II, III and IV are based on research papers written jointly with my co-authors [BMRS24; MRY23; BCM24; CKMSS24]. The content for [BMRS24; MRY23; BCM24] is taken from the full versions of the papers that are published on ePrint. The text for [CKMSS24] is not public yet. It is under submission at PKC '25. Unless explicitly mentioned, there have been no modifications to the texts other than correcting typographical errors, adjusting layout (mostly tables and boxes) to work in this LaTeX book format, adjusting sections and subsections for better readability, and moving content from the appendices to the appropriate sections in the main body.

Chapter 6: MPC with Identifiable Abort
This chapter is based on the ePrint version [BMRS23] of the paper:

> [BMRS24] Carsten Baum, Nikolas Melissaris, Rahul Rachuri, and Peter Scholl. *Cheater Identification on a Budget: MPC with Identifiable Abort from Pairwise MACs*. In: *Advances in Cryptology – CRYPTO 2024, Part VIII*. ed. by Leonid Reyzin and Douglas Stebila. Vol. 14927. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, Aug. 2024, pp. 454–488. DOI: 10.1007/978-3-031-68397-8_14

The idea for this paper was due to Carsten Baum and Peter Scholl who have an established line of work on this subject. I wrote parts of Sections 6.1, 6.4, and 6.5.

Chapter 7: MPC with Friends and Foes This chapter is based on the ePrint version [MRY22] of the paper:

> [MRY23] Nikolas Melissaris, Divya Ravi, and Sophia Yakoubov. *Threshold-Optimal MPC with Friends and Foes*. In: *Progress in Cryptology - INDOCRYPT 2023: 24th International Conference in Cryptology in India, Part II*. ed. by Anupam Chattopadhyay, Shivam Bhasin, Stjepan Picek, and Chester Rebeiro. Vol. 14460. Lecture Notes in Computer Science. Goa, India: Springer, Cham, Switzerland, Dec. 2023, pp. 3–24. DOI: 10.1007/978-3-031-56235-8_1

The idea for this paper is due to Divya Ravi and Sophia Yakoubov. I wrote parts of Sections 7.2, 7.3, and 7.6.

Chapter 8: Compressing Pseudorandom Permutation Correlations
This chapter is based on a paper that is under submission at PKC '25:

> [CKMSS24] Geoffroy Couteau, Alexander Koch, Nikolas Melissaris, Sacha Servan-Schreiber, and Peter Scholl. *Compressing Pseudorandom Permutation Correlations*. In Submission. 2024

The idea for this work is due to Geoffroy Couteau and Sacha Servan-Schreiber. I wrote parts of Sections 8.1, 8.2, 8.3, and 8.4.

Chapter 9: Structured-Seed Local Pseudorandom Generators and their Applications

This chapter is based on [BCM24] which is uploaded on ePrint:

> [BCM24] Dung Bui, Geoffroy Couteau, and Nikolas Melissaris. *Structured-Seed Local Pseudorandom Generators and their Applications*. Cryptology ePrint Archive, Report 2024/1027. 2024. URL: https://eprint.iacr.org/2024/1027

The idea for this work is due to Geoffroy Couteau.
I wrote parts of Section 9.7.

Part II.

Better Security Guarantees for MPC

6. MPC with Identifiable Abort

6.1. Introduction

Secure multiparty computation (MPC) is a class of cryptographic protocols allowing a group of distrusting parties to jointly compute a function over their private inputs, without revealing anything beyond the output of the computation. While many different factors impact the usability of MPC protocols, one of the most important security-wise is the corruption threshold. It provides a limit on how many of the participants can collude and share their information, without losing the privacy guarantees of MPC. Many popular protocols, such as SPDZ [DPSZ12], BDOZ [BDOZ11] and their follow-up works ensure privacy, even if $n-1$ out of the n participants are corrupted, and even when attackers may actively deviate from the protocol.

Nevertheless, privacy is not the only security guarantee that an MPC protocol may have to achieve. Fairness requires that if the corrupted parties obtain the output, then so do the honest parties. It is known [Cle86] that in the dishonest majority setting (i.e. when $\geq n/2$ parties are corrupted) we cannot achieve fairness, or the even stronger notion of guaranteed output delivery. Therefore, current highly efficient protocols settle for a weaker notion of security: security with abort. This typically means that a corrupt party can force the protocol to abort, so that some (or all) of the honest parties will abort instead of learning the correct output.[1]

Identifiable Abort for MPC.

Since fairness is impossible in the dishonest majority setting, the *next best* property would be if, in the case that the protocol aborts, the honest parties agree that the protocol aborted and also agree on the identity of at least one corrupt party. This can work as a deterrent since honest parties can exclude said corrupt party if they restart the computation. This property is called *identifiable abort*.

Cheater identification in dishonest-majority MPC (ID-MPC) was first formally studied by Ishai, Ostrovsky and Seyalioglu [IOS12], who showed that it is impossible to build unconditionally secure ID-MPC in a model with a broadcast channel and any pairwise ideal functionality, such as oblivious transfer (OT). This is in contrast to the secure-with-abort model, where pairwise OT suffices. Later, Ishai, Ostrovsky and Zikas [IOZ14] constructed a compiler that takes any semi-honest protocol that uses a source of correlated randomness, and transforms it into a protocol with security against malicious parties and with identifiable abort (in the correlated randomness model). The compiler can

[1] This is called *selective abort*, in contrast to *unanimous abort*, where the honest parties must all agree that the protocol aborted.

be seen as an information-theoretic version of the GMW compiler [GMW87a]: each party commits to its input and randomness that they intend to use for the semi-honest protocol and then runs the semi-honest protocol by broadcasting their messages in each round and using zero-knowledge to prove that their messages are correct. To generate the correlated randomness needed for this protocol, [IOZ14] also described a compiler that transforms any cryptographic preprocessing phase that is secure-with-abort into one that has identifiable abort.[2] Overall, this yields the first construction with identifiable abort that makes only black-box use of cryptographic primitives, namely an adaptively secure oblivious transfer protocol and a broadcast channel. The main downsides of this construction are the need for adaptively secure OT in the preprocessing phase, and the overall complexity of proving that each protocol step was executed correctly in the online phase.

To resolve this, multiple works [BOS16; SF16; CFY17; BOSS20] have given more "practical" constructions of ID-MPC. Baum et al [BOS16] construct an identifiable abort protocol for arithmetic circuits in the preprocessing model where the online phase is a variant of BDOZ [BDOZ11] that permits cheater identification. While avoiding adaptively secure OTs, their preprocessing phase needs to perform at least n times as much computation as non-identifiable protocols, and also relies on cheater identification for lattice-based cryptography which is far from being practically efficient. [SF16] modify the SPDZ protocol to identify cheating by ensuring that correct shares are opened. Their preprocessing would, in order to be identifiable, have to rely on the same expensive mechanisms as [BOS16] (such as verifiable decryption). Cunningham et al. [CFY17] used Pedersen commitments to identify cheaters in the online phase, which limits the finite field over which the computation can happen and makes preprocessing costly as all these commitments have to be generated during preprocessing. Finally, Baum et al.[BOSS20] construct an ID-MPC protocol for boolean circuits which runs in a constant number of rounds and uses cryptographic primitives in a black box away. While in their work, public key operations after the setup phase and zero knowledge (ZK) machinery (as well as adaptive OTs) are avoided, their construction is limited to the binary setting and their use of multiparty BMR [BMR90] has a substantial overhead from reconstructing a large garbled circuit.

Challenge of adaptive security and identifiable abort.

When considering solely a preprocessing protocol, the [IOZ14] compiler offers a simple and attractive approach to obtaining identifiable abort. At a high level, their idea is to have every party first commit to a random tape, and then run a standard, secure-with-abort protocol; if the protocol aborts, every party will open the commitments to their random tapes. This allows all other parties to detect which party cheated by re-running a local copy of the protocol. Furthermore, intuitively, opening random tapes in case of abort does not pose a privacy issue, since the preprocessing phase is independent of all parties' inputs. What is needed for this to work is that any deviation from the honest protocol

[2] This result bypasses the impossibility of [IOS12] by relying on black-box use of an *OT protocol* rather than an ideal OT functionality.

can be consistently detected by every party using the randomness that they committed to (called $\mathcal{P}$-*verifiability* in [IOZ14]).

The challenge with this approach lies in simulating the view of the corrupted parties. If the protocol aborts, the simulator needs to be able to open the honest parties' random tapes to the adversary, in a way that is consistent with the previous (simulated) transcript. One way to do this is if the preprocessing protocol is adaptively secure, so that honest random tapes can be 'explained' by the simulator as if that party had just been adaptively corrupted. This is where the reliance of [IOZ14] on adaptive OT comes from, and it seems inherent to this commit-and-open paradigm[3]. While some works have attempted to circumvent the adaptivity problem [BDD20; BOSS20], no efficient UC-secure solution for ID-MPC over arbitrary finite fields is known.

6.1.1. Our Contribution

In this work, we construct an efficient MPC protocol with identifiable abort for arithmetic circuits over large fields, with UC security [Can01]. A key feature of our protocol is an online phase based on simple, pairwise information-theoretic MACs, just as in the (secure-with-abort) BDOZ protocol [BDOZ11]. Thanks to this simple online phase, the correlated randomness that must be produced by the preprocessing phase is just standard, authenticated multiplication triples, the same as in secure-with-abort protocols. To allow identifiable abort in the online phase, our main tool is a new compiler that transforms certain classes of *sender-receiver protocols*, where one party has private input, into ones that support cheater identification. Our compiler overcomes some limitations of the related compiler from [IOZ14], which only works for preprocessing protocols, and also requires adaptive security of the original protocol.

6.1.2. Technical Overview

Online Phase.

A natural approach to achieving identifiable abort in MPC is to use a form of linear secret sharing where the parties are committed to their shares via linearly homomorphic commitments. If the commitments support *multiple receivers* and *identifiable abort*, then secret-shared values can be reliably opened, by checking commitments on the shares. Given a preprocessing phase that generates random multiplication triples, where each share is authenticated to all other parties using the homomorphic commitments, one can construct a standard MPC protocol by exploiting linearity of the commitments and using Beaver multiplication. This was done, for instance, in [BOS16], using an

[3] [BOSS20] manages to avoid the use of adaptively secure primitives by making use of a homomorphic commitment scheme and redefinitions of the offline ideal functionality. Specifically, in case of an abort of the offline phase their ideal functionality at this point did not yet output any values to the environment, so the original random tapes can safely be opened. Moreover, their preprocessing protocol uses homomorphic commitments for shares and require that all parties commit to the values they used in the preprocessing. The consistency is then ensured by opening random linear combinations of the commitments, and in the online phase these commitments can be used for cheater detection.

information-theoretic identifiable commitment scheme; however, the structure of the commitments is more complex than information-theoretic MACs used in secure-with-abort MPC [BDOZ11; DPSZ12], which led to a much more costly preprocessing protocol in [BOS16].

In this work, our online phase follows the same general approach, using preprocessed triples and identifiable linear commitments. The key differences compared with prior work are how we instantiate the preprocessing to generate multiplication triples with identifiable abort, and how we instantiate the identifiable, linearly homomorphic commitments.

Preprocessing Phase.

The goal of our preprocessing phase is to create additive secret shares of random multiplication triples over a large field, which are committed to using linearly homomorphic commitments. To do this, the parties will first run a secure-with-abort protocol, Π_{Trip}, to create unauthenticated triples (for instance, using pairwise OLE), and then commit to their shares with the homomorphic commitments. To guarantee that parties have committed to the correct shares, we then run a sacrificing-based correctness check, where one triple is sacrificed to check another, similarly to [DPSZ12].

To identify cheaters in this approach, we must make two important changes. First, following the IOZ compiler [IOZ14] and other similar approaches [BOSS20; SSS22], we have the parties commit to their random tapes of the secure-with-abort protocol Π_{Trip} before running it. If Π_{Trip} aborts, the parties then open their random tapes and reconstruct the protocol transcript to identify who cheated. This stage requires the original protocol to have a form of verifiable transcripts, meaning that it is always possible to identify who cheated, when given the (alleged) views of all parties together with their random tapes. We formalize this property, which we call *identifiable cheating*, and show that it can be cheaply added to any secure-with-abort protocol by adding digital signatures to all pairwise communication. Signatures guarantee that if some party cheats, there is a signed record of the messages it sent that can later be used to help prove this.

It remains to discuss how we can still simulate the execution of Π_{Trip}, without running into the aforementioned adaptive security issue. Following [BOSS20], we have the simulator run an honest copy of Π_{Trip}, to generate the honest parties' messages seen by the adversary. Now, it is easy to simulate the opening of random tapes in case of abort, the only problem is that the simulator no longer has any power to extract the corrupt parties' inputs. In this case, however, the only inputs that need to be extracted are the corrupted parties' shares of multiplication triples, which were already committed to via the homomorphic commitment scheme. By relying on a UC secure homomorphic commitment functionality, these shares can easily be extracted without having to use the Π_{Trip} simulator[4].

The last issue is that even if the triple protocol Π_{Trip} runs correctly, the overall protocol may still abort if the triple sacrifice fails, due to a corrupted party committing to the

[4]We still rely on the existence of the Π_{Trip} simulator in the proof, to argue that the simulated view is indistinguishable from the real protocol

wrong share. To recover from this, we again have the parties open their random tapes from Π_{Trip}, so that all parties' shares can be recovered, and then compare these with the shares that were committed to in the homomorphic commitment scheme by opening all the committed shares.

Building Identifiable, Homomorphic Commitments.

To instantiate the homomorphic commitment scheme, we design a scheme based on pairwise information-theoretic MACs, where the committed value is authenticated to every other party with a MAC, as in BDOZ. An advantage of such MACs is that they can be generated very efficiently using vector oblivious linear evaluation (VOLE) protocols based on variants of the learning parity with noise assumption, such as [BCGI18; WYKW21; BCGIKRS19]. However, the problem is that MACs do not provide a way for parties to agree upon who cheated in the event that an opening fails. Indeed, the impossibility result of [IOS12] implies that this is impossible to do with black-box use of pairwise information-theoretic MACs.

At a high level, our approach is to follow the same commit-and-open approach as for the triple generation: if an opening fails, the parties will open their random tapes for the VOLE protocols used to generate the MACs, and use the reconstructed VOLE outputs to help identify who cheated. However, we are now met with two new challenges. Firstly, the commit-and-open paradigm only works for preprocessing protocols, since all parties need to open their random tapes in case of an abort — when using homomorphic commitments in the online phase, this would leak any private, committed inputs. Secondly, we again have to deal with the adaptivity problem, which was essentially deferred in the preprocessing stage, by relying on the security of the homomorphic commitment scheme.

Compiling Sender-Receiver Protocols to Identifiable Abort.

We present a new identifiable abort compiler that works for a general class of *sender-receiver protocols*, where only one party (the sender) has private input. Like IOZ, our compiler makes black-box use of the underlying secure-with-abort protocol. Unlike IOZ, however, we are not restricted to preprocessing protocols where no party has private input — this allows us to apply our compiler to an arbitrary, linearly homomorphic commitment scheme with multiple receivers, which we instantiate with a VOLE-based protocol for setting up information-theoretic MACs.

At a high level, our compiler follows the same strategy as our preprocessing phase, except that to prevent leakage of the sender's private inputs, we only require the *receivers* to commit to and open their random tapes, and not the sender. Under a mild assumption on the communication pattern in the sender-receiver protocol, we show that the receivers will still be able to identify a cheating sender, since in this case there will always be at least one honest receiver, who can prove that they followed the protocol and aborted due to the cheating sender. There is one issue with this approach, however. Even though receivers do not have private *inputs*, they may have private *outputs* that can't be revealed. To remedy this, we use two different types of recovery mechanisms, depending on whether

a sender or receiver is claiming an abort. In the first case, the receivers will all *privately* send their evidence to the sender, who will select and publish a proof. In the second case, the aborting receiver must instead immediately open its view for all parties to inspect and confirm that it aborted; because of the restricted communication pattern of sender-receiver protocols, this would imply that the sender has cheated (and so it is not a problem to leak the receiver's output, which only depends on the corrupt sender's input).

Avoiding Adaptive Security via Online Extractability.

The final challenge in our compiler is ensuring that all of the identification stages, where honest receivers open their random tapes, can be simulated. Instead of relying on adaptive security, we observe that a weaker property suffices, which we call *online extractability*. In the UC security proof, there are two cases, depending on whether the adversary corrupts the sender, or only (a subset of) the receivers. In the first case, the main job of the simulator is to simulate messages from the honest receivers in such a way that it can extract the inputs of the corrupted sender.[5] If the simulator later has to open the honest receivers' random tapes, due to the malicious sender causing an abort, the natural approach relying on adaptive security is for the simulator to adaptively corrupt the honest receivers, so that it learns randomness that explains the previously simulated messages. Online extractability instead defines a special type of simulation, where the normal protocol execution suffices to extract adversarial inputs, if one does only imperceptible changes to the CRS or other hybrid functionalities. While this is already how many UC protocol simulators work, we define this property formally and show that it is composable. Having online extractability, the task of the simulator in our IA compiler is now much easier: it can simulate messages of the honest receivers by simply running an honest copy of the protocol, except for the interaction with a setup functionality like a CRS. This makes it trivial to open the random tape to identify a cheater, since the simulator has followed the protocol honestly. [6]

Efficiency Analysis

Efficiency compared with MPC with abort.

To investigate the overhead of obtaining identifiable abort, we compare our protocol with the preprocessing and online phases from Le Mans [RS22], which is secure with abort. There are two ways to run the preprocessing in Le Mans. The first way, called Le Mans 1 in Table 6.1, is to generate what they call "partial triples", and authenticate the triples during the online phase. Asymptotically, the preprocessing cost in this approach can have a total of $O(n^2 \log|C|)$ communication, where $|C|$ is the circuit size, when

[5] In this case, note that the simulator does not need to equivocate the corrupted receivers' outputs to match those of the ideal functionality, because of the structure of the sender-receiver protocol: these outputs only depend on the corrupted sender's input, so there is nothing to simulate.

[6] Of course, one still has to prove that a protocol is online-extractable, but this is seemingly simpler than a security proof for adaptive security. Indeed, we observe that many protocols in the literature are already online-extractable.

using pseudorandom correlation generators for OLE and VOLE correlations. The local computation of PCG approaches is still $O(n^2|C|)$, however. If instead, "non-silent" OLE or VOLE protocols are used, such as from homomorphic encryption or OT, the communication would also be $O(n^2|C|)$. The online cost is $12n$ elements per party (by using the king approach). The second version of Le Mans generate the partial triples in the preprocessing, but also authenticates and checks them, costing an additional $O(n^2|C|)$ field elements, but bringing the online cost down from $12n$ to $4n$ elements per party.

Our preprocessing has the same base cost as Le Mans 1, plus an additional $2(n-1)|C|$ field elements per party, sent via point-to-point channels. When it comes to the online phase, we use the standard BDOZ online phase with authenticated triples and signatures added to the messages, which again, increases the cost by $O(n)$. Overall, our online communication cost per party is dominated by $2(n-1)|C|$ field elements, in an honest execution.

Note that an adversary can always increase the cost of our preprocessing by forcing complaint procedures to be run (by sending invalid messages). This increases our round complexity by a factor of 2, and forces the entire transcript to go via a secure broadcast channel instead of point-to-point channels. The adversary could also cause an abort at any point during the protocol, forcing parties to open their views. However, resolving an abort in our protocol is fairly cheap in terms of computation: once the parties receive the view(s), they only need to locally compute the messages that should have been sent, with no need for expensive ZK proofs.

Efficiency compared to other ID-MPC protocols.

We now compare our construction to [BOS16] and [BOSS20].

In the preprocessing phase, [BOS16] requires $O(n^3)$ broadcast messages per multiplication gate because the parties need to perform $O(n^2)$ verifiable decryptions of RLWE ciphertexts. In our protocol, even in the worst case when all messages between parties are forced to be broadcast, we only need $O(n^2)$ broadcasts per multiplication. This asymptotic difference is due to the more complex information-theoretic signatures used in [BOS16], which take more work to set-up than our simple pairwise MACs. In the online phase, both [BOS16] and our protocol have $O(n^2)$ complexity.

Concretely, while it is hard to estimate costs without an implementation, we expect that our protocol will perform much faster than [BOS16]. Our protocol is designed to use Pseudo-random Correlation Generator (PCG) techniques for generating Oblivious Linear Evaluation (OLE) and Vector OLE correlations, and prior works estimate [BCGIKS20b] that concretely, these have orders of magnitude less communication than homomorphic encryption-based (HE) approaches. In [BOS16], the complexity of its preprocessing requirements makes it much harder to employ practical PCG techniques instead of HE, and in particular, we do not see an easy way to avoid their asymptotic $O(n^3)$ overhead.

The protocol of [BOSS20] is incomparable to ours as it is a garbled circuit-based construction that works for Boolean circuits (with a constant-round online phase). In comparison, our construction allows the evaluation of circuits over $\mathbb{F}_p$ for large p with a round complexity that depends on the circuit depth. Both their and our construction use

Protocol	Building blocks	IA	Preprocessing cost	Online cost
Le Mans, v1	(V)OLE	✗	$n^2 \times$ OLE*	$12n$
Le Mans, v2	(V)OLE	✗	$n^2 \times$ OLE*+ $O(n)$	$4n$
[BOS16]	depth-1 HE	✓	$O(n^3)^\dagger$	$O(n^2)^\ddagger$
Ours	(V)OLE	✓	$n^2 \times$ OLE*+ $O(n^2)$ ‡	$O(n^2)^\ddagger$

* Random, pairwise OLE and VOLE correlations. Can be generated with amortized $o(1)$ communication using variants of LPN [BCGIKRS19; BCGIKS20b].
† Must be broadcast
‡ Corrupted party can force to be broadcast

Table 6.1.: Comparing efficient MPC protocols with and without identifiable abort. Preprocessing cost reflects the cost per multiplication in the preprocessing phase, in terms of building blocks (OLE/VOLE) plus total communication in field elements.

homomorphic commitments during the offline phase: [BOSS20] commits each party to its GC keys, while we let each party commit to its shares. To achieve this, [BOSS20] uses a non-interactive vector commitment while we use a VOLE-based construction. Adapting our commitments to their setting might be interesting future work.

6.1.3. Related work

Interest in the area of MPC with Identifiable Abort has increased recently, leading to many exciting research directions.

Brandt et al. [BMMM20] and independently Simkin et al. [SSY22] investigated how to realize dishonest-majority MPC with identifiable abort from correlations among less than all n parties.

Cohen et al. [CGZ20] investigated the two-round MPC setting with dishonest majority and broadcast. They showed in which cases identifiable abort is achievable, depending on the broadcast use. This was extended to the honest majority setting by Damgård et al. [DMRSY21]. In follow-up work, [DRSY23] investigated which setup is necessary for the two-round setting to achieve identifiable abort. In the plain model, Ciampi et al.[CRSW22] showed how to construct ID-MPC in the optimal 4 rounds.

When considering covert instead of malicious security, Faust et al. [FHKS21] as well as Scholl et al. [SSS22] constructed compilers from passively secure MPC to covertly [AL07] secure MPC with security against $n-1$ corruptions using time-lock puzzles. Later, Attema et al. [ADEL22] showed how to realize this without time-lock puzzles, although requiring an honest majority. All these constructions actually achieve a stronger property called publicly verifiable MPC which implies identifiable abort.

Hazay et al. [HVW22] used framing-free designated-verifier Zero-Knowledge proofs to construct an alternative to the IOZ14 compiler. Their construction only works for honest majority protocols.

More concretely, Chen et al [Che+21] constructed a dedicated RSA key generation protocol with Identifiable Abort and security against a dishonest majority. The effi-

ciency of their construction comes from a communication model that uses a centralized "coordinator" which realizes broadcast.

Concurrent Work.

Recently, Cohen et al. [CDKs23] presented another approach to identifiable abort, which also manages to avoid the need for adaptively secure OT in the [IOZ14] compiler. Their method is based on revealing committed input values in case of cheating; in contrast to our approach of revealing random tapes to verify protocol messages, [CDKs23] do not make use of the underlying protocol messages in this way, instead relying on a special form of committed OT functionality.

We believe that our work has a couple of advantages over [CDKs23]:

1. We directly support MPC for computing arithmetic circuits on private inputs, instead of just correlated randomness functionalities. While [CDKs23] could be used to instantiate the correlated randomness for, say, the [IOZ14] online phase, the amount of correlated randomness needed would be at least n times more than what's used by our protocol, due to the use of $O(n)$ sized information-theoretic signatures used in [IOZ14] to authenticate the correlated randomness.

2. Even for computing correlated randomness, our protocol seems to be more efficient. [CDKs23] gives an instantiation of MASCOT-like preprocessing, with roughly a 50% overhead on top of the secure-with-abort protocol. Our protocol can be instantiated using VOLE based on SoftSpokenOT and triple generation using OT, to obtain a similar result but with essentially no extra communication cost for achieving identifiable abort.[7]

Roadmap.

In contrast to the "top-down" presentation in the technical overview in Section 6.1.2, the remainder of the paper proceeds in a "bottom-up" fashion. We start with our notion of online extractability in Section 6.3, followed by a construction of homomorphic commitments in Section 6.4, which are compiled to support identifiable abort in Section 6.5. Section 6.6 then describes our triple generation protocol, which uses the previous building blocks. In the Supplementary Material, we describe the online phase, as well as various additional technical details.

6.2. Preliminaries and Notation

We use κ as the security parameter and ρ as the statistical security parameter. Bold letters such as $\boldsymbol{a}$ are used to indicate vectors, and $\boldsymbol{a}[i]$ refers to the i-th element of the vector. We write $[a, b]$ to denote the set of natural numbers $\{a, \ldots, b\}$ and $[a, b) = \{a, \ldots, b-1\}$. We use $\boldsymbol{a} \odot \boldsymbol{b}$ to indicate the component-wise product of vectors.

[7] The main overhead incurred in our protocol is that, in the worst case, an adversary can force all all point-to-point messages to be broadcast. This broadcast is done by default in [CDKs24]

6.2.1. Modeling Security

We work in the universal composability (UC) framework [Can01] for analyzing security and assume some familiarity with this. In UC, protocols are run by interactive Turing Machines (iTMs) called *parties*. We make the simplifying assumption that any protocol π runs between a fixed set of parties, typically denoted $\mathcal{P} = \{P_1, \ldots, P_n\}$. The *adversary* $\mathcal{A}$, which is also an iTM, can actively corrupt a subset $\mathcal{P}_\mathcal{A} \subset P$ and gains control over these parties. We denote the set of honest parties by $\mathcal{P}_H = P \setminus \mathcal{P}_\mathcal{A}$. We focus on static corruptions, so these sets are fixed from the beginning. The parties can exchange messages via resources, called *ideal functionalities* (which themselves are iTMs) and which are denoted by $\mathcal{F}$. We assume that all parties can communicate via authenticated channels, and sometimes also use secure point-to-point channels and a reliable broadcast channel. These are all modelled as ideal functionalities that can be realized on top of authenticated channels using standard methods. In protocol descriptions, instead of referring to the specific functionalities, we typically write e.g. P_i *privately sends* x *to* P_j or P_i *broadcasts* x. Moreover, we work in the synchronous model, where protocols proceed in a sequence of rounds, such that every message sent in one round is guaranteed to be delivered before the start of the next round. Such synchronous communication channels can also be modelled in UC [KMTZ13].

As usual, we define security with respect to an iTM $\mathcal{Z}$ called the *environment*. The environment provides inputs to and receives outputs from the parties in $\mathcal{P}$. To define security, let $\pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ be the distribution of the output of an arbitrary $\mathcal{Z}$ when interacting with $\mathcal{A}$ in a real protocol instance π using resources $\mathcal{F}_1, \ldots$. Furthermore, let $\mathcal{S}$ denote an *ideal world adversary* and $\mathcal{F} \circ \mathcal{S}$ be the distribution of the output of $\mathcal{Z}$ when interacting with parties which run with $\mathcal{F}$ instead of π and where $\mathcal{S}$ takes care of adversarial behavior.

Definition 6.1. We say that π UC-securely implements $\mathcal{F}$ if for every iTM $\mathcal{A}$ there exists an iTM $\mathcal{S}$ (with black-box access to $\mathcal{A}$) such that for every environment $\mathcal{Z}$, the outputs of $\mathcal{Z} \circ \pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ and $\mathcal{Z} \circ \mathcal{F} \circ \mathcal{S}$ are identical except with negligible probability.

In the security experiment $\mathcal{Z}$ may arbitrarily activate parties or $\mathcal{A}$, though *only one iTM (including* $\mathcal{Z}$*) is active at each point of time.*

Definition 6.2 (Identifiable Abort)**.** Let $\mathcal{F}$ be a functionality running with a set of parties $\mathcal{P}$. We define $[\mathcal{F}]^{\mathsf{IA}}$ to be the corresponding functionality with identifiable abort, where at any time, if $\mathcal{A}$ sends a message $(\mathsf{Abort}, \mathcal{J})$ for some non-empty set $\mathcal{J} \subset \mathcal{P}_\mathcal{A}$, $[\mathcal{F}]^{\mathsf{IA}}$ sends $(\mathsf{Abort}, \mathcal{J})$ to all parties and terminates. Additionally, if $\mathcal{F}$ would at any point send a message Abort to $\mathcal{P}$, it first waits to receive a non-empty $\mathcal{J} \subset \mathcal{P}_\mathcal{A}$ from $\mathcal{A}$, and then sends $(\mathsf{Abort}, \mathcal{J})$ instead.

6.2.2. VOLE and Information-Theoretic MACs

The VOLE[8] functionality $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ (Figure 6.1) generates a batch of m random VOLE correlations between two parties P_A and P_B, usually called *sender* and *receiver*.

[8] More precisely, this is actually a so-called subfield VOLE.

Functionality $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$

Parameters: Finite field $\mathbb{F}_{p^r}$, and expansion function $\mathsf{Expand} : S \to \mathbb{F}_p^m$ with seed space S and output length m.
The functionality runs between parties P_A and P_B.

Initialize: On receiving Init from P_A and P_B, sample $\Delta^B \leftarrow \mathbb{F}_{p^r}$ for P_B, and ignore all subsequent Init commands. Store Δ^B and send it to P_B.

Extend: On receiving Extend from P_B and $(\mathsf{Extend}, \mathsf{seed})$ from P_A, where $\mathsf{seed} \in S$:

1. Compute $\boldsymbol{u} = \mathsf{Expand}(\mathsf{seed})$.
2. Sample $\boldsymbol{v} \leftarrow \mathbb{F}_{p^r}^m$ and compute $\boldsymbol{w} = \boldsymbol{u} \cdot \Delta^B + \boldsymbol{v}$.
3. Send $\boldsymbol{w}$ to P_A and $\boldsymbol{v}$ to P_B.

Corrupt Parties: If P_B is corrupt, Δ^B and $\boldsymbol{v}$ may be chosen by $\mathcal{A}$. For a corrupt P_A, $\mathcal{A}$ can choose $\boldsymbol{w}$ (and then $\boldsymbol{v}$ is recomputed accordingly).

Key Query: If P_A is corrupted, $\mathcal{A}$ may send a message $(\mathsf{guess}, \Delta')$ with $\Delta' \in \mathbb{F}_{p^r}$. If $\Delta' = \Delta$, send $\mathsf{success}$ to P_A. Else, send abort to both parties and abort.

Figure 6.1.: Functionality for Programmable VOLE

A VOLE correlation consists of two vectors $\boldsymbol{u} \in \mathbb{F}_p^m, \boldsymbol{w} \in \mathbb{F}_{p^r}^m$ held by P_A, and the element $\Delta^B \in \mathbb{F}_{p^r}$ and a vector $\boldsymbol{v} \in \mathbb{F}_{p^r}^m$ held by P_B, such that $\boldsymbol{w} = \boldsymbol{u} \cdot \Delta^B + \boldsymbol{v}$. Δ^B and $\boldsymbol{v}$ are chosen uniformly at random, while the $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ functionality allows the sender to program its share $\boldsymbol{u}$ of the correlation by providing a seed. Hence, when running two instances of $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ with different receivers P_B, P_B' but the same seed, the sender P_A ends up with the same values $\boldsymbol{u}$ as part of its VOLE correlations. The Expand function in $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ should be a pseudorandom generator, whose precise implementation depends on how the protocol is instantiated (for instance, when using LPN-based VOLE [BCGI18; WYKW21], Expand is an LPN-based PRG).

One can view a VOLE correlation as an information-theoretic MAC on the vector $\boldsymbol{u}$ of P_A. This is because: $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ does not reveal Δ^B to the sender. Assume that P_A could produce a pair of vectors $\boldsymbol{u}' \in \mathbb{F}_p^m, \boldsymbol{w}' \in \mathbb{F}_{p^r}^m$ with $\boldsymbol{u}' \neq \boldsymbol{u}$ such that the VOLE correlation holds, along with the original $\boldsymbol{u}, \boldsymbol{w}$. Therefore,

$$\boldsymbol{w} = \boldsymbol{u} \cdot \Delta^B + \boldsymbol{v}, \qquad \boldsymbol{w}' = \boldsymbol{u}' \cdot \Delta^B + \boldsymbol{v} \tag{6.1}$$

That means for the pairs, it needs to hold that $(\boldsymbol{w}[i] - \boldsymbol{w}'[i])/(\boldsymbol{u}[i] - \boldsymbol{u}'[i]) = \Delta^B$ (where the index $i \in [1, m]$ is such that $\boldsymbol{u}[i] \neq \boldsymbol{u}'[i]$). In order for this to hold, P_A needs to know the secret Δ^B if it can forge a MAC on a different value $\boldsymbol{u}'$. However, Δ^B is chosen randomly from a set of size p^r so the probability of a correct guess is negligible if p^r is exponential in the security parameter.

In $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$, P_B only obtains $\Delta^B, \boldsymbol{v}$ which are chosen uniformly at random and independent of $\boldsymbol{u}$. Therefore, P_B learns no information about $\boldsymbol{u}$ from its share of the correlation. The MAC scheme implied by VOLE is also linearly homomorphic. Details can be found e.g. in [BDSW23].

6.2.3. Signatures

In this work, we crucially rely on digital signatures and we use the standard security notion for digital signature schemes, namely, existential unforgeability under adaptive chosen-message attacks (EUF-CMA).

Definition 6.3 (Signature Scheme)**.** A signature scheme consists of the following three PPT algorithms,

$\mathsf{Gen}(1^\lambda)$**:** On input the security parameter λ, outputs a public key pk and signing key sk.

$\mathsf{Sig}(\mathsf{sk}, \mathsf{msg})$**:** On input the signing key sk and message $\mathsf{msg} \in \{0,1\}^*$ outputs a string σ.

$\mathsf{Ver}(\mathsf{pk}, \sigma, \mathsf{msg})$**:** On input a public key pk, signature σ and message $\mathsf{msg} \in \{0,1\}^*$ outputs a bit b.

We require that $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ is correct, namely that

$$\Pr_{\mathsf{msg} \in \{0,1\}^*} \left[\mathsf{Ver}(\mathsf{pk}, \sigma, \mathsf{msg}) = 1 \;\middle|\; \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, \mathsf{msg}) \end{array} \right] = 1$$

Functionality $\mathcal{F}_{\mathsf{Commit}}$

The functionality runs between a set of parties $\mathcal{P}$ and an adversary $\mathcal{A}$.

Commit: On receiving $(\mathsf{Commit}, P_i, x)$ from P_i, store (P_i, x) and send P_i to all parties.

Open: On receiving $(\mathsf{Open}, P_i, P_j)$ from P_i, retrieve x and send (x, P_i) to P_j.

Public Open: On receiving (Open, P_i) from P_i, retrieve x and send (x, P_i) to all parties.

Figure 6.2.: Functionality for a Commitment

Functionality $\mathcal{F}_{\mathsf{Rand}}$

The functionality runs between a set of parties $\mathcal{P}$ and an adversary $\mathcal{A}$. The adversary can corrupt a subset of the parties, denoted by $\mathcal{I}$.

Upon receiving a description of a domain $\mathbb{F}_{p^r}^m$ from every party in $\mathcal{P}$, uniformly sample $(x_1, \ldots, x_m) \leftarrow \mathbb{F}_{p^r}^m$ and send this to $\mathcal{A}$. If $\mathcal{A}$ responds with $\mathsf{Deliver}$, send $x_1, \ldots, x_m$ to all parties and terminate. Otherwise, if $\mathcal{A}$ sends $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J} \subset \mathcal{P}_{\mathcal{A}}$, send $(\mathsf{Abort}, \mathcal{J})$ to all parties and terminate.

Figure 6.3.: Functionality for Coin Tossing with Identifiable Abort

Definition 6.4 (EUF-CMA security)**.** Given a signature scheme $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ and security parameter λ, we say that Sig is *EUF-CMA-secure* if any PPT algorithm $\mathcal{A}$ has negligible advantage in the EUF-CMA game, defined as

$$\mathbf{Adv}_{\mathcal{A}}^{\text{EUF-CMA}} = \Pr\left[\begin{array}{c|c} \mathsf{Ver}(\mathsf{pk}, \sigma^*, \mathsf{msg}^*) = 1 & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \wedge\ \mathsf{msg}^* \notin Q & (\mathsf{msg}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sig}(\mathsf{sk},\cdot)}(\mathsf{pk}) \end{array}\right],$$

where $\mathcal{A}^{\mathsf{Sig}(\mathsf{sk},\cdot)}$ denotes $\mathcal{A}$'s access to a signing oracle with private key sk and Q denotes the set of messages msg that were queried to $\mathsf{Sig}(\mathsf{sk}, \cdot)$ by $\mathcal{A}$.

6.2.4. Basic Functionalities

We additionally use a one-to-many commitment functionality $\mathcal{F}_{\mathsf{Commit}}$, shown in Figure 6.2, as well as a coin-tossing functionality, $\mathcal{F}_{\mathsf{Rand}}$, in Figure 6.3.

These functionalities should have identifiable abort. UC commitments (with identifiable abort) can be easily realized with a random oracle or CRS and a broadcast channel, while coin-tossing can be realized using $\mathcal{F}_{\mathsf{Commit}}$ with a standard commit-and-open approach.

6.3. Online-Extractable Protocols

We now define a new subclass of UC-secure protocols, which we call *online-extractable*. For such a protocol π we define a special experiment called the *extractor execution*, that runs with a PPT iTM called the *extractor*, $\mathcal{E}$, which must extract the inputs of the adversary during a real execution of the protocol. The extractor is allowed to manipulate any Common output functionalities (see Definition 6.5) used in π, or observe any random oracle queries, as well as see all communication between the adversary and any hybrid functionalities or honest parties. Otherwise, the protocol π is run as in the *real world experiment*. This manipulation done by $\mathcal{E}$ should not be noticeable to the environment, while the inputs extracted by $\mathcal{E}$ should be indistinguishable from those that the simulator $\mathcal{S}$ obtains in the ideal world.

The definition is inspired by many security proofs of UC protocols, where the simulator $\mathcal{S}$ in the ideal setting simulates by running the actual protocol π with dummy inputs for honest parties. At the same time, $\mathcal{S}$ can extract the actual inputs of the dishonest parties that are controlled by $\mathcal{A}$ without actually deviating from the protocol[9]. This means that many protocols have this extractability property already, and constructing $\mathcal{E}$ for them will be simple by manipulating $\mathcal{S}$ (removing equivocation etc). We will later see that such $\mathcal{E}$ comes in handy when simulating protocols that have identifiable abort without relying on adaptive security.

Defining Online Extractability.

Towards formalizing online extractability, let π be a protocol that UC-implements a functionality $\mathcal{F}$, possibly using some other hybrid functionalities. For simplicity, we assume that parties in π either communicate directly or access hybrid functionalities. We call certain hybrid functionalities "Common output functionalities" if their input/output behavior towards parties is independent of which party called it:

Definition 6.5 (Common Output Functionality)**.** A functionality $\mathcal{F}$ is a *Common output functionality* if, on receiving an input (sid, x) from party P_i, the functionality will give the same response that it would also give (i) upon later query (sid, x) by P_i; and (ii) upon query (sid, x) by any other party P_j at any point.

Some examples of *Common output functionalities* are the standard CRS functionality $\mathcal{F}_{\mathsf{CRS}}$ (where $x = \perp$) or a random oracle. Definition 6.5 essentially rules out that $\mathcal{F}$ has an updatable memory that would change query results over time, unless the update is provably undetectable to protocol parties.

Let us denote by $\hat{\mathcal{F}}$ a version of the ideal functionality $\mathcal{F}$ which immediately outputs any inputs from $\mathcal{A}$ to $\mathcal{F}$ onto a special tape. We call this special tape the *ideal input tape*. It can neither be seen by parties nor $\mathcal{A}$ or the environment in any security experiment.

We also denote by $[\pi]_{\mathcal{E}}$ the *extractor execution* of protocol π, which is a modified execution of a protocol π with the extractor $\mathcal{E}$, where:

[9] A similar idea, although in the context of public verifiability, was used previously, e.g. in [BDD20].

1. $\mathcal{E}$ is allowed to observe the inputs sent to any Common output functionality and freely program the output which is given in response to any input;

2. Every message sent between two parties P_i, P_j, where P_i is honest and P_j corrupt, is first given to $\mathcal{E}$ and then forwarded to the receiver;

3. Every (non-common-output) hybrid functionality $\mathcal{F}_H$ in π is modified to $\hat{\mathcal{F}}_H$, with an ideal input tape only accessible to $\mathcal{E}$, on which the inputs from $\mathcal{A}$ to $\mathcal{F}_H$ are placed.

4. $\mathcal{E}$ continuously outputs certain values, whenever they are available, on a special tape of its own, called the *extractor tape*.

As with the ideal input tape of $\hat{\mathcal{F}}$, we assume that in the extractor execution $[\pi]_{\mathcal{E}}$, the extractor tape cannot be accessed by any party, functionality, adversary or environment unless mentioned otherwise. The only difference between $[\pi]_{\mathcal{E}}$ and π that *may* be noticeable to the environment is the change to the Common output functionalities.

We now formally define online extractability.

Definition 6.6. Let π be a protocol that UC-securely implements an ideal functionality $\mathcal{F}$ with a fixed set of parties $\mathcal{P}$ and corrupted parties $\mathcal{P}_{\mathcal{A}} \subset \mathcal{P}$. Then, we say that π is *online-extractable for corrupt* $\mathcal{P}_{\mathcal{A}}$ if there exists a PPT iTM $\mathcal{E}$, and a UC simulator $\mathcal{S}$ for π, such that, for all environments $\mathcal{Z}$ and adversaries $\mathcal{A}$ who statically corrupt the parties in $\mathcal{P}_{\mathcal{A}}$:

1. $\mathcal{Z}$ cannot distinguish $\pi \circ \mathcal{A}$ from $[\pi]_{\mathcal{E}} \circ \mathcal{A}$ (where the extractor tape of $\mathcal{E}$ is not available to $\mathcal{Z}$ or $\mathcal{A}$).

2. The distribution of the ideal input tape of $\hat{\mathcal{F}}$ in $\hat{\mathcal{F}} \circ \mathcal{S}$ is indistinguishable from that of the extractor tape of $\mathcal{E}$ in $[\pi]_{\mathcal{E}} \circ \mathcal{A}$.

UC Security vs. Online Extractability. The motivation for Definition 6.6 is that many existing UC-secure protocols can easily have such an online extractor $\mathcal{E}$. On the other hand, we highlight that this does not imply that the extractor $\mathcal{E}$ is a good UC simulator for π. Indeed, while $\mathcal{E}$ is required to successfully extract the corrupted parties' inputs, it does not (and cannot) equivocate by simulating protocol messages to be consistent with outputs of an ideal functionality. Later, when we use the definition, we will restrict ourselves to a class of protocols where such equivocation is not needed.

We also observe that online extractability is *not implied* by UC security. Consider a UC-secure protocol π that uses a trapdoor in the CRS for extraction, such as the oblivious transfer protocol of [PVW08]. Moreover, let $\mathcal{F}_{\mathsf{MPC}}$ be a general MPC functionality. We construct a protocol π' as follows:

1. Generate a CRS crs' for π, using $\mathcal{F}_{\mathsf{MPC}}$, by running a sampling algorithm for the CRS distribution inside $\mathcal{F}_{\mathsf{MPC}}$, seeded using secret randomness.

2. After crs' is output by $\mathcal{F}_{\mathsf{MPC}}$, run π using crs'.

Assuming that $\mathcal{F}_{\mathsf{MPC}}$ is UC-secure, π' is also UC-secure. To construct a simulator for π', one uses the simulator of π to generate a CRS crs that can be used for equivocation. Then, the simulator for π' programs $\mathcal{F}_{\mathsf{MPC}}$'s output to be crs and otherwise runs the simulator of π as before. Indistinguishability of the new simulator follows because of the indistinguishability of the simulator of π, as crs must be indistinguishable from crs' by assumption.

At the same time, π' is clearly not online-extractable because no $\mathcal{E}$ can change the outputs of $\mathcal{F}_{\mathsf{MPC}}$, which would be necessary in order to extract inputs of the adversary as in π.

Remark 1. Clearly, if $\mathcal{F}_{\mathsf{MPC}}$ was replaced with a normal CRS functionality then its output could be programmed. Hence, this makes the above counter-example somewhat contrived, and it might be that all "natural" UC protocols are indeed online-extractable. Unfortunately, a broader definition which may be implied by UC security[10] seems tricky to formalize, since there are always more convoluted counter-examples that can evade Definition 6.5 or similar requirements: for example, a protocol might use $\mathcal{F}_{\mathsf{MPC}}$ both to perform some useful computation, and simultaneously to generate a CRS used in a subsequent protocol.

Composition of Online Extractability.

We now provide a lemma which shows under which conditions the online extractability property composes. For this, for protocols ρ, π and a functionality $\mathcal{F}$ we define $\rho^{\mathcal{F}\to\pi}$ to be the protocol that replaces the functionality $\mathcal{F}$ in ρ with an instance of π as usual in UC composition. We show that in such a case the composed protocol is online-extractable if ρ as well as π are online-extractable.

Lemma 6.1. *Let ρ be a protocol that UC-securely implements $\mathcal{F}_\rho$ in the $\mathcal{F}$-hybrid model and is online-extractable for a corrupt set $\mathcal{P}_\mathcal{A} \subset \mathcal{P}$. Let π be a protocol that UC-securely implements $\mathcal{F}$ and is online-extractable for corrupt $\mathcal{P}_\mathcal{A}$. Then $\rho^{\mathcal{F}\to\pi}$ is online-extractable with the same $\mathcal{F}_\rho$ and $\mathcal{P}_\mathcal{A}$.*

The computational overhead from this composition is additive for each functionality that gets replaced as we only run the new $\mathcal{E}^\pi$ parallel to π. We can therefore apply the lemma a polynomial number of times.

In this part, for notation, we write $\rho \setminus \{\mathcal{F}\}$ to mean "all parts of the protocol ρ that have neither in- nor output to $\mathcal{F}$. We similarly write $\rho \setminus \pi$, if π is a subprotocol used in ρ.

Proof of Lemma 6.1. To prove the statement, we have to construct a PPT algorithm $\mathcal{E}$ that fulfills Definition 6.6 for $\rho^{\mathcal{F}\to\pi}$. We can assume that $\mathcal{E}^\rho$ exists for the protocol ρ and $\mathcal{E}^\pi$ for π, and we use these to construct $\mathcal{E}$.

Let $[\rho^{\mathcal{F}\to\pi}]_\mathcal{E}$ be the $[\cdot]_\mathcal{E}$ transformation applied to the protocol but for a so-far unspecified $\mathcal{E}$. We define $\mathcal{E}$ as follows:

[10] We believe that this is unsurprising: while our definition captures an observation of many known UC simulators, given the nature of the UC framework it seems hard to prove that all simulators must follow this simulation strategy.

- Initially run an instance of $\mathcal{E}^\rho$ and $\mathcal{E}^\pi$. Let both manipulate the CRS-like functionalities for the respective protocols $\rho \setminus \pi$ and π.
- Any messages sent between one honest party and a dishonest party in $\rho \setminus \pi$ are forwarded to $\mathcal{E}^\rho$. If the message is sent in π then we forward it to $\mathcal{E}^\pi$.
- Any hybrid functionality in $\rho \setminus (\{\mathcal{F}\} \cup \pi)$ has its ideal input tape be connected to $\mathcal{E}^\rho$. Any wrapped hybrid functionality in π has its ideal input tape be connected to $\mathcal{E}^\pi$.
- Any output written on the extractor tape of $\mathcal{E}^\pi$ is given to $\mathcal{E}^\rho$ as if it was coming from the ideal input tape of the (non-existent) wrapped $\mathcal{F}$.
- The extractor tape of $\mathcal{E}$ will be the extractor tape of $\mathcal{E}^\rho$.

For criterion 1 of Definition 6.6, we have to show that $\rho^{\mathcal{F}\to\pi}$ and $[\rho^{\mathcal{F}\to\pi}]_{\mathcal{E}}$ are indistinguishable to any environment that does not have access to the extractor tape of $\mathcal{E}$.

The change due to wrapping functionalities and copying messages is not visible to $\mathcal{Z}$ as the extractor tapes of $\mathcal{E}^\rho, \mathcal{E}^\pi$ are not accessible to it. The only changes are due to the changes to CRS-like functionalities. By first replacing the CRS-like functionalities as done by $\mathcal{E}^\rho$ and then as done by $\mathcal{E}^\pi$ we get exactly the CRS-like functionality behavior of $\mathcal{E}$, and thereby indistinguishability by a hybrid argument.

For the second criterion, observe that by assumption, the extractor tape of $\mathcal{E}^\pi$ is indistinguishable from the ideal input tape of $\hat{\mathcal{F}}$ because π is online-extractable using $\mathcal{E}^\pi$. But this in particular means that the extractor tape of $\mathcal{E}^\rho$ must have the same distribution, both when using $\hat{\mathcal{F}}$ or the output of $\mathcal{E}^\pi$, as we'd otherwise have constructed a distinguisher for $\mathcal{E}^\pi$ (since the only interaction that $\mathcal{E}^\rho$ has with $\mathcal{E}^\pi$ is via its extractor tape). Therefore, $\mathcal{E}$ must have an extractor tape distribution indistinguishable from $\hat{\mathcal{F}}_\rho$. □

2-Message OT is Online-Extractable.

To give an example of an online-extractable protocol, we observe that any 2-message OT protocol in the CRS model, such as [PVW08], is online-extractable against a corrupted receiver. We will later build on this for showing that VOLE can be realised with online-extractable protocols. We assume a standard OT functionality $\mathcal{F}_{\mathsf{OT}}$, for instance, as in [PVW08].

Lemma 6.2. *Let* Π *be any 2-message protocol that securely realizes the* $\mathcal{F}_{\mathsf{OT}}$ *functionality in the* $\mathcal{F}_{\mathsf{CRS}}$*-hybrid model. Then,* Π *is online-extractable for a corrupted receiver.*

Proof. Recall that in 2-message OT, the receiver must always send the first message. Without loss of generality, any simulator for a corrupted receiver can then be defined in terms of the randomized algorithms:

- crsSim: On input the security parameter, it outputs a crs together with a trapdoor τ.

- Ext_A: On input crs, τ and a receiver message msg_A, it outputs an extracted input $\sigma \in \{0,1\}$.
- Sim_A: On input crs, τ, a message msg_A and the receiver's output x_σ, it outputs a simulated sender message msg_B.

The UC simulator uses crsSim to emulate $\mathcal{F}_{\mathsf{CRS}}$, and then, on receiving the adversary's message msg_A, extracts its input σ using Ext_A and the trapdoor, before receiving x_σ from the ideal functionality and simulating the sender's response using Sim_A.

We define the extractor $\mathcal{E}$, which starts by emulating $\mathcal{F}_{\mathsf{CRS}}$ using crsSim, and then uses the trapdoor τ and the intercepted msg_A from the honest receiver to extract an input σ with Ext_A, which it writes to the special extractor tape. The only difference between the two executions $[\pi]_{\mathcal{E}} \circ \mathcal{A}$ and $\pi \circ \mathcal{A}$ to any $\mathcal{Z}$ is the way the CRS is sampled; but since Π is a secure protocol, these must be indistinguishable. Furthermore, since the extractor tape is defined using Ext_A, it is distributed identically to the extracted input in the UC simulation, as required. □

Online Extractability of VOLE

We show that the VOLE protocol from Wolverine [WYKW21] can be used to realise $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ (Figure 6.1), and satisfies the online extractability property for a corrupt P_A. Although Wolverine was originally shown to realize a non-programmable VOLE functionality $\mathcal{F}_{\mathsf{VOLE}}$ (where the output of an honest P_A is sampled at random by the functionality), as observed in [RS22], it can easily be extended to be programmable. The analysis in this section is focused on the non-programmable variant, but applies equally to the programmable one.

Single-Point VOLE.

The main component of the VOLE construction is a protocol for single-point VOLE, where P_A's vector $\boldsymbol{u}$ has a single non-zero entry. This is modelled by a functionality $\mathcal{F}_{\mathsf{spVOLE}}$, which is then used to build $\mathcal{F}_{\mathsf{VOLE}}$ using the LPN assumption. The latter transformation is completely non-interactive, so clearly online extractable in the $\mathcal{F}_{\mathsf{spVOLE}}$-hybrid model. We now analyze the protocol Π_{spVOLE} that realizes $\mathcal{F}_{\mathsf{spVOLE}}$, which is significantly more complex and also uses several setup functionalities.

Setup Functionalities: $\mathcal{F}_{\mathsf{OT}}$, $\mathcal{F}_{\mathsf{EQ}}$ and $\mathcal{F}_{\mathsf{VOLE}}$.

Π_{spVOLE} uses three hybrid functionalities: oblivious transfer ($\mathcal{F}_{\mathsf{OT}}$), equality testing and a smaller $\mathcal{F}_{\mathsf{VOLE}}$ functionality (which is essentially bootstrapped to the larger, more efficient one). Using Lemma 6.2, we can replace $\mathcal{F}_{\mathsf{OT}}$ by a 2-round OT protocol in the $\mathcal{F}_{\mathsf{CRS}}$ model, and preserve online extractability, since P_A is always the OT receiver. $\mathcal{F}_{\mathsf{EQ}}$ is a weak equality test functionality, which leaks P_A's input if the two parties' inputs differ. It can be easily realized using random oracle based commitment, as described in [WYKW21]. In the resulting protocol, after receiving a commitment from P_B, P_A sends its input in the clear to P_B; this makes the protocol trivially online extractable. Finally,

the $\mathcal{F}_{\mathsf{VOLE}}$ functionality used as setup can be realised with $\mathcal{F}_{\mathsf{OT}}$ using OT extension techniques [Roy22]. After analyzing Π_{spVOLE}, we will argue that this setup VOLE protocol also satisfies online extractability.

Online Extractability of Single-Point VOLE.

In the following, we refer to the protocol and proof of Π_{spVOLE}, which realizes the functionality $\mathcal{F}_{\mathsf{spVOLE}}$, in Fig. 7 and Theorem 3 of [WYKW21].

Proposition 6.3. *The protocol Π_{spVOLE} for single-point VOLE in [WYKW21. Fig. 7] is online-extractable for a corrupt P_A.*

Proof. To show online extractability, we need to show the existence of an extractor $\mathcal{E}$, which can extract all inputs that are sent to $\mathcal{F}_{\mathsf{spVOLE}}$ in a way that is indistinguishable from the inputs sent by the simulator in the ideal world. Before defining $\mathcal{E}$, we briefly recap the simulator from the proof in [WYKW21. Theorem 3] for a corrupt P_A. Briefly, the view of P_A in this protocol consists of the following:

- Interaction with hybrid functionalities $\mathcal{F}_{\mathsf{VOLE}}$, $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{EQ}}$
- A value d sent by P_B, used to fix one of P_B's outputs to the correct value

The simulator, $\mathcal{S}$, emulates the hybrid functionalities and produces a value d, while interacting with $\mathcal{F}_{\mathsf{spVOLE}}$. These interactions happen as follows:

- The $\mathcal{F}_{\mathsf{VOLE}}$ functionality does not send any output to the adversary; the simulator simply receives P_A's inputs.
- For the $\mathcal{F}_{\mathsf{OT}}$ invocations, where P_A plays receiver, the simulator receives the choice bits $\bar{\alpha}_i$ and responds by sending random values $K^i_{\bar{\alpha}_i}$ to the adversary (in the protocol, these are instead pseudorandom, derived using a GGM tree).
- The simulated d is sampled uniformly at random.
- In $\mathcal{F}_{\mathsf{EQ}}$, $\mathcal{S}$ receives a value V_A from the adversary, and compares this with V'_A, which is computed based on previously extracted values known to $\mathcal{S}$. If they differ, it then defines a key guess Δ' that is sent to $\mathcal{F}_{\mathsf{spVOLE}}$. If the guess is incorrect, then $\mathcal{F}_{\mathsf{spVOLE}}$ aborts and $\mathcal{S}$ also aborts. Otherwise, $\mathcal{S}$ sends a response to the corrupt P_A and sends P_A's extracted input to $\mathcal{F}_{\mathsf{spVOLE}}$.

In [WYKW21], it is argued that when using $\mathcal{S}$, the ideal execution is computationally indistinguishable from the real execution.

We now define the extractor $\mathcal{E}$. Recall that $\mathcal{E}$ may observe all the inputs to the hybrid functionalities $\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{OT}}, \mathcal{F}_{\mathsf{EQ}}$, as well as all communication. Since there are no CRS-like functionalities in use, the only task of $\mathcal{E}$ is to write to its extractor tape the relevant inputs to $\mathcal{F}_{\mathsf{spVOLE}}$. By observing the inputs to the hybrid functionalities, $\mathcal{E}$ has all the same information that $\mathcal{S}$ uses to define P_A's inputs. In particular, when $\mathcal{F}_{\mathsf{EQ}}$ is called by P_A, $\mathcal{E}$ can use the previously extracted values to check V_A, and if needed,

define the key guess Δ' and write this to the tape. Then, if the protocol did not abort, $\mathcal{E}$ writes P_A's extracted input, computed the same way as $\mathcal{S}$, to its tape. Since the values $\mathcal{E}$ writes to its extractor tape can be computed exactly the same way as the values sent by $\mathcal{S}$ to $\mathcal{F}_{\mathsf{spVOLE}}$, it follows that Π_{spVOLE} is online-extractable. □

Online Extractability of the VOLE Setup.

Since Π_{spVOLE} uses a smaller VOLE functionality as setup, we need to show that this can also be implemented with online extractability. We consider the main VOLE protocol from [Roy22], which is in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model. In our case, the party P_A translates to the VOLE sender in [Roy22]. Most of the challenges in that security proof come from extracting the sender's input when it is corrupt; simulating the view of the adversary is actually trivial, and done exactly as in the protocol. Online extractability is therefore straightforward, as the extractor can simply run the simulator to obtain the extracted inputs.

6.4. Homomorphic Commitments Based on VOLE

In this section, we first define a functionality for homomorphic commitment, $\mathcal{F}_{\mathsf{HCom}}$, which will be used as a building block in our preprocessing phase. We then show how to efficiently instantiate this with a sender-receiver protocol based on VOLE, giving security with abort. Using the compiler of Section 6.5, this can be directly upgraded to achieve identifiable abort.

The functionality, shown in Figure 6.4, allows a sender to input values that will be committed, as well as have random committed values sampled by the functionality. $\mathcal{F}_{\mathsf{HCom}}$ allows linear operations to be performed on the commitments, and for values to be opened privately to any one receiver, as well as publicly to all receivers. Note that $\mathcal{F}_{\mathsf{HCom}}$ only supports selective abort, and not unanimous abort. However, our compiler to identifiable abort only requires a protocol with selective abort.

Information-theoretic MACs.

We now introduce the notation for information-theoretic MACs as we use it in the protocol, building on Section 6.2.2. In the protocol, the sender P_S will be committed to values in $\mathbb{F}_p$ by holding a MAC on $x \in \mathbb{F}_p$ for each receiver, under keys known only to the receivers. The linear MAC with a receiver P_i is defined as,

$$M_i^S[\![x]\!] = x \cdot \Delta^i + K_S^i[\![x]\!]$$

where $\Delta^i \in \mathbb{F}_{p^r}$ is a long-term or global key and $K_S^i[\![x]\!] \in \mathbb{F}_{p^r}$ is a local key, used only for the MAC on x. Both keys are held by receiver P_i, while P_S holds x and $M_i^S[\![x]\!]$ (for each $i \in [n]$).[11] We occasionally write M_i^S, K_S^i when it is clear from context which value is being MACed.

[11] Note that the index in superscript denotes the party who holds a value.

Functionality $\mathcal{F}_{\mathsf{HCom}}$

Parameters: Finite field $\mathbb{F}_p$. The functionality runs between a sender P_S and a set of receiver parties $\mathcal{P}_R = \{P_1, \ldots, P_n\}$. We assume all parties have agreed upon public identifiers id_x, for each variable x used in the computation. For a vector $\boldsymbol{x} = (x_1, \ldots, x_m)$, we write $\mathsf{id}_{\boldsymbol{x}} = (\mathsf{id}_{x_1}, \ldots, \mathsf{id}_{x_m})$.

Input: On receiving $(\mathsf{Input}, \mathsf{id}_{\boldsymbol{x}}, \boldsymbol{x})$ from P_S, where $\boldsymbol{x} \in \mathbb{F}_p^l$, where l is the length of the vector, and $(\mathsf{Input}, \mathsf{id}_{\boldsymbol{x}})$ from all the other parties, store the pair $(\mathsf{id}_{\boldsymbol{x}}, \boldsymbol{x})$, and send $\mathsf{InputReceived}$ to $\mathcal{A}$.

Linear Operation: On receiving $(\mathsf{LinComb}, \mathsf{id}_{\boldsymbol{z}}, \mathsf{id}_{\boldsymbol{x}}, \mathsf{id}_{\boldsymbol{y}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$ from every P_i, compute $\boldsymbol{z} = \boldsymbol{\alpha} \odot \boldsymbol{x} + \boldsymbol{\beta} \odot \boldsymbol{y} + \boldsymbol{\gamma}$ and store $(\mathsf{id}_{\boldsymbol{z}}, \boldsymbol{z})$.

Random: On receiving $(\mathsf{Random}, \mathsf{id}_{\boldsymbol{r}}, m)$ from all parties:

1. Sample $\boldsymbol{r} \leftarrow \mathbb{F}_p^m$. If $P_S \in \mathcal{P}_{\mathcal{A}}$, instead receive $\boldsymbol{r}$ from $\mathcal{A}$.
2. Store $(\mathsf{id}_{\boldsymbol{r}}, \boldsymbol{r})$ and send $\boldsymbol{r}$ to P_S.

Private Opening: On receiving $(\mathsf{PrivOpen}, \mathsf{id}_{\boldsymbol{x}}, P_j)$ from P_S and if $(\mathsf{id}_{\boldsymbol{x}}, \boldsymbol{x})$ is stored, send $\boldsymbol{x}$ to P_j.

Output: On receiving $(\mathsf{Output}, \mathsf{id}_{\boldsymbol{z}})$ from every P_i, where $\mathsf{id}_{\boldsymbol{z}}$ has been stored previously, if $P_S \in \mathcal{P}_{\mathcal{A}}$, send Abort to the parties $\mathcal{A}$ chooses, and deliver $\boldsymbol{z}$ to the rest. If $P_S \in \mathcal{P}_H$, deliver $\boldsymbol{z}$ to all parties.

Figure 6.4.: Functionality for a Homomorphic Commitment

Protocol Π_{HCom}

Parameters: Extension field $\mathbb{F}_{p^r}$, a sender P_S and receivers $\mathcal{P}_R = \{P_1, \ldots, P_n\}$.

Initialize: Every pair of parties (P_S, P_i), for $P_i \in \mathcal{P}_R$, calls an instance of $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ with Init, so P_i receives $\Delta^i \in \mathbb{F}_{p^r}$.

Input: P_S commits to an input $x \in \mathbb{F}_p$:

1. P_S broadcasts $x - l_j$, where l_j is the next available random value, to all the parties. If no such l_j is available, run the **Random** procedure.
2. Each $P_i \in \mathcal{P}_R$, locally updates its key as $K_S^i[\![x]\!] = K_S^i[\![l_j]\!] - \Delta^i \cdot (x - l_j)$. P_S sets $M_i^S[\![x]\!] = M_i^S[\![l_j]\!]$.
3. $P_i \in \mathcal{P}_R$ sets $\langle x \rangle^i = K_S^i[\![x]\!]$ and P_S sets $\langle x \rangle^S = (x, \{M_i^S[\![x]\!]\}_{i \in [1,n]})$, for $P_i \in \mathcal{P}_R$.

Linear Operation: To compute $z = \alpha \cdot x + \beta \cdot y + \gamma$, for public $\alpha, \beta, \gamma \in \mathbb{F}_{p^r}$, where $\langle x \rangle, \langle y \rangle$ have been committed, the parties locally compute $\langle z \rangle = \alpha \cdot \langle x \rangle + \beta \cdot \langle y \rangle + \gamma$.

Random: To generate m random commitments $\langle l_1 \rangle, \ldots, \langle l_m \rangle$:

1. P_S samples a seed s.
2. Each pair of parties (P_S, P_i), for $P_i \in \mathcal{P}_R$, calls $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$, with P_S sending (Extend, s) and P_i sending Extend. P_S obtains $\{\boldsymbol{u}^S, \boldsymbol{M}_i^S\}$ and P_i receives $\boldsymbol{K}_S^i = M_i^S - \boldsymbol{u}^S \cdot \Delta^i$.
3. Each $P_i \in \mathcal{P}_R$ sets $\langle l_j \rangle^i = K_{S,j}^i$ and P_S sets $\langle l_j \rangle^S = (u_j^S, \{M_{i,j}^S\}_{i \in [1,n]})$, for $j \in [1, m+1]$.
4. The parties do the following to check the consistency of inputs to $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$:
   a) Call $\mathcal{F}_{\mathsf{Rand}}$ to get random values $\chi_1, \ldots, \chi_m \in \mathbb{F}_{p^r}$.
   b) Locally compute $\langle C \rangle = \sum_{j=1}^{m} \chi_j \cdot \langle l_j \rangle + \langle l_{m+1} \rangle$
   c) Write $\langle C \rangle^S = (C, \{M_i^S\}_{i \in [n]})$ and $\langle C \rangle^i = K_S^i$.
   d) P_S broadcasts C, and privately sends M_i^S to each P_i.
   e) Each P_i checks that $M_i^S = C \cdot \Delta^i + K_S^i$, for $i \in [1, n]$. If the check fails, abort.

Private Output: To open a value $\langle x \rangle$ to a receiver P_i, P_S privately sends x, $M_i^S[\![x]\!]$ to P_i. P_i checks that $M_i^S[\![x]\!] = \Delta^i \cdot x + K_S^i[\![x]\!]$ and aborts if it fails.

Output: To open a vector of values $\langle \boldsymbol{z} \rangle$, P_S sends $\boldsymbol{z}$, $\boldsymbol{M}_i^S[\![\boldsymbol{z}]\!]$ to P_i. Each P_i checks that $\boldsymbol{M}_i^S[\![\boldsymbol{z}]\!] = \Delta^i \cdot \boldsymbol{z} + \boldsymbol{K}_S^i[\![\boldsymbol{z}]\!]$. If the checks fail, P_i outputs abort. Otherwise, P_i outputs $\boldsymbol{z}$.

Figure 6.5.: Protocol for a Homomorphic Commitment

When x is MACed with every other receiver, we use the notation

$$\langle x \rangle = \{(x, \{M_i^S[\![x]\!]\}_{i\in[n]}), K_S^1[\![x]\!], \ldots, K_S^n[\![x]\!]\}$$

We write $\langle x \rangle^i$ to denote the parts of $\langle x \rangle$ known to P_i, that is, $\langle x \rangle^i = K_S^i[\![x]\!]$ if $i \in [n]$ and $\langle x \rangle^S = (x, \{M_i^S[\![x]\!]\}_{i\in[n]})$ for the sender P_S.

We write $\langle x \rangle + \langle y \rangle$ to denote addition of each party's respective components, which gives a valid set of MACs $\langle x + y \rangle$, thanks to the linearity of the MACs. We also write $\langle x \rangle + \gamma$ to denote adding a public constant γ to $\langle x \rangle$, which is done by having P_S add γ to x, while each receiver P_i subtracts $\gamma \cdot \Delta^i$ from $K_S^i[\![x]\!]$, giving $\langle x + \gamma \rangle$.

6.4.1. Protocol with Abort

Our protocol (Figure 6.5) is based on a similar MAC generation protocol from [RS22], with the difference that we only have a single sender instead of n senders, which allows us to simplify the protocol. MACs are set up using the VOLE functionality $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ (Figure 6.1) introduced in Section 6.2.2, which generates a batch of random MACed values between two parties. Importantly, even though the authenticated values are random, the $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ functionality allows the sender to program these by providing a seed, such that when running two instances of $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ among different receivers, it ends up committed to the same set of random values.

Consistency Check.

Since $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ does not guarantee that in each pair (P_S, P_i) for $i \in \mathcal{P}_R$, P_S inputs the same seed s, we use a consistency check in Π_{HCom}. In the check, the receivers challenge the sender to open a linear combination of all the committed values, with an extra random mask (l_{m+1}) to ensure privacy of the opened combination. We formalise the security of the check by modelling the errors introduced by a corrupt P_S as follows.

Suppose that P_S used inconsistent seeds with two receivers P_1 and P_2. Since the seeds are used to compute the $\boldsymbol{u}$ values using the Expand function, this will result in two different $\boldsymbol{u}$ values, say, $\boldsymbol{u}^1$ and $\boldsymbol{u}^2$, and hence, different commitments $\left\langle l_j^1 \right\rangle, \left\langle l_j^2 \right\rangle$. Without loss of generality, define the seed used with party P_1 to be the correct seed. In the security proof, the simulator can extract all seeds and then compute the errors $\delta_j^2 = l_j^2 - l_j^1$, for $j \in [1, m+1]$. If both P_S and the receiver party are corrupt, we set the errors to be 0. We prove that an adversary cannot pass the check with inconsistent seeds except with negligible probability via the following lemma.

Lemma 6.4. *Suppose $P_S \in \mathcal{A}$ introduces errors of the form δ_j^i with party P_i, for $j \in [m+1]$, in the Random command in Π_{HCom} (Figure 6.5). If the consistency check passes, then every pair of parties (P_S, P_i), for $i \in [1, n]$ hold a secret sharing of $l_j \cdot \Delta^i$, for $j \in [m]$. In other words, $\delta_j^i = 0$, for every i and $j \in [m]$, except with probability $1/|\mathbb{F}|$.*

Proof. Let the seed used with party P_1 be the "correct" seed, denoted by s. If a corrupted sender P_S used a different seed s^i in step 2 with a party P_i, this will result in additive errors δ_j^i in the $\langle \ell_j \rangle$ values that are committed to P_i.

In step 4d of the consistency check, $\mathcal{A}$ may send an incorrect MAC value it sends to each P_i; let us denote this by $\widetilde{M}_i^S$, and by M_i^S the actual MAC derived from the authenticated values. Then, the following relation needs to hold in order for the adversary to pass the check with party P_i,

$$\begin{aligned}\widetilde{M}_i^S &= (C + \sum_{j=1}^{m} \chi_j \cdot \delta_j^i + \delta_{m+1}^i) \cdot \Delta^i + K_S^i \\ &= M_i^S + (\sum_{j=1}^{m} \chi_j \cdot \delta_j^i + \delta_{m+1}^i) \cdot \Delta^i\end{aligned}$$

This gives $\widetilde{M}_i^S - M_i^S = (\sum_{j=1}^{m} \chi_j \cdot \delta_j^i + \delta_{m+1}^i) \cdot \Delta^i$. Since the χ_j values are sampled after $\mathcal{A}$ picks the errors δ_j^i, and at least one value of δ_j^i, for $j \in [m]$, is non-zero, and $\mathcal{A}$ does not know Δ^i for the honest parties, then the value on the right of the equation is uniformly random to $\mathcal{A}$. Therefore, the probability of $\mathcal{A}$ passing the check is at most $1/|\mathbb{F}|$. □

Theorem 6.5. *Protocol* Π_{HCom} *UC-securely realises the functionality* $\mathcal{F}_{\mathsf{HCom}}$ *assuming a broadcast channel in the presence of a malicious adversary that can statically corrupt up to* $n-1$ *parties, in the* $\left(\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}, \mathcal{F}_{\mathsf{Rand}}\right)$*-hybrid model.*

Proof. We have two cases of corruption. One is when the adversary corrupts the sender and some of the receivers and the other is when the adversary corrupts a set of only receivers.

For both cases, we construct a PPT Simulator ($\mathcal{S}$) that runs the adversary ($\mathcal{A}$) as a subroutine, and is given access to $\mathcal{F}_{\mathsf{HCom}}$. It internally emulates the functionalities $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}, \mathcal{F}_{\mathsf{Rand}}$ and we implicitly assume that it passes all communication between $\mathcal{A}$ and the environment ($\mathcal{Z}$).

The parties controlled by the $\mathcal{A}$ are indicated by $\mathcal{P}_{\mathcal{A}}$ and the honest parties by $\mathcal{P}_{\mathcal{H}}$. The sender is denoted by P_S and receiver parties are denoted by $\mathcal{P}_R$. The $\mathcal{S}$ also keeps track of a flag that is set to 0 initially, and set to 1 if the $\mathcal{A}$ cheats in any of the steps.

Adversary corrupts a subset of receivers and P_S (Case 1).

The simulation proceeds as follows:

Initialize: $\mathcal{S}$ receives Init and emulates $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$. It receives Δ^i from P_i, where $P_i \in \mathcal{P}_{\mathcal{A}}$ and $P_i \in \mathcal{P}_R$ and stores it.

Input: $\mathcal{S}$ receives a message d from the adversary. $\mathcal{S}$ uses the next unused l_j that was generated in Random and sends $(\mathsf{Input}, \mathsf{id}_x, x)$, where $x = d + l_j$, to $\mathcal{F}_{\mathsf{HCom}}$, and stores (id_x, x).

Linear Operation: These are local operations. $\mathcal{S}$ computes $z = \alpha \cdot x + \beta \cdot y + \gamma$ (also the corresponding MAC), picks a new id id_z, stores $(z, \{M_i^S[\![z]\!]\}_{i\in\mathcal{P}_H})$, and sends $(\mathsf{LinComb}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y, \alpha, \beta, \gamma)$ to $\mathcal{F}_{\mathsf{HCom}}$.

Random:

1. $\mathcal{S}$ receives (Extend, s^i) from P_S for $i \in \mathcal{P}_H$. $\mathcal{S}$ also receives $\boldsymbol{M}_i^S$ from a corrupt P_S, for all $i \in \mathcal{P}_H$, and stores them. If P_S sends inconsistent seeds, $\mathcal{S}$ sets $\mathsf{flag} = 1$. We do not simulate the case when both the sender and receiver are corrupt.
2. If $\mathsf{flag} = 0$, $\mathcal{S}$ sets $\boldsymbol{u}^S = \mathsf{Expand}(s)$ and stores $\langle l_j \rangle = \{u_j^S, w_j^S\}$ as P_S's shares. If $\mathsf{flag} = 1$, $\mathcal{S}$ arbitrarily chooses one of the seeds received and computes $\langle l_j \rangle$ with it.
3. $\mathcal{S}$ samples $\chi_1, \ldots, \chi_n \in \mathbb{F}_{p^r}$ and sends them to $\mathcal{A}$ to emulate $\mathcal{F}_{\mathsf{Rand}}$.
4. $\mathcal{S}$ stores $\widetilde{C}^S$, $\left(\widetilde{M}_i^S\right)_{i\in[1,\mathcal{P}_H]}$ it receives from P_S.
5. If $\widetilde{C}^S = C^S$ and $\mathsf{flag} = 0$, send $(\mathsf{Random}, \mathsf{id}_l, l, P_S)$ to $\mathcal{F}_{\mathsf{HCom}}$, along with s, where s is the seed received in step 1.
6. If $\widetilde{C}^S = C^S$ and $\mathsf{flag} = 1$, or $\widetilde{C}^S \neq C^S$, send abort to $\mathcal{F}_{\mathsf{HCom}}$ and abort.

Private Opening: $\mathcal{S}$ receives $(z, \widetilde{M}_i^S[\![z]\!])$, where z is a previously stored value and i is the index of the party to open to. It checks if $\widetilde{M}_i^S[\![z]\!] = M_i^S[\![z]\!]$, since the simulator knows what the MAC on z is supposed to be. If the MACs are not consistent, it aborts.

Batch Opening: $\mathcal{S}$ receives $(\boldsymbol{z}, \widetilde{M}_i^S[\![\boldsymbol{z}]\!])$, where $\boldsymbol{z}$ are a set of stored values, for all $i \in \mathcal{P}_H$. It checks if $\widetilde{M}_i^S[\![\boldsymbol{z}]\!] = M_i^S[\![\boldsymbol{z}]\!]$, since the simulator knows what the MAC on $\boldsymbol{z}$ is supposed to be. If the MACs are not consistent, it aborts.

Output: $\mathcal{S}$ receives $(z, \widetilde{M}_i^S[\![z]\!])$, where z is a previously stored value, for party P_i. It checks if $\widetilde{M}_i^S[\![z]\!] = M_i^S[\![z]\!]$, since the simulator knows what the MAC on z is supposed to be. If the MACs are not consistent, it aborts.

We need to argue that an adversary $\mathcal{A}$ cannot distinguish whether it interacts with Π_{HCom} or the simulator $\mathcal{S}$ equipped with $\mathcal{F}_{\mathsf{HCom}}$. First we'll prove indistinguishability of the simulator when $P_S \in \mathcal{P}_{\mathcal{A}}$ along with a subset of the receivers.

During **Initialize**, in both worlds the adversary picks its own random values Δ^i for the corrupt receivers. In **Input**, in the real world, $\mathcal{A}$ sends a message d, which is supposed to be $x - l_j$, where l_j is unused random value generated in **Random**. In the ideal world, the adversary sends a message d and the simulator extracts the adversary's input by computing $d - l_j$ since it knows l_j, and sends it to $\mathcal{F}_{\mathsf{HCom}}$. For **Random**, in the ideal world, the $\mathcal{S}$ receives the seeds and adversary's MACs. Then the $\mathcal{S}$ decides to abort if it received inconsistent seeds. If $\mathcal{A}$ cheats by sending inconsistent seeds, $\mathcal{S}$ always aborts where as in the real world the $\mathcal{A}$ can pass the check with probability $1/p^r$, as proven in

Lemma 6.4. Therefore, **Random** is indistinguishable except with negligible probability. In the **Output phase**, the simulator always aborts if the MAC sent by the adversary is incorrect, as it knows what the correct MAC is supposed to be. In the real world, $\mathcal{A}$ can send an inconsistent MAC and still pass the check, if it manages to guess the honest parties' Δ values correctly, which happens with negligible probability. The argument is similar for the **Private Opening** and **Batch Opening** commands.

Adversary corrupts only a subset of receivers (Case 2).

The simulation proceeds as follows:

Initialize: $\mathcal{S}$ receives Init from $\mathcal{A}$ and emulates $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$. It receives Δ^i from P_i, where $P_i \in \mathcal{P}_{\mathcal{A}}$ and stores it. $\mathcal{S}$ sends Init to $\mathcal{F}_{\mathsf{HCom}}$.

Input: $\mathcal{S}$ samples d uniformly, sends it to the adversary, and sends $(\mathsf{Input}, \mathsf{id}_x)$ to $\mathcal{F}_{\mathsf{HCom}}$.

Linear Operation: These are local operations. $\mathcal{S}$ computes $K_i^S[\![z]\!] = \alpha \cdot K_i^S[\![x]\!] + \beta \cdot K_i^S[\![]\!]y$ for all $i \in \mathcal{P}_{\mathcal{A}}$. It sends $(\mathsf{LinComb}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y, \alpha, \beta)$ to $\mathcal{F}_{\mathsf{HCom}}$.

Random:

1. $\mathcal{S}$ receives Extend from P_i for $i \in \mathcal{P}_{\mathcal{A}}$ and receives $\boldsymbol{v}^i$ from $\mathcal{A}$.
2. $\mathcal{S}$ samples $\chi_1, \ldots, \chi_n \in \mathbb{F}_{p^r}$ and sends them to $\mathcal{A}$ to emulate $\mathcal{F}_{\mathsf{Rand}}$.
3. $\mathcal{S}$ picks C^S, $\left(M_i^S\right)_{i\in[1,n]}$ such that the check in step 4e passes, and sends them to $\mathcal{P}_{\mathcal{A}}$.

Private Opening: $\mathcal{S}$ sends $(\mathsf{PrivOpen}, \mathsf{id}_z, P_i)$ on behalf of $\mathcal{P}_{\mathcal{A}}$ to $\mathcal{F}_{\mathsf{HCom}}$ to privately open the value to P_i. It receives z from $\mathcal{F}_{\mathsf{HCom}}$, and picks $M_i^S[\![z]\!]$ such that check passes and sends it to P_i.

Output: $\mathcal{S}$ sends $(\mathsf{Output}, \mathsf{id}_z)$ to $\mathcal{F}_{\mathsf{HCom}}$ to receive the output z. It computes $M_i^S[\![z]\!] = K_S^i[\![z]\!] + \Delta^i \cdot z$ and then outputs $(z, M_i^S[\![z]\!])$, for all $i \in \mathcal{P}_H$.

Now we'll provide the indistinguishability argument for the case when $P_S \notin \mathcal{P}_{\mathcal{A}}$. During **Initialize**, in both worlds the adversary picks its own random values Δ^i for the corrupt receivers. In **Input**, in the real world $\mathcal{A}$ receives a random share of the senders input. In the ideal world, $\mathcal{A}$ receives a random message d. For **Random**, in the real world $\mathcal{S}$ sends C^S, $\left(M_i^S\right)$. The adversary will check whether $M_i^S = C^S \cdot \Delta^i + K_S^i[\![C]\!]$ but $\mathcal{S}$ knows Δ^i and $K_S^i[\![C]\!]$ so it can pick C^S, $\left(M_i^S\right)$ accordingly so that the check always passes. In

the **Output** (and similarly in **Private Opening** and **Batch Opening**, $\mathcal{S}$ receives the output z from $\mathcal{F}_{\mathsf{HCom}}$ and computes the appropriate MAC which sends to $\mathcal{A}$.

□

6.4.2. Online Extractibility of Π_{HCom}

Lemma 6.6. *Protocol Π_{HCom} in Figure 6.5 is online-extractable, for any adversary corrupting the sender and any subset of receivers.*

Proof. Let $\mathcal{S}$ be the simulator for Π_{HCom} given in the proof of Theorem 6.5, for the case when the sender and a subset of the receivers are corrupted. In Π_{HCom}, there is never any communication from a receiver to the sender, so the only task of $\mathcal{S}$ is to emulate the hybrid functionalities $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ and $\mathcal{F}_{\mathsf{Rand}}$ towards the adversary, while interacting with the $\mathcal{F}_{\mathsf{HCom}}$ functionality. We can therefore use $\mathcal{S}$ to define the extractor $\mathcal{E}$, running in the execution $[\pi]_{\mathcal{E}}$, as follows:

- $\mathcal{E}$ runs an internal copy of $\mathcal{S}$; since $\mathcal{E}$ receives any message sent from the corrupt sender to an honest receiver, it can forward these messages to $\mathcal{S}$, acting as the adversary.
- Whenever $\mathcal{A}$ sends a message to a hybrid functionality $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$, $\mathcal{E}$ forwards the message to $\mathcal{S}$.
- Whenever $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{HCom}}$ with some message msg, $\mathcal{E}$ outputs msg on its special extractor tape. $\mathcal{E}$ responds to $\mathcal{S}$ exactly as $\mathcal{F}_{\mathsf{HCom}}$ would; this is possible because P_S is corrupted, so $\mathcal{E}$ knows all of the committed inputs and can correctly open them as needed. If $\mathcal{F}_{\mathsf{HCom}}$ aborts, then $\mathcal{E}$ aborts.

To show that $\mathcal{E}$ is a good extractor, we first require that the executions $\pi \circ \mathcal{A}$ and $[\pi]_{\mathcal{E}} \circ \mathcal{A}$ are indistinguishable. The only difference between the two executions is that $\mathcal{E}$ is simulating the $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ instances, and also may abort in case the underlying simulator $\mathcal{S}$ aborts. However, it follows from the proof of Theorem 6.5 that these differences are negligible.

Secondly, we must show that the special extractor tape in $[\pi]_{\mathcal{E}}$ is indistinguishable from the special functionality tape of $\hat{\mathcal{F}}_{\mathsf{HCom}}$ in $\hat{\mathcal{F}}_{\mathsf{HCom}} \circ \mathcal{S}$ to any environment $\mathcal{Z}$. This is trivially true, because $\mathcal{E}$ is running $\mathcal{S}$ the same way as in an ideal execution, and the extractor tape of $\mathcal{E}$ contains exactly the messages $\mathcal{S}$ sends to $\mathcal{F}_{\mathsf{HCom}}$. □

Online Extractability of VOLE Protocol.

To use Π_{HCom} in our compiler for identifiable abort, it is not enough that Π_{HCom} is online-extractable on its own, since the compiler from Section 6.5 requires that Π_{HCom} only uses $\mathcal{F}_{\mathsf{Rand}}$ and/or $\mathcal{F}_{\mathsf{CRS}}$ as its hybrid functionalities. We show that $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ can be replaced with a VOLE protocol in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, where the sender plays the OT receiver, and this protocol is online-extractable when the sender is corrupted. Since we showed in Lemma 6.2 how to realize $\mathcal{F}_{\mathsf{OT}}$ in an online-extractable way, by applying

Functionality $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$

Parameters: Finite field $\mathbb{F}_p$. The functionality runs between a sender P_S and a set of receiver parties $\mathcal{P}_R = \{P_1, \ldots, P_n\}$. We assume all parties have agreed upon public identifiers id_x, for each variable x used in the computation. For a vector $\boldsymbol{x} = (x_1, \ldots, x_m)$, we write $\mathsf{id}_{\boldsymbol{x}} = (\mathsf{id}_{x_1}, \ldots, \mathsf{id}_{x_m})$.

Inherits **Input**, **Linear Operation**, **Random**, **Output**, and **Private Opening** from $\mathcal{F}_{\mathsf{HCom}}$.

Abort Behaviour: $\mathcal{A}$ may corrupt any subset $\mathcal{I} \subset P_S \cup \{\mathcal{P}_R\}$. At any point in the protocol, it may send $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J} \neq \emptyset$ and $\mathcal{J} \subseteq \mathcal{I}$, upon which the functionality will send $(\mathsf{Abort}, \mathcal{J})$ to all parties and aborts.

Figure 6.6.: Functionality for a Homomorphic Commitment with Identifiable Abort

composition (Lemma 6.1), this gives an online-extractable protocol for $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model.

The identifiable abort version of $\mathcal{F}_{\mathsf{HCom}}$, $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ appears in Figure 6.6.

6.5. Compiling to Identifiable Abort

In this section, we show how to compile a protocol with active security with selective abort, and online extractibility, into a protocol that achieves identifiable abort. More specifically, we handle any class of protocols that are in the CRS model and are sender-receiver protocols, where the receivers do not have any private inputs and do not have any communication between them. This is defined formally below.

Definition 6.7. Let Π be a protocol realizing a functionality $\mathcal{F}$ in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model. We say that Π is a sender-receiver protocol if (1) No receiver has private inputs and only interacts with the sender, except when communicating with $\mathcal{F}_{\mathsf{CRS}}$ or $\mathcal{F}_{\mathsf{Rand}}$; and (2) Whenever the sender P_S, with random tape ρ_S, has an input inp and sends a message to a receiver, this is done with either:

- A broadcast channel, using a function $\mathsf{NextBC}(\rho_S, \mathsf{inp}, \mathsf{state})$, which outputs an updated state and the message msg to be broadcast. The view_S may contain any outputs from $\mathcal{F}_{\mathsf{CRS}}$ or $\mathcal{F}_{\mathsf{Rand}}$, but is otherwise only used by NextBC.
- Private communication to receiver P_i, using a function $\mathsf{NextMsg}(\rho_S, P_i, \mathsf{msgs}_i, \mathsf{state})$, where msgs_i contains the set of messages previously received from P_i.

In particular, this definition implies that any messages sent from the sender to a receiver, including via the broadcast channel, cannot depend on any previous message sent from another receiver to the sender.

Subroutine $\mathsf{Complain}(P_i, P_j)$

1. P_i broadcasts $(\mathsf{Complain}, P_j)$.
2. Let (m, σ) be the last message sent from P_j to P_i in round r (if P_j was honest). P_j broadcasts (m, σ).
3. All parties check that $\mathsf{Ver}(\mathsf{pk}_j, \sigma, (r \| P_j \| P_i \| m)) = 1$. If not, or if P_j does not broadcast the message, the parties output abort_j.

Figure 6.7.: Complaint procedure for a missing message from P_j to P_i

Algorithm $\mathsf{VerifyAbort}(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, r)$

1. Using pk_i, check all $(m^r_{S,i}, \sigma^r_{S,i})$ pairs in view_i. If any check fails, output fail.[a]
2. Use view_i and ρ_i, and the round number r to reconstruct what the messages of the receiver should have been according to $\mathsf{NextMsg}_\Pi$. If the output of the receiver is abort, output good. Else, output fail.

[a] An honest receiver would not have an invalid signature in their view without having entered the $\mathsf{Complaint}$ procedure which would have either identified the cheater or returned a valid signature.

Figure 6.8.: Algorithm for verifying whether a receiver should have aborted.

It is straightforward to see that if we take Π_{HCom} (Figure 6.5), and replace $\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}$ with any secure 2-party protocol in the CRS model, we obtain a sender-receiver protocol. In **Input**, **Private Output**, **Batch Output**, and **Output**, P_S is the only party sending messages to the receivers in Π_{HCom}. In **Random**, it is clear that the messages sent from P_S to the receivers, and as input to $\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}$, only depend on P_S's random tape and the messages received from $\mathcal{F}_{\mathsf{Rand}}$. Since $\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}$ is a two-party functionality, replacing it with a secure two-party protocol ensures that the protocol messages to P_i still cannot depend on the view of any other receiver P_j.

6.5.1. The Compiler

In the protocol (Figure 6.9), the parties start by picking a public and secret key pair for a signature scheme, and broadcast the public key. We use an EUF-CMA secure signature scheme $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$. The compiler runs the original protocol Π, and in each round, the parties add signatures to every message they are supposed to send. If any signature does not verify, or a message was not received, the receiving party P_i initiates the complaint procedure in Figure 6.7, which forces the sending party to broadcast the message to all parties (or be identified as a cheater).

The main challenge is to handle the case where Π aborts, and we use different strategies

based on the party which aborts. If there was an abort in Π, the aborting party starts the Abort phase of the protocol, where the parties identify the cheater as follows. If a receiver party, say P_i, aborts, then since Π is a sender-receiver protocol, it must be the case that either P_S or P_i is a cheater, so the parties just need to establish which. We therefore have P_i broadcast its view, and open its random tape to all the parties. The rest of the parties can locally check if P_i cheated by running the $\mathsf{VerifyAbort}$ algorithm, which verifies the correctness of the messages it sent by recomputing the actual messages using the $\mathsf{NextMsg}$ function. $\mathsf{VerifyAbort}$ has the guarantee that if run with an honest P_i's view and random tape, it always outputs good, and that it is not possible to frame an honest P_i by making it output fail because that would require forging a signature. Parties abort with either abort_i or with abort_S depending on who cheated.

On the other hand, if P_S was the party that aborted, the natural approach would be to have P_S broadcast its view and random tape as in the earlier case. However, we do not want to reveal the sender's random tape. We do not want the sender broadcasting even its view with all the receiver parties, as this poses a problem for simulation. In this case, we are operating with an honest sender, and a subset of receivers that are corrupt. Because this is a sender-receiver protocol, the honest receivers may have private outputs from the sender. This means the simulator cannot forge a view for the honest sender to give to the adversary.

Instead, we have all the receivers send their views and random tapes to the sender. The sender locally runs $\mathsf{Identify}$ (Figure 6.10) on all the views and random tapes, including its own view, to identify the receiver party that cheated. If a receiver P_i does not send its view to the sender then the sender broadcasts a complaint message for P_i who is then forced to broadcast its view and its random tape. $\mathsf{Identify}$ can reconstruct what an honest receiver should have sent, based on the random tape of the receiver and the $\mathsf{NextMsg}$ function. Then it compares these messages to the messages from the sender's view, which allows it to always identify if the receiver cheated. P_S then broadcasts its view *only* with respect to the cheating party, along with that party's view and random tape. The other honest parties can locally run $\mathsf{Identify}$ on these to be convinced that P_i was the cheater. This avoids the problem of the simulator having to send the *full* view of the honest sender. A formal description of the compiler appears in Figure 6.9.

Theorem 6.7. *Let Π be a perfectly correct, sender-receiver protocol that UC-securely realises a functionality $\mathcal{F}$ with active security and dishonest majority, and supports online extractability when the sender and a subset of receivers are corrupt. Let $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a EUF-CMA secure signature scheme. Then the compiled protocol $\Pi^{\mathsf{IA}}_{\mathsf{Cmp}}$ securely realizes $\mathcal{F}$ with active security in the $\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Commit}}$-hybrid model, and achieves identifiable abort.*

Proof. We have two cases of corruption. In the first case, the sender and a subset of the receivers is corrupt and in the second case the sender is honest and only a subset of receivers is corrupt. For both cases, we construct a PPT Simulator ($\mathcal{S}$) that runs the adversary ($\mathcal{A}$) as a subroutine, and is given access to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$.

Whenever $\mathcal{A}$ communicates with $\mathcal{F}_{\mathsf{CRS}}$, $\mathcal{S}$ calls the extractor $\mathcal{E}$ which picks whichever CRS it wants and $\mathcal{S}$ forwards it to $\mathcal{A}$. The $\tilde{}$ (tilde) symbol is used to indicate the potentially inconsistent views received from $\mathcal{A}$.

Compiler $\Pi^{\mathsf{IA}}_{\mathsf{Cmp}}$

Let Π be a sender-receiver protocol that realises $\mathcal{F}$, is actively secure with selective abort and supports online extractability. Π uses a CRS. We assume that Π is a protocol with one sender P_S and a set of receivers $P_j \in [1, n]$. All the calls to functionalities inside of Π are replaced by their corresponding protocols.

1. Before any step of the protocol is executed, each P_i sends $(\mathsf{Commit}, P_i, \rho_i)$ to $\mathcal{F}_{\mathsf{Commit}}$, where ρ_i is the random tape.

2. Each party P_i also samples a $(\mathsf{pk}_i, \mathsf{sk}_i)$ pair, and broadcasts pk_i.

3. Run the Π protocol as follows. In each round r of the protocol,

   a) Let $m^r_{i,j}$ be the message that P_i should have sent to P_j, according to $\mathsf{NextMsg}_\Pi$. Note that either P_i or P_j must be P_S.
   b) P_i sends $(m^r_{i,j}, \sigma^k_{i,j})$ to P_j, where $\sigma^r_{i,j} = \mathsf{Sig}(\mathsf{sk}_i, r||P_i||P_j||m^r_{i,j})$.
   c) P_j checks $\mathsf{Ver}(\mathsf{pk}_i, \sigma^r_{i,j}, (r||P_i||P_j||m^r_{i,j})) = 1$. If not, or if P_i did not send a message at all, P_j calls $\mathsf{Complain}(P_j, P_i)$ (Figure 6.7)
   d) If any party P_i terminates with output abort then it initiates the Abort procedure.

Abort:

1. If a receiver P_i aborted in round r:

   a) P_i broadcasts $(\mathsf{abort}, \mathsf{view}_i)$ and opens ρ_i publicly using $\mathcal{F}_{\mathsf{Commit}}$.
   b) All parties run $\mathsf{VerifyAbort}\,(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, r)$ (Figure 6.8) to establish if P_i cheated.
   c) If $\mathsf{VerifyAbort}$ returns fail, the parties output abort_i, else output abort_S.

2. If the sender P_S aborted:

a) P_S broadcasts abort.

b) All receivers P_i send view_i to P_S and privately open ρ_i to P_S using $\mathcal{F}_{\mathsf{Commit}}$.

   i. If P_S does not receive the view of some P_i, it broadcasts a complaint message for P_i. P_i is forced to broadcast (view_i)[a] and publicly open ρ_i by calling $\mathcal{F}_{\mathsf{Commit}}$ with Open.
   ii. If P_i does not broadcast, then everyone outputs abort_i.

c) P_S runs $\mathsf{IA.Identify}\,(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, \mathsf{pk}_S, \mathsf{view}_{S,i})$ (Figure 6.10) for all receivers P_i to establish who cheated. $\mathsf{view}_{S,i}$ is the view of the the sender P_S that contains only the messages from one particular party P_i.

d) P_S broadcasts $\mathsf{view}_{S,i}$ for the cheating party P_i. P_i broadcasts view_i and publicly opens ρ_i using $\mathcal{F}_{\mathsf{Commit}}$.

e) All honest parties run $\mathsf{IA.Identify}\,(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, \mathsf{pk}_S, \mathsf{view}_{S,i})$ and output abort_i. If P_S never broadcast the views, then they identify P_S as the cheater.

[a] This is ok because in this case either the receiver or the sender is corrupt.

Figure 6.9.: Compiler for identifiable abort

Algorithm IA.Identify $(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, \mathsf{pk}_S, \mathsf{view}_S)$

1. Using the random tape ρ_i and view_i, first check if the ρ_i is the one that was committed to and then compute what the messages of P_i in each round should be. Let those be $m^r_{i,S}$.
2. Check if $m^r_{i,S} \neq \hat{m}^r_{i,S}$, where $\hat{m}^r_{i,S}$ are P_i's messages in view_S, for all rounds r of the protocol so far. If any of them are inconsistent, output P_i as the cheater.
3. Else if any of the signatures from view_S fail the check $\mathsf{Ver}\left(pk_i, \sigma_S, (r_S||P_S||P_i||\hat{m}^r_{i,S})\right) = 1$, output P_i as the cheater.
4. Else, verify signatures sent by P_S. If any of them are not valid, output P_S as the cheater.

Figure 6.10.: Algorithm for identifying a cheater.

The adversary $\mathcal{A}$ corrupts a subset of receivers and P_S (Case 1).

The simulation proceeds as follows.

1. $\mathcal{S}$ receives the random tapes that $\mathcal{A}$ picked.
2. $\mathcal{S}$ samples $(\mathsf{pk}_\mathcal{H}, \mathsf{sk}_\mathcal{H})$ for the honest receivers, broadcasts $\mathsf{pk}_\mathcal{H}$, and receives the corresponding $\mathsf{pk}_\mathcal{A}$ from $\mathcal{A}$.
3. $\mathcal{S}$ picks random tapes for the honest parties and runs the Π protocol honestly. During the execution of Π, $\mathcal{S}$ sees all messages between the adversary and the honest parties and forwards them to $\mathcal{E}$ who outputs $\mathcal{A}$'s inputs on its extractor tape. As $\mathcal{E}$ writes on its extractor tape, $\mathcal{S}$ forwards $\mathcal{A}$'s inputs to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ for all the relevant commands. Additionally:
   - $\mathcal{S}$ adds a signature to every message it sends, as in the compiled protocol.
   - When $\mathcal{S}$ receives $(m^r_{S,j}, \sigma^r_{S,j})$ it verifies the signature and if the check fails, it broadcasts $(\mathsf{Complain}, P_S)$. If P_S fails to broadcast a valid signature during $\mathsf{Complain}$, $\mathcal{S}$ sends (Abort, P_S) to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ and aborts.

Abort. Depending on which party aborted in the execution of Π, $\mathcal{S}$ takes care of each case as follows.

1. Abort by an honest receiver P_i:
   a) $\mathcal{S}$ broadcasts $(\mathsf{abort}, \mathsf{view}_i, \rho_i)$ for the honest receiver P_i who aborted.
   b) $\mathcal{S}$ sends (Abort, P_S) to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ and aborts with output abort_S.
2. Abort by corrupt receiver P_j:

a) $\mathcal{S}$ receives $(\mathsf{abort}, \mathsf{view}_i, \rho_i)$ from $\mathcal{A}$ for some P_i.

b) $\mathcal{S}$ will run $\mathsf{VerifyAbort}$ $(\mathsf{pk}_i, \mathsf{view}_i, \rho_i)$ to establish if P_i aborted.

c) If P_i indeed aborted, $\mathcal{S}$ will send (Abort, P_S) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. Else it will send (Abort, P_i) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$.

3. Abort by corrupt sender P_S:

a) $\mathcal{S}$ receives abort from $\mathcal{A}$.

b) $\mathcal{S}$ sends $(\mathsf{view}_i, \rho_i)$ to $\mathcal{A}$ for all honest P_i

c) $\mathcal{A}$ broadcasts $(\widetilde{\mathsf{view}}_{S,i}, \mathsf{view}_i, \rho_i)$ for some P_i.

d) $\mathcal{S}$ runs $\mathsf{IA.Identify}\left(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, \mathsf{pk}_S, \mathsf{view}_{S,i}\right)$. If P_i aborted then send (Abort, P_S) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. In the unusual case where P_i is also corrupt but did not abort send (Abort, P_i) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$.

We will now prove why $\mathcal{A}$ cannot distinguish if it is interacting with Π^{IA} or the simulator $\mathcal{S}$ equipped with $\mathcal{F}^{\mathsf{IA}}$.

In step 1, in both worlds $\mathcal{A}$ chooses and broadcasts its own random tapes. In step 2 in both worlds $\mathcal{A}$ receives public keys for the honest parties and broadcasts the keys that it picked. For step 3, because the honest parties have no input and their messages only depend on what the corrupt sender sends so $\mathcal{S}$ can perfectly match those messages. In step 5 if $\mathcal{A}$ has received a complaint message, it will broadcast $(m_{S,j}^r, \sigma_{S,j}^r)$ in both worlds. In step 6, $\mathcal{A}$ receives an abort message in both worlds. The abort in the real world will happen if the signature check fails. Because of the security of the signature scheme the probability of $\mathcal{A}$ fooling an honest party is negligible.

In the abort phase we have three cases:

Abort by honest receiver. In step 1(a) $\mathcal{S}$ broadcasts $(\mathsf{abort}, \mathsf{view}_i, \rho_i)$. This is identically distributed to the real world, again, because $\mathcal{S}$ ran the real protocol Π using honestly generated random tapes, and because the receivers have no input.

Abort by corrupt receiver. The simulator doesn't send anything in this case so it's straightforward to argue indistinguishability.

Abort by corrupt sender. In step 3(b) $\mathcal{A}$ receives view_i, ρ_i and the argument is the same as in step 1(a). The only way a corrupt sender can frame an honest receiver is by producing some incriminating view but that only happens with negligible probability due to the security of the signature scheme.

Finally, we also consider the outputs of $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ seen by the environment. In case of abort, we already argued above that a corrupt party will always be identified. For the non-abort outputs of $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$, we rely on the online extractability property, which guarantees that the inputs to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ sent by $\mathcal{S}$ (from the extractor tape) are indistinguishable from those in the original simulation of Π, for $\mathcal{F}_{\mathsf{HCom}}$. Therefore, the non-aborting outputs of $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ are distributed the same as those in the original simulation, and indistinguishable from the real world.

The adversary $\mathcal{A}$ corrupts only a subset of receivers (Case 2).

The simulation proceeds as follows.

1. $\mathcal{S}$ receives the random tapes that $\mathcal{A}$ picked.
2. $\mathcal{S}$ samples and broadcasts $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{H}})$ for the honest parties and receives the corresponding $\mathsf{pk}_{\mathcal{A}}$ from $\mathcal{A}$.
3. $\mathcal{S}$ runs the simulator $\mathcal{S}_{\Pi}$:
   a) Whenever $\mathcal{S}_{\Pi}$ sends messages to $\mathcal{F}_{\mathsf{HCom}}$, $\mathcal{S}$ forwards these messages to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ for all the relevant commands.
   b) When $\mathcal{S}$ receives $(m_{i,S}^r, \sigma_{i,S}^r)$ it verifies the signature and if the check fails, it broadcasts $(\mathsf{Complain}, P_i)$. If the check passes $\mathcal{S}$ sends $(m_{i,S}^r)$ to $\mathcal{S}_{\Pi}$.
4. $\mathcal{A}$ either broadcasts $(m_{i,S}^r, \sigma_{i,S}^r)$ or sends nothing.
5. $\mathcal{S}$ sends (Abort, P_i) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ and aborts.

Abort. Depending on which party aborted during the execution of $\mathcal{S}_{\Pi}$, $\mathcal{S}$ takes care of each case as follows.

1. Abort by an honest receiver P_i:
   - This will never happen since the sender is honest so we can ignore it.
2. Abort by a corrupt receiver P_j:
   a) $\mathcal{S}$ receives $(\mathsf{abort}, \mathsf{view}_i, \rho_i)$ from $\mathcal{A}$ for some P_i.
   b) $\mathcal{S}$ will run $\mathsf{VerifyAbort}$ $(\mathsf{pk}_i, \mathsf{view}_i, \rho_i)$ to establish if P_i cheated. This is not really necessary.
   c) $\mathcal{S}$ will send (Abort, P_i) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ and abort.
3. Abort by an honest sender P_S:
   a) $\mathcal{S}$ broadcasts abort.
   b) $\mathcal{A}$ sends $\widetilde{\mathsf{view}}_j$ and the opening to the random tape ρ_j for all corrupt parties P_j or sends nothing (in which case $\mathcal{S}$ launches a complaint and the views need to be broadcast). If $\mathcal{A}$ doesn't broadcast these for some party P_i, then $\mathcal{S}$ sends abort_i to the functionality.
   c) $\mathcal{S}$ runs $\mathsf{IA.Identify}\left(\mathsf{pk}_j, \mathsf{view}_j, \rho_j, \mathsf{pk}_S, \mathsf{view}_{S,j}\right)$ for all corrupt parties P_j to establish who cheated.
   d) $\mathcal{S}$ broadcasts $(\mathsf{view}_{S,j}, \mathsf{view}_j, \rho_j)$ for the particular P_j who cheated.
   e) $\mathcal{S}$ sends (Abort, P_j) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ and aborts.

We will now prove why $\mathcal{A}$ cannot distinguish if it is interacting with Π^{IA} or the simulator $\mathcal{S}$ equipped with $\mathcal{F}^{\mathsf{IA}}$.

In step 1, in both worlds $\mathcal{A}$ chooses and broadcasts its own random tapes. In step 2 in both worlds $\mathcal{A}$ receives public keys for the honest parties and broadcasts the keys that it picked. In step 3(b), we know that $\mathcal{S}_\Pi$ is a good simulator for Π so the view it simulates is indistinguishable from the real world.

In the abort phase we have three cases. In 2(c) $\mathcal{S}$ aborts with the same probability as in the real world. In step 3(d) $\mathcal{S}$ broadcasts $(\mathsf{view}_{S,j}, \mathsf{view}_j, \rho_j)$. $\mathcal{S}$ can reproduce $\mathsf{view}_{S,j}$ because it consists only of messages received from $\mathcal{A}$.

□

6.5.2. Identifiable Cheating

We now present another transformation, which does not directly yield identifiable abort, but on the other hand, is not restricted to sender-receiver protocols. Similarly to the previous compiler, we use signatures to verify point-to-point communication. This ensures that a protocol transcript is verifiable, in the sense that, in an execution where an honest party aborts, a cheater can be identified given the views of all parties, even if a corrupt party may lie about its view. We will use this transformation as part of our preprocessing protocol in Section 6.6, to ensure that the triple generation subprotocol can be verified in case of an abort.

Protocol assumptions.

We let $\mathsf{NextMsg}$ denote a component of each party's state transition function that, on input the party identifier P_i, random tape ρ_i, the view of P_i, and round r, outputs the next message m that P_i is supposed to send. We assume that the protocol Π realizes a functionality $\mathcal{F}$, and is in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model.

Definition 6.8 (Dishonest execution)**.** Consider a non-aborting execution of protocol Π between parties $P_1, \ldots, P_n$ wit h random tapes $(\rho_1, \ldots, \rho_n)$, and a set $\mathcal{P}_\mathcal{A}$ of corrupted parties. We say that the execution was dishonest with respect to $\mathcal{P}_\mathcal{A}$, if there exists at least one honest party whose view in the execution is different compared to the view of the same party in an honest execution of Π on $(\rho_1, \ldots, \rho_n)$.

The notion is defined via a game and a polynomial time algorithm $\mathsf{Identify}$. The idea of the definition is as follows, we first run the protocol Π with a set of parties $P_1, \ldots, P_n$. The protocol generates the set of views $\{\mathsf{view}_i\}_{i\in[1,n]}$. The adversary is allowed to replace up to $n-1$ views with corrupted ones. We show that the identifiable cheating compiler guarantees that, except with negligible probability, given all the views, the random tapes of the parties, and public keys of all the parties, the $\mathsf{Identify}$ algorithm successfully identifies the cheating party. Formally, the definitions are as follows:

Definition 6.9 (Identifiable cheating)**.** Let Π be an actively secure protocol that UC-securely realises $\mathcal{F}$ and in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model. Let $\mathsf{Identify}$ be a deterministic polynomial-time algorithm with the syntax:

Compiler $\Pi^{\mathsf{IC}}_{\mathsf{Cmp}}$

Let Π be a maliciously secure protocol with abort that realizes $\mathcal{F}$ and is in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model. Let $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a signature scheme.

1. Each P_i, for $i \in [1, n]$, samples $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ and broadcasts pk_i.
2. The parties run Π. Let ρ_i be the random tape of each party P_i. When P_i receives a message, inp, in round r:
   a) If inp is a message from P_j of the form (m, σ), P_i checks that $\mathsf{Ver}(\mathsf{pk}_j, \sigma, (P_j \| P_i \| m \| r - 1) = 1$. If the check fails, run $\mathsf{Complain}(P_i, P_j)$.
   b) If P_i is next instructed to send a message to some party P_j:
      i. Let $(m_{i,j}, \mathsf{state}) = \mathsf{NextMsg}(P_i, \rho_i, \mathsf{view}_i, r)$.
      ii. Let $\sigma_{i,j} = \mathsf{Sig}(\mathsf{sk}_i, P_i || P_j || m_{i,j} \| r)$
      iii. Send $(m_{i,j}, \sigma_{i,j})$ to P_j

      Otherwise, P_i executes its next instruction as usual.

Figure 6.11.: Compiler for identifiable cheating

Algorithm $\mathsf{Identify}\left((\mathsf{pk}_i, \mathsf{view}_i, \rho_i)_{i \in [1,n]}\right)$

1. Emulate an execution of Π with virtual parties $P_1, \ldots, P_n$ and random tapes $\rho_1, \ldots, \rho_n$.
2. At each step where P_i sends a message $m_{i,j}$ to P_j in round r:
   a) Retrieve the next message $(\hat{m}_{i,j}, \sigma)$ from view_j.
   b) Check whether $m_{i,j} = \hat{m}_{i,j}$, where $m_{i,j} = \mathsf{NextMsg}(P_i, \rho_i, \mathsf{view}_i, r)$. If not, output P_i as a cheater. If $m_{i,j} = \hat{m}_{i,j}$, check if there was a $\mathsf{Complain}$ procedure initiated in the list of broadcasted messages. If there was one, output P_j as a cheater.
   c) Check whether $\mathsf{Ver}(\mathsf{pk}_i, \sigma, (P_i \| P_j \| \hat{m}_{i,j} \| r)) = 1$. If not, output P_j as a cheater.
3. If Π ends successfully without identifying a cheater, output $\perp$.

Figure 6.12.: Algorithm for identifying a cheater.

Experiment $\mathsf{Exp}^{\mathsf{ic}}_{\mathcal{A},\Pi}(\lambda)$

1. $\mathcal{A}$ corrupts a set of parties $\mathcal{P}_{\mathcal{A}} \subset \mathcal{P}$. Let $\mathcal{P}_H$ be the set of honest parties.
2. For each $i \in \mathcal{P}_H$, sample a random tape ρ_i.
3. For each $j \in \mathcal{P}_{\mathcal{A}}$, $\mathcal{A}$ chooses a random tape ρ_j.
4. The parties run Π, where P_i uses ρ_i as its random tape. Let view_i denote the list of messages received by P_i.
5. If all honest parties output abort_j for some $j \in \mathcal{P}_{\mathcal{A}}$, then output 0.
6. $\mathcal{A}$ receives $(\rho_i, \mathsf{view}_i)$, for $i \in \mathcal{P}_H$.
7. $\mathcal{A}$ outputs $\{\widetilde{\mathsf{view}}_j\}$ for $j \in \mathcal{P}_{\mathcal{A}}$. Redefine $\mathsf{view}_j := \widetilde{\mathsf{view}}_j$.
8. Output 1 if one of the following holds:
   - The execution of Π is dishonest with respect to $\mathcal{P}_{\mathcal{A}}$, and $\mathsf{Identify}\left((\mathsf{pk}_i, \rho_i, \mathsf{view}_i)_{i=1}^n\right) = \perp$
   - $\mathsf{Identify}\left((\mathsf{pk}_i, \rho_i, \mathsf{view}_i)_{i=1}^n\right) \in \{P_i\}_{i \in \mathcal{P}_H}$.

   Else, output 0.

Figure 6.13.: Experiment for Identifiable Cheating

- $\mathsf{Identify}\left((\mathsf{pk}_i, \rho_i, \mathsf{view}_i)_{i\in[n]}\right)$: On input the public keys, random tapes and views of all the parties, $\mathsf{Identify}$ either outputs a corrupt party P_i or an honest execution symbol $\perp$.

A protocol Π supports identifiable cheating if for any P.P.T adversary $\mathcal{A}$ it holds that:

$$\mathsf{Pr}[\mathsf{Exp}^{\mathsf{ic}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq v(\lambda)$$

where $\lambda \in \mathbb{N}$, v is a negligible function and $\mathsf{Exp}^{\mathsf{ic}}_{\mathcal{A},\Pi}(\lambda)$ is defined as in Figure 6.13.

The idea of our compiler $\Pi^{\mathsf{IC}}_{\mathsf{Cmp}}$ that ensures identifiable cheating is to have parties add signatures to their messages. In order for parties to sign and verify messages, each party chooses a public key and a secret key for a signature scheme, and broadcasts the public key before running the compiled protocol. If a party in the protocol thinks a message or the signature it received is invalid, it broadcasts a complaint message, upon which the sender of the message must broadcast the message and the signature to all the parties. The formal protocol for the identifiable cheating compiler appears in Figure 6.11.

We prove that using this compiler with any protocol that is actively secure in the dishonest majority with abort, gives a protocol that has the identifiable cheating property.

Theorem 6.8. *Let* Π *be a protocol that UC-securely realises a functionality* $\mathcal{F}$ *with active security and dishonest majority. Let* $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *be a EUF-CMA secure signature scheme. Then the compiled protocol* Π_{Cmp} *securely realises* $\mathcal{F}$ *with active security in the CRS model and using broadcast, and achieves the identifiable cheating property.*

Proof. UC-Security: If Π has any calls to hybrid functionalities, they are replaced with the corresponding protocols in the CRS model. This is allowed according to the UC theorem, so it does not break security. The transformed protocol securely realises $\mathcal{F}$ in the CRS model. The simulator $\mathcal{S}$ for a static adversary $\mathcal{A}$ corrupting up to $n-1$ parties works as follows:

1. $\mathcal{S}$ emulates the CRS by picking it according to the simulator for Π and sends it to the adversary.
2. $\mathcal{S}$ records pk from $\mathcal{A}$, picks keys on behalf of the honest parties, and sends the public keys to $\mathcal{A}$.
3. Assume that P_i was supposed to send a message to P_j in a given round of the protocol. There can be three different kinds of communication between parties P_i, P_j:
   a) P_i is corrupt, but not P_j: $\mathcal{S}$ receives a message and the corresponding signature from $\mathcal{A}$. If the signature does not verify, $\mathcal{S}$ asks $\mathcal{A}$ to broadcast the signature. If $\mathcal{A}$ does not broadcast the signature, or if it does and the signature it broadcasted does not verify, parties abort with abort_i.
   b) P_i is honest, and P_j is corrupt: $\mathcal{S}$ runs the simulator for Π to get the message m that P_i was supposed to send in the current round. $\mathcal{S}$ signs m under the keys it picked for P_i. It sends $m, \mathsf{Sig}(m)$ to the receiver P_j and waits for a response from $\mathcal{A}$. It forwards $\mathcal{A}$'s response to the functionality $\mathcal{F}$.
   c) Both parties are corrupt: This case is trivial to simulate.
4. Whenever $\mathcal{S}$ is supposed to send a message to the functionality, it does whatever the simulator for Π does to compute the message to be sent and forwards it to the functionality.

Indistinguishability is straightforward to argue as the protocol messages of Π which the adversary sees (and the messages to $\mathcal{F}$) are identically distributed as in the regular simulation. In order to distinguish between the ideal world and the real world, one must therefore break UC security of the underlying protocol. *Identifiable cheating:* Consider an adversary $\mathcal{A}$ who wins the experiment $\mathsf{Exp}^{\mathsf{ic}}_{\mathcal{A},\Pi}(\lambda)$ with some non-negligible probability. The adversary can win in one of two ways. The first is when the adversary corrupts some party and misbehaves, but $\mathsf{Identify}$ does not output any party as corrupt. The second is when the adversary manages to make $\mathsf{Identify}$ output an honest party, say P_j, as the corrupt party.

Assume it wins due to the first scenario. In this case, no parties are in conflict, meaning that none of the honest parties broadcasted a $\mathsf{complain}$ message and the $\mathsf{Identify}$ algorithm

did not identify any party as misbehaving. Let the message that the adversary incorrectly generated be $\tilde{m}^l_{i,j}$ (according to ρ_i), and $\tilde{m}^l_{i,j} \neq m^l_{i,j}$. However, $\mathsf{Identify}$ always identifies a party P_i as cheater if it produces an inconsistent message. Therefore, it is a contradiction that the adversary can misbehave with an inconsistent message and not get caught by $\mathsf{Identify}$.

The other case can only happen if $\mathcal{A}$ has forged a signature of some honest party. Assume that P_i was identified as the cheater in round l. Since round l was when the party P_i was identified, all the messages until round $l-1$ should have been consistent. This means the messages P_i sent in round l will be correct and have signatures on $(P_i||P_j||m^l_{i,j}||l)$. If $\mathcal{A}$ can instead produce a valid signature on $(P_i||P_j||\tilde{m}^l_{i,j}||l)$ with $\tilde{m}^l_{i,j} \neq m^l_{i,j}$ then a successful $\mathcal{A}$ can directly be used to construct an attacker on the EUF-CMA property of the signature scheme with a loss factor n in success probability as the reduction has to guess which simulated honest party to use to embed the challenge pk in. □

6.6. Preprocessing

In this section, we build our preprocessing protocol with identifiable abort, using the homomorphic commitments with identifiable abort from the previous section. The preprocessing protocol allows parties to secret-share random values such that the secret is known to one party only, as well as to create sharings of multiplication triples, that is, two random values together with a sharing of their product. In both cases, all shares are homomorphically committed. Parties can also apply linear operations to these sharings without interaction, or open them with identifiable abort. The preprocessing functionality, abstracting this, is formally described in Figure 6.14.

In the preprocessing protocol $\Pi^{\mathsf{IA}}_{\mathsf{Prep}}$, we will use (amongst other functionalities) n sessions of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$, where we denote by $\mathcal{F}^{\mathsf{IA},i}_{\mathsf{HCom}}$ the session where party P_i is sender and all other parties are receivers, and refer to this as P_i's session of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$. We use the notation $\langle\cdot\rangle_C$ to denote values that are additively shared and where each party P_i's share is committed using $\mathcal{F}^{\mathsf{IA},i}_{\mathsf{HCom}}$ where it is the sender. For a value $\langle x\rangle_C$, each party P_i holds $(x^i, \mathsf{id}_1, \ldots, \mathsf{id}_n)$, where id_i is the identifier where $\mathcal{F}^{\mathsf{IA},i}_{\mathsf{HCom}}$ stores the share x^i. Any linear operation performed on $\langle x\rangle$ can also be performed on the commitments, by calling $\mathsf{LinComb}$ with each $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ session. To open $\langle x\rangle_C$, each P_i calls $\mathcal{F}^{\mathsf{IA},i}_{\mathsf{HCom}}$ with $(\mathsf{Output}, \mathsf{id}_i)$, for $i \in [1, n]$, and all receivers now either receive x or $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J}$ indicates the set of cheating parties. $[x]$ denotes that $x \in \mathbb{F}_p$ is additively shared between the parties, that is, $x = x^1 + \ldots + x^n$ where P_i holds x^i.

The preprocessing protocol is described in Figure 6.16 and Figure 6.17. To generate a random input towards a party, say P_i, each P_j generates a private random value, committed using $\mathcal{F}^{\mathsf{IA},j}_{\mathsf{HCom}}$, and then privately opens this to P_i. P_i sets its random input to be the sum of all the random values it receives across the n sessions of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$. Since the parties only use $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ functionalities here, cheater identification is trivial.

To generate triples, we rely on a secure-with-abort triple generation protocol, Π_{Trip}, that securely realizes the $\mathcal{F}_{\mathsf{Triple}}$ functionality (Figure 6.15); this can be efficiently re-

Functionality $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$

Parameters: Finite field $\mathbb{F}_p$, parties $P_1, \ldots, P_n$. The adversary is allowed to corrupt a subset of parties, denoted by $\mathcal{I}$.

Random Input: On receiving $(\mathsf{RandInput}, P_i, l)$ for some $i \in [1, n]$ from all parties:

1. Sample $\boldsymbol{r} \leftarrow \mathbb{F}_p^l$.
2. Store $(\mathsf{id}_{\boldsymbol{r}}, \boldsymbol{r})$ for a fresh $\mathsf{id}_{\boldsymbol{r}}$ and send $(\mathsf{id}_{\boldsymbol{r}}, \boldsymbol{r})$ to P_i and $\mathsf{id}_{\boldsymbol{r}}$ to everyone else.

Linear Operation: On receiving $(\mathsf{LinComb}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y, \alpha, \beta, \gamma)$ from every P_i where $\mathsf{id}_x, \mathsf{id}_y$ are assigned, id_z is unassigned and where $\alpha, \beta, \gamma \in \mathbb{F}_p$, compute $z = \alpha \cdot x + \beta \cdot y + \gamma$ and store (id_z, z).

Triple Generation: On receiving $(\mathsf{TripGen}, l)$ from all parties:

1. Sample $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_p$ of length l and let $\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b}$.
2. Store $(\mathsf{id}_{\boldsymbol{a}}, \boldsymbol{a}), (\mathsf{id}_{\boldsymbol{b}}, \boldsymbol{b}), (\mathsf{id}_{\boldsymbol{c}}, \boldsymbol{c})$ for unused $\mathsf{id}_{\boldsymbol{a}}, \mathsf{id}_{\boldsymbol{b}}, \mathsf{id}_{\boldsymbol{c}}$.
3. Send $(\mathsf{id}_{\boldsymbol{a}}, \mathsf{id}_{\boldsymbol{b}}, \mathsf{id}_{\boldsymbol{c}})$ to all parties.

Output: Upon receiving $(\mathsf{Output}, \mathsf{id}_x)$ from all parties and where id_x is assigned:

1. Send x to $\mathcal{A}$.
2. If $\mathcal{A}$ sends $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J} \subseteq \mathcal{I}, \mathcal{J} \neq \emptyset$, then send $(\mathsf{Abort}, \mathcal{J})$ to all the parties and terminate.
3. If $\mathcal{A}$ sends $\mathsf{Deliver}$ then output x to all parties.

Corrupt Party Behaviour: Whenever the adversary is supposed to send a value, it can choose to not send a value at all, triggering an abort. The functionality receives $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J} \subseteq \mathcal{I}$ from $\mathcal{A}$ and $\mathcal{J} \neq \emptyset$, sends it to all the parties and terminates.

Figure 6.14.: Functionality for Preprocessing

Functionality $\mathcal{F}_{\mathsf{Triple}}$

Parameters: Finite field $\mathbb{F}_p$. Parties $P_1, \ldots, P_n$. The adversary is allowed to corrupt a subset of parties, denoted by $\mathcal{I}$. Denote the honest parties as $\mathcal{P}_H$.

Generate Triples: On receiving (Trip, l) from all the parties, sample a fresh set of l triples $(a_j, b_j, c_j) \in \mathbb{F}_p^3$ for $j \in [1, l]$. Output additive shares $[a_j], [b_j], [c_j]$ of the triple to each party.

Corrupt Parties: The adversary is allowed to choose its shares of the triples, as well as additive errors for the triples. If the errors are δ_a^i, δ_b^i for $i \in \mathcal{P}_H$ for a triple, the triple will now be computed as $c = a \cdot b + \Sigma_{i \in \mathcal{P}_H} \left(a_i \cdot \delta_b^i + b_i \cdot \delta_a^i \right)$.

Figure 6.15.: Functionality for unauthenticated triples

alized, for instance, using pairwise OLE correlations as in the preprocessing protocol of Le Mans [RS22]. We then compile Π_{Trip} to support identifiable cheating, using our compiler from Figure 6.11, to obtain a protocol $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$.

After running $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$, each party gets unauthenticated shares of a batch of triples. These triples are then authenticated, by having parties commit to their shares using the respective sessions of $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$, and then checked for correctness, using random challenges and a standard triple sacrifice protocol.

Notice that there can two types of errors when creating triples. Firstly, the triple generation protocol $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ might abort. The other kind of error is when $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ results in consistent shares of triples, but the adversary inputs inconsistent shares into $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. This would lead to the triple sacrifice failing. If there is an abort in $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$, parties open their random tapes using $\mathcal{F}_{\mathsf{Commit}}$ and broadcast their views from $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$. Since $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ has identifiable cheating, parties can now run the $\mathsf{Identify}$ algorithm locally on input $(\mathsf{pk}_i, \mathsf{view}_i, \rho_i)_{i=1}^n$ to identify a corrupt party. If there is an abort in the sacrifice check, on the other hand, in addition to running $\mathsf{Identify}$, they also need to check consistency of the inputs to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ to outputs of $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$. In order to do this, they call $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ with Output across all sessions of $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ and check that the inputs to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ match with the outputs of $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$. Here, any deviation allows to directly identify a cheater.

Theorem 6.9. *Suppose that a perfectly correct protocol Π_{Trip} UC-securely implements $\mathcal{F}_{\mathsf{Triple}}$ against an active adversary corrupting at most $n-1$ parties.*

Then, the protocol $\Pi_{\mathsf{Prep}}^{\mathsf{IA}}$ (using the compiled protocol $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$) UC-securely implements the functionality $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ in the presence of a malicious adversary that statically corrupts up to $n-1$ parties, in the $(\mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}, \mathcal{F}_{\mathsf{Commit}})$-hybrid model.

In the proof, we construct a simulator $\mathcal{S}$ which simulates honest parties and the ideal functionalities towards the adversary. For **Random Input** $\mathcal{S}$ just runs the protocol, except for a malicious receiver P_i where $\mathcal{S}$ equivocates an honest party's commitment

Protocol $\Pi^{\mathsf{IA}}_{\mathsf{Prep}}$ (Part 1)

Parameters: Finite field $\mathbb{F}_p$. Parties $P_1, \ldots, P_n$.

Initialize: Each P_i samples a random tape Rnd_i. P_i sends $(\mathsf{Commit}, P_i, \mathsf{Rnd}_i)$ while every other party receives P_i from $\mathcal{F}_{\mathsf{Commit}}$. Parties repeat this process with every P_i committing to its random tape. Parties also run the first step of the compiled $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ and each party obtains the verification keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$.

Random Input: P_i uses the next available random input sharing id_i. If it has no random inputs left, generate a batch of l as follows:

1. Each P_j calls $\mathcal{F}^{\mathsf{IA},j}_{\mathsf{HCom}}$ with $(\mathsf{Random}, \mathsf{id}_j, l)$, while the other parties call $\mathcal{F}^{\mathsf{IA},j}_{\mathsf{HCom}}$ acting as receivers. P_j receives $\boldsymbol{r}_j$ of length l.
2. Each P_j then calls each instance of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ with $(\mathsf{PrivOpen}, \mathsf{id}_\mathsf{j}, P_i)$. P_i receives $\boldsymbol{r}_j$ for $j \in [1, n]$ and sets its random input as $\boldsymbol{x}_i = \Sigma_{j=1}^{n} \boldsymbol{r}_j$. It considers the value of id_i as the first unused element of $\boldsymbol{x}_i$.

Linear Operation: To compute $z = \alpha \cdot x + \beta \cdot y + \gamma$, parties set $\langle z \rangle_C = \alpha \cdot \langle x \rangle_C + \beta \cdot \langle y \rangle_C + \gamma$.

Output: To output a value $\langle \boldsymbol{x} \rangle_C$, each party calls all instances of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ with $(\mathsf{Output}, \mathsf{id}_{\boldsymbol{x}})$.

Figure 6.16.: Protocol for preprocessing

Protocol $\Pi^{\mathsf{IA}}_{\mathsf{Prep}}$ (Part 2)

Triple Generation:

1. Parties run $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ using the random tapes Rnd_i for P_i. They receive additive shares of $2l$ triples – $[a_j], [b_j], [c_j]$, for $j \in [1, 2l]$. If any party notices an abort while running $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$, then it broadcasts Abort and all parties go to **Abort 1**.

2. Each P_i calls $\mathcal{F}^{\mathsf{IA},i}_{\mathsf{HCom}}$, where it acts as the sender, with $\Big(\mathsf{Input}, \mathsf{id}_{a\text{-}j}, \mathsf{id}_{b\text{-}j}, \mathsf{id}_{c\text{-}j}, [a_j], [b_j], [c_j]\Big)$ for $j \in [1, 2l]$. Using the identifiers, parties form $\langle a\rangle_C, \langle b\rangle_C, \langle c\rangle_C$.

3. Parties call $\mathcal{F}_{\mathsf{Rand}}$ to receive public random values $\boldsymbol{t} \in (\mathbb{F}^*_p)^l$ and a set of random combiners $\chi_1, \ldots, \chi_l \in \mathbb{F}_p$.

4. For $i = 1, \ldots, l$, the parties do the following (in parallel):

   a) For iteration i, parties select a pair of previously unused triples $(\langle a\rangle_C, \langle b\rangle_C, \langle c\rangle_C), (\langle a'\rangle_C, \langle b'\rangle_C, \langle c'\rangle_C)$.

   b) Compute $\langle \alpha\rangle_C = \langle t_i \cdot a + a'\rangle_C$ and $\langle \beta\rangle_C = \langle b + b'\rangle_C$. Open these values using the Output command of each instance of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$. If any $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ instance sends $(\mathsf{Abort}, \mathcal{J})$, parties abort with $\mathcal{J}$ being the set of cheating parties.

   c) Locally compute $\langle d_i\rangle_C = t_i \cdot \langle c\rangle_C - \langle c'\rangle_C + \alpha \cdot \langle b\rangle_C + \beta\langle a'\rangle_C - \alpha \cdot \beta$.

5. Compute $\langle \sigma\rangle_C = \sum_{i=1}^{l} \chi_i \cdot \langle d_i\rangle_C$.

6. Open σ by calling each $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ with Output. If $\sigma = 0$, accept all $\langle a\rangle_C, \langle b\rangle_C, \langle c\rangle_C$ as good triples and discard all $\langle a'\rangle_C, \langle b'\rangle_C, \langle c'\rangle_C$. If $\sigma \neq 0$, parties abort and go to **Abort 2**.

Abort 1: If there is an abort in $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ in Step 1 of the Triple Generation,

1. Each P_i opens its commitment to Rnd_i to everyone, by sending Open to $\mathcal{F}_{\mathsf{Commit}}$.

2. Each P_i broadcasts view_i from $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$. Then each party runs $\mathsf{Identify}((\mathsf{pk}_i, \mathsf{view}_i, \rho_i)_{i\in[n]})$ and output as cheater whatever the algorithm outputs.

Abort 2: If there is an abort in the triple sacrifice in Step 4, parties first run the same as in **Abort 1**, and in addition, parties call all instances of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ with Output to open their triple shares. Parties check that the inputs of each P_i to $\mathcal{F}^{\mathsf{IA},i}_{\mathsf{HCom}}$ matched the triple shares P_i obtained as outputs from $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$. If not, parties output $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J}$ is the set of parties with inconsistent inputs to that instance of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$.

Abort 3: If there is an abort in $\mathcal{F}_{\mathsf{Rand}}$ or any instance of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$, all parties abort with $(\mathsf{Abort}, \mathcal{J})$ received from the respective functionality.

Figure 6.17.: Protocol for preprocessing (Triple Generation)

in $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ to open to the output provided $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$. For **Output**, it does exactly the same. **Linear Operation** is entirely local, so simulation is trivial. For **Triple Generation** $\mathcal{S}$ runs the protocol, but will always abort if the $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ sessions contain values that are not consistent multiplication triples.

To show that $\mathcal{S}$'s output can only be distinguished from $\Pi_{\mathsf{Prep}}^{\mathsf{IA}}$ with the given probability, the main difference lies in the occurrence of aborts and the identified cheaters. The $1/(p-1)$ term comes from aborts that also happen in $\mathcal{S}$ if $d = 0$ (while the protocol never aborts). Concerning identified cheaters, we have that $\mathsf{Identify}$ identifies no or an honest party (i.e. the wrong party) with probability at most $\mathsf{negl}(\lambda)$ due to the Identifiable Cheating property of Π_{Trip}, while $\mathcal{S}$ always identifies corrupt dishonest parties.

Proof. We construct a PPT simulator $\mathcal{S}$ that runs the adversary $\mathcal{A}$ as a subroutine, and is given access to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$. It internally emulates the functionalities $\mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},1}, \ldots, \mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},n}$, $\mathcal{F}_{\mathsf{Commit}}$ and we implicitly assume that it passes all communication between $\mathcal{A}$ and the environment $\mathcal{Z}$.

Parties controlled by the adversary are indicated by $\mathcal{P}_{\mathcal{A}}$ and the honest parties are denoted by $\mathcal{P}_H$.

For simplicity, we specify behavior as if the adversary uses $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}, \mathcal{F}_{\mathsf{Commit}}$ or $\mathcal{F}_{\mathsf{Rand}}$ truthfully as specified in the protocol. If any of the functionalities abort, or a dishonest party does not send a command in a round to a functionality as it was supposed in the protocol, then the simulator will simply collect the sets of corrupted parties that are identified by the hybrid functionalities as $\mathcal{J}$ and immediately send $(\mathsf{Abort}, \mathcal{J})$ to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ as parties would do in **Abort 3** of the protocol.

Initialize: $\mathcal{A}$ chooses its random tape ρ_i for each dishonest party $P_i \in \mathcal{P}_{\mathcal{A}}$ and sends them to $\mathcal{F}_{\mathsf{Commit}}$ which is emulated by $\mathcal{S}$. For each honest party $P_i \in \mathcal{P}_H$, the simulator emulates making a commitment via $\mathcal{F}_{\mathsf{Commit}}$ to $\mathcal{A}$.

Random Input: Let P_i be the party to receive l inputs. If $P_i \in \mathcal{P}_H$ then the simulator just emulates running the protocol and sends $(\mathsf{RandInput}, P_i, l)$ in the name of each dishonest party P_j to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ whenever $\mathcal{A}$ sends $\mathsf{PrivOpen}$ to the respective $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j}$. If P_i is dishonest:

1. For every $P_j \in \mathcal{P}_{\mathcal{A}}$ that sends $(\mathsf{Random}, \mathsf{id}_{\boldsymbol{r}}, l)$ to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j}$ send $(\mathsf{RandInput}, P_i, l)$ in its name to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ and note the committed value as $\boldsymbol{r}_j$. For every honest party $P_j \in \mathcal{P}_H$, simulate committing to these values via each simulated session of $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j}$ for a randomly chosen $\boldsymbol{r}_j$.

2. Let P_{j*} be a designated honest party. Once $\mathcal{S}$ obtains $(\mathsf{id}_{\boldsymbol{r}}, \boldsymbol{r})$ from $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ the simulator emulates opening the random value $\boldsymbol{r}_j$ to P_i by $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j}$ of any honest $P_j \neq P_{j*}$ with $(\mathsf{PrivOpen}, \mathsf{id}_{\boldsymbol{r}})$. For P_{j*}, it instead lets $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j*}$ send the value $\delta = \boldsymbol{r} - \sum_{j \in [n], j \neq j*} \boldsymbol{r}_j$ to P_i.

Linear Operation: These are a local operations so they need not be simulated.

Triple Generation:

1. $\mathcal{S}$ runs $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$, which is secure with identifiable cheating, by picking dummy random tapes for each $P_i \in \mathcal{P}_H$. If an abort occurs in $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$, the simulator opens its commitment to the honest parties' random tapes and receives the opening from $\mathcal{A}$ for its tapes. Then, $\mathcal{S}$ sends $\{\mathsf{view}_i\}_{i\in\mathcal{P}_H}$ for each honest party to $\mathcal{A}$ and receives $\{\mathsf{view}_i\}_{i\in\mathcal{P}_\mathcal{A}}$ for all the parties $\mathcal{A}$ controls. It runs the $\mathsf{Identify}$ algorithm with input $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, \rho_2, \ldots, \rho_n, \mathsf{view}_1, \ldots, \mathsf{view}_n)$. If $\mathsf{Identify}$ only identifies dishonest parties $\mathcal{J}$, then $\mathcal{S}$ sends $(\mathsf{Abort}, \mathcal{J})$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ and terminates. If $\mathsf{Identify}$ outputs $\perp$ or also an honest party, then $\mathcal{S}$ sends $(\mathsf{Abort}, \mathcal{P}_\mathcal{A})$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ and terminates.

2. $\mathcal{S}$ emulates each session of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ by receiving $\mathcal{A}$'s shares of the triples it obtains from $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ and storing them, and also consistently inputting its own shares into $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ sessions consistent with the outputs it obtained from $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$.

3. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Rand}}$ by picking a random value t and random combiners $\chi_1, \ldots, \chi_l$, and sending them to $\mathcal{A}$.

4. $\mathcal{S}$ receives commands to each $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ from $\mathcal{A}$ needed to compute $\langle t \cdot a - a' \rangle_C$ and $\langle b - b' \rangle_C$ for each triple. It honestly computes the corresponding shares for the honest parties. $\mathcal{S}$ then opens its shares via Output to $\mathcal{A}$ to open $t \cdot a - a'$ and $b - b'$ and waits for $\mathcal{A}$ to open its shares.

5. $\mathcal{S}$ honestly computes $\langle \sigma \rangle_C$ for the honest parties and sends the shares to $\mathcal{A}$ via the opening of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$. It receives $\mathcal{A}$'s shares of $\langle \sigma \rangle_C$ and checks if $\sigma = 0$.

   a) If $d = 0$ and all multiplication triples are consistent, then $\mathcal{S}$ sends $(\mathsf{TripGen}, l)$ in the name of each dishonest party to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$.

   b) If $d = 0$ but there are inconsistent multiplication triples committed to, then $\mathcal{S}$ simply sends $(\mathsf{Abort}, \mathcal{P}_\mathcal{A})$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ and aborts.

   c) If $d \neq 0$, $\mathcal{S}$ broadcasts an Abort, opens the honest parties' random tape commitments as well as all triple shares it has committed to via $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$. Then, it waits to receive $\mathcal{A}$'s openings of its random tape commitments and triple share commitments via $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$. $\mathcal{S}$ then broadcasts $\{\mathsf{view}_i\}_{i\in\mathcal{P}_H}$ and waits for the $\mathcal{A}$'s views $\{\mathsf{view}_i\}_{i\in\mathcal{P}_\mathcal{A}}$. Using all the views and the random tapes, $\mathcal{S}$ can now run the $\mathsf{Identify}$ algorithm as in the protocol. As above, if $\mathsf{Identify}$ identifies dishonest parties $\mathcal{J}$, then $\mathcal{S}$ sends $(\mathsf{Abort}, \mathcal{J})$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ and terminates. If $\mathsf{Identify}$ outputs $\perp$ or also an honest party, then $\mathcal{S}$ sends $(\mathsf{Abort}, \mathcal{P}_\mathcal{A})$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ and terminates. If no cheaters are detected in $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$, $\mathcal{S}$ checks if the $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ output shares that $\mathcal{A}$ opened via $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ are consistent with the shares $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ generated. If $\mathcal{S}$ then identifies parties where these are different, then it sends $(\mathsf{Abort}, \mathcal{J})$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$, where $\mathcal{J}$ is the set of parties with inconsistent triple commitments. If no such party could be identified, then $\mathcal{S}$ sends $(\mathsf{Abort}, \mathcal{P}_\mathcal{A})$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$.

Output: On receiving $(\mathsf{Output}, \mathsf{id}_x)$ from $P_i \in \mathcal{P}_\mathcal{A}$ to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},i}$, the simulator forwards the message to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$, from which it gets the value x. $\mathcal{S}$ knows $\mathcal{A}$'s shares of the output as they are committed in $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},i}$, so it picks shares for one honest party P_{j*} (as in the input phase) such that they add up to x and makes $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j*}$ output the correct share to $\mathcal{P}_\mathcal{A}$.

Indistinguishability: We argue that no computationally bounded environment can distinguish between the real world and ideal world executions, except with probability $1/p + \mathsf{negl}(\lambda)$.

For all operations except **Triple Generation**, it is trivial to see that simulation and real protocol are perfectly indistinguishable in its abort behavior and in terms of consistency as $\mathcal{S}$ does the same as **Abort 3** and each hybrid functionality only identifies dishonest parties as cheaters. We can therefore focus on **Triple Generation**.

First we look at the part running $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ or where it may abort. In the ideal world, if we have an abort during $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ then the simulator will always abort with dishonest parties only. In the real world, the algorithm $\mathsf{Identify}$ may identify no cheater at all or even an honest party. But since $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ has the identifiable cheating property, by Definition 6.9 this can only occur with probability $\mathsf{negl}(\lambda)$.

In the ideal world, the simulation of the triple check always aborts if a committed triple is incorrect (i.e. even if $d = 0$). In the real protocol, this may not be the case. By a standard argument (see e.g. [LN17. Lemma 3.5]), the probability of this event happening to allow distinguishability is $1/(p-1)$.

Next, consider the case where $d \neq 0$ and the triple check turns to cheater identification. There, if $\mathsf{Identify}$ identifies an honest party then this is distinguishable between real and ideal world as $\mathcal{S}$ always aborts in the ideal world, but this can happen with only with probability $\mathsf{negl}(\lambda)$. The other abort that can happen is if no cheater is identified from running $\mathsf{Identify}$ on $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ and from the opening of all $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ sessions, i.e. each party consistently committed to the outputs of $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$, but these did not form consistent multiplication triples. In this case, $\mathcal{S}$ always aborts with $\mathcal{P}_\mathcal{A}$ in the ideal world. In the real world, since $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ UC-securely implements $\mathcal{F}_{\mathsf{Triple}}$, and each party acted honestly during $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ (except with probability $\mathsf{negl}(\lambda)$ as otherwise $\mathsf{Identify}$ would have identified the party deviating from the protocol as having cheated by the Identifiable Cheating of $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$), the outputs of $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ in case of no abort must be valid multiplication triples except with probability $\mathsf{negl}(\lambda)$ by assumption. □

7. MPC with Friends and Foes

7.1. Introduction

A set of n mutually distrusting parties who have secrets $x_1, \ldots, x_n$ can use *secure multi-party computation* (MPC) [BGW88; Yao86] to compute a joint function $f(x_1, \ldots, x_n)$ of their secrets, without revealing anything more about those secrets to one another. MPC is typically parametrized by a threshold t such that as long as t or fewer participants collude, they cannot subvert the privacy and correctness guarantees of the computation. However, if t parties deviate from the protocol, no guarantees are made about what the remaining $n - t$ parties learn. Many MPC protocols (such as [IKP10; IKKP15; PR18]) make use of this by relying on fall-back protocols where, in the event of cheating, if one or more parties are identified as definitely *not* being one of the t cheaters, they are entrusted with the others' secrets.

Of course, this is not what we would like to use in practice. We would like even our honest peers — who do not collude with some central malicious adversary — not to learn our secrets. Alon *et al.* [AOP20] introduce MPC with Friends and Foes (or MPC with FaF security), which captures exactly this guarantee. Informally, a protocol achieves (t, h^*)-FaF security if, as in the standard definition of MPC, for any (non-uniform) adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}_\mathcal{A}$ which produces a view indistinguishable from that of the t corrupt parties without seeing the inputs of the honest parties. However, for FaF security, there must additionally exist a simulator $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ for every subset of up to h^* of the honest parties which produces a view indistinguishable from that of those honest parties, without seeing the inputs of the *remaining* honest parties. This implies that no matter what messages the corrupt parties send, they can not cause any h^* honest parties to learn more about their peers' inputs than they should.

Alon *et al.* define two degrees of FaF security:

Weak FaF. Here, though the output of $\mathcal{S}_\mathcal{A}$ must be indistinguishable from the real view of the t corrupt parties and the output of $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ must be indistinguishable from the real view of the h^* honest parties, taken together, those views may be distinguishable from the set of real views. (That is, the simulated views may not be mutually consistent.)

Strong FaF. Here, the outputs of $\mathcal{S}_\mathcal{A}$ and $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ must be *jointly* indistinguishable from the real views of the t corrupt parties and h^* honest parties.

One can think of strong FaF as modeling the case where the adversary receives some feedback about what the honest parties learned, and weak FaF as modeling the case where there is no such feedback.

7.1.1. Prior Work

Alon *et al.* showed some inherent limitations on MPC with FaF security (Section 7.1.1), and gave some initial constructions (Section 7.1.1). Their results primarily focus on the notion of guaranteed output delivery (GOD) where corrupt parties cannot prevent the honest parties from obtaining the output.

Limitations

Alon *et al.* consider two parameters of MPC protocols: number of rounds, and thresholds. They showed that two-round MPC with weak FaF security (and thus also strong FaF security) and GOD is impossible even for the lowest possible thresholds ($t = h^* = 1$); so, three rounds is the best we could hope to achieve. They then showed that even weak FaF security (and thus also strong FaF security) is unachievable for certain thresholds irrespective of the number of rounds. In particular, let n be the number of participants, t the bound on the number of corrupt parties, and h^* the bound on the number of honest parties who should learn nothing about other honest parties' secrets. Alon *et al.* show the following:

- Weak FaF secure MPC with GOD is impossible when $2t + h^* \geq n$.
- Information theoretic (statistical) weak FaF secure MPC with GOD is impossible if:
  - $2t + 2h^* \geq n$ (even if broadcast is available).
  - $2t + 2h^* \geq n$ or $3t \geq n$ (when broadcast is *not* available).
- Information theoretic (perfect) weak FaF secure MPC with GOD is impossible (even when broadcast is available) when $3t + 2h^* \geq n$.

Constructions

Alon *et al.* give several initial constructions of FaF secure MPC with GOD. They describe a round-optimal (three-round) construction that achieves strong FaF security, but only for $5t + 3h^* < n$. They also describe a threshold-optimal ($2t + h^* < n$) construction that only achieves *weak* FaF security. Finally, they show several information theoretic constructions. We summarize all of the constructions in Figure 7.1.

7.1.2. Related Work

Exploring the potential of FaF security belongs in the general area of exploring the robustness of different security models. Robustness is a highly desirable feature because it removes a potential denial of service threat and supports user participation. Towards this goal there is work that has been done by Koti *et al.* [KPPS21] which proposes a robust Privacy Preserving Machine Learning (PPML) framework for a variety of machine learning tasks, Dalskov *et al.* [DEK21] in which the authors introduce a novel four-party

Construction	FaF Level	Security	Threshold	Rounds	Assumptions	Preprocessing
[AOP20]						
GMW-based	Weak	Comp	$2t + h^* < n$	$\mathsf{poly}(\kappa)$	OT & OWP	no
DI-based	Strong	Comp	$5t + 3h^* < n$	3	PRG	no
BGW-based	Strong	Stat IT	$2t + 2h^* < n$	$\mathsf{poly}(\kappa)$	Broadcast	no
BGW-based	Strong	Perf IT	$3t + 2h^* < n$	$\mathsf{poly}(\kappa)$	None	no
This Work						
TFHE-FaF	Weak	Comp	$2t + h^* < n$	3	Lattices & Broadcast	no
BGW-BT-Comp	Strong	Comp	$2t + h^* < n$	$O(\kappa)$	ETP	Beaver triples

Figure 7.1.: Our constructions compared to those of [AOP20]. Notation: n denotes the total number of participants, t denotes the bound on the number of corruptions (foes), h^* denotes the bound on the number of honest parties against whom we want privacy (friends), and κ denotes the multiplicative depth of the circuit being evaluated.

honest-majority MPC protocol with active security which has guaranteed output delivery (with some extensions to their main protocol), and [KPRS21] where the authors introduce a robust actively secure 4-party protocol for secure training and inference.

Exploring FaF security is a relatively new endeavour and the following works concurrent to ours have shown some promising results. In [KKPG22], Koti *et al.* show a concretely efficient (1,1)-FaF secure 5PC protocol and in [HKKPPP22] Hedge *et al.* prove the necessity of semi-honest oblivious transfer for FaF-secure protocols with optimal resiliency and they show a ring-based 4PC protocol, which achieves fairness and GOD in the case of optimal corruptions: 1 semi-honest and 1 malicious adversaries.

7.1.3. Our Contributions

In this paper, we close two of the gaps left open by the constructions of Alon *et al.*. We also extend the study of how FaF security relates to other security notions. The focus of our work is FaF security with GOD.

First, we give a three-round construction that achieves weak FaF security for $2t+h^* < n$ in the CRS (common reference string) model. This is the first construction that is optimal both in terms of the number of rounds and in terms of the threshold (even though it does not achieve the stronger notion of FaF). Second, we give a construction that achieves strong FaF security for $2t + h^* < n$. This is the first strong FaF construction to achieve the optimal threshold (though the number of rounds depends on the multiplicative depth of the function being computed). A caveat of our second construction is that it relies on correlated randomness.

Finally, we further the study of the relationship between FaF security and other related notions of security. Recalling the standard notions, while actively corrupt parties are completely controlled by the adversary and may deviate arbitrarily from the protocol; passively corrupt parties follow the protocol steps but leak their internal states to the adversary. Alon *et al.* showed that *Mixed Adversary* security (where the adversary can make t active corruptions and h^* passive ones) does not imply FaF security in the computational setting. We show the other direction; that FaF security does not imply

mixed adversary security. We additionally consider *Best of Both Worlds* (BoBW) security [IKLP06; Kat07] (where the adversary can either make t active corruptions or $t + h^*$ passive ones, but not both). We show that FaF security does not imply BoBW security, and vice versa. These results are summarized in Figure 7.2.

Technical Overview

Three-Round Weak FaF Construction Our three-round construction is based on decentralized threshold FHE (described in Section 7.4), and follows the blueprint of Gordon *et al.* [GLS15]. In the first round, the participants exchange public keys. They then encrypt their inputs *to the set of all participants' public keys*, and broadcast the resulting ciphertexts in the second round. Once they receive one another's ciphetexts, they perform the homomorphic computation of the function locally, and broadcast their individual partial decryptions in the third round. Everyone is then able to locally combine these partial decryptions and obtain the output. Gordon *et al.* show that this construction achieves guaranteed output delivery in the presence of a dishonest minority. We show that it has weak FaF security as long as $2t + h^* < n$.

Strong FaF Construction Our strong FaF construction is based on BGW [BGW88]. We proceed in three steps; first, we show that BGW with Beaver triple pre-processing [Bea92] achieves guaranteed output delivery in the presence of an adaptive mixed adversary making t fail-stop corruptions (where fail-stop corruptions are similar to passive corruptions, except that the parties may additionally choose to abort at any step) and h^* passive corruptions, as long as $2t + h^* < n$. We then apply the compiler of Canetti *et al.* [CLOS02], which relies on adaptively secure commitments and zero knowledge proofs, to instead allow our mixed adversary to make t *active* corruptions and h^* passive corruptions. Finally, we rely on the observation of Alon *et al.* that adaptive security implies strong FaF security to obtain our result.

Relation of FaF to Other Notions We consider FaF security, BoBW security and mixed adversary security. We describe several protocols that achieve some of these notions but not others, which, taken together with the results of Alon *et al.*, shows that all three notions are incomparable. In Figure 7.2 we summarize what we know about the relationship of FaF, BoBW and mixed adversaries.

7.1.4. Organization

In Section 7.2, we recall the definitions of FaF security. In Section 7.3, we describe our results about the relationship of FaF, BoBW and mixed adversary security. In Section 7.4, we describe decentralized threshold fully homomorphic encryption (dTFHE), which we use in one of our constructions. In Section 7.5 and 7.6, we describe our round optimal weak FaF and strong FaF constructions, respectively.

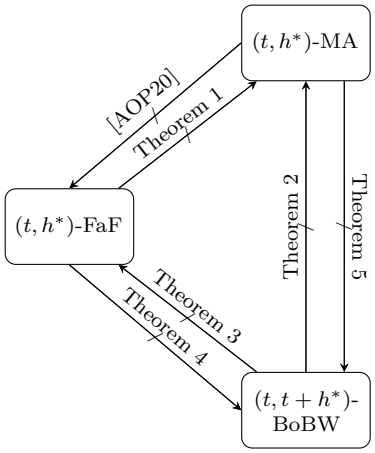


Figure 7.2.: Relationships of FaF to other notions. MA denotes security against mixed adversaries; BoBW denotes (active / passive) best of both worlds security.

7.1.5. Notation

We use λ to denote the security parameter. By $\mathsf{poly}(\lambda)$ we denote a polynomial function in λ. By $\mathsf{negl}(\lambda)$ we denote a negligible function; that is, a function f such that $f(\lambda) < \frac{1}{p(\lambda)}$ holds for any polynomial $p(\cdot)$ and sufficiently large λ. We denote the set $\{1, \ldots, k\}$ by $[k]$ (or, equivalently, by $[1, \ldots, k]$).

7.2. Definitions

In this section, we recall the definitions given by Alon *et al.* [AOP20] of Friends and Foes security. They consider a classical adversary $\mathcal{A}$ corrupting t of parties, who would like to leak unauthorized information to some honest parties. So, we really have *two separate adversaries* in this setting:

1. The adversary $\mathcal{A}$ who *actively* corrupts a subset $\mathcal{I} \subseteq [n]$ of the parties, meaning that she can instruct the parties in $\mathcal{I}$ to arbitrarily deviate from the protocol. ($\mathcal{A}$ is given auxiliary input $y_{\mathcal{A}}$.)

2. An adversary $\mathcal{A}_{\mathcal{H}^*}$ who *passively* corrupts a subset $\mathcal{H}^* \subseteq [n] \setminus \mathcal{I}$ of the honest parties. ($\mathcal{A}_{\mathcal{H}^*}$ is given auxiliary input $y_{\mathcal{H}^*}$.)

For security parameter λ, inputs $\boldsymbol{x} = (x_1, \ldots x_n)$ and auxiliary inputs $y_{\mathcal{A}}, y_{\mathcal{H}^*}$, we define the following random variables for a real-world execution of protocol Π that computes a function f:

$\mathsf{OUT}^{\mathsf{REAL}}_{\mathcal{A},\Pi}(1^\lambda, \boldsymbol{x})$ is the output of the non-active parties (where, non-active refers to the honest and passively corrupt parties) $\mathcal{H} = [n] \setminus \mathcal{I}$.

$\mathsf{VIEW}^{\mathsf{REAL}}_{\mathcal{A},\Pi}(1^\lambda, \boldsymbol{x})$ is $\mathcal{A}$'s view during an execution of the protocol.

$\mathsf{VIEW}^{\mathsf{REAL}}_{\mathcal{A},\mathcal{A}_{\mathcal{H}^*},\Pi}(1^\lambda, \boldsymbol{x})$ is $\mathcal{A}_{\mathcal{H}^*}$'s view during an execution of the protocol. Since $\mathcal{A}$'s best strategy in order to leak information is to send her entire view, we assume that $\mathcal{A}_{\mathcal{H}^*}$'s view includes $\mathcal{A}$'s view.

We can now formalize the global view of the real world execution of Π:

$$\mathsf{REAL}^{\Pi,\mathcal{A},\mathcal{A}_{\mathcal{H}^*}}_{1^\lambda,\boldsymbol{x},y_{\mathcal{A}},y_{\mathcal{H}^*}} = \left(\mathsf{VIEW}^{\mathsf{REAL}}_{\mathcal{A},\Pi}(1^\lambda, \boldsymbol{x}), \mathsf{VIEW}^{\mathsf{REAL}}_{\mathcal{A},\mathcal{A}_{\mathcal{H}^*},\Pi}(1^\lambda, \boldsymbol{x}), \mathsf{OUT}^{\mathsf{REAL}}_{\mathcal{A},\Pi}(1^\lambda, \boldsymbol{x})\right)$$

It is useful to define the following projection of the global view to the view of each of the adversaries and the non-active parties' output:

$$\mathsf{REAL}^{\Pi,\mathcal{A},\mathcal{A}_{\mathcal{H}^*}}_{1^\lambda,\boldsymbol{x},y_{\mathcal{A}},y_{\mathcal{H}^*}}(\mathcal{A}) = \left(\mathsf{VIEW}^{\mathsf{REAL}}_{\mathcal{A},\Pi}(1^\lambda, \boldsymbol{x}), \mathsf{OUT}^{\mathsf{REAL}}_{\mathcal{A},\Pi}(1^\lambda, \boldsymbol{x})\right)$$

and

$$\mathsf{REAL}^{\Pi,\mathcal{A},\mathcal{A}_{\mathcal{H}^*}}_{1^\lambda,\boldsymbol{x},y_{\mathcal{A}},y_{\mathcal{H}^*}}(\mathcal{A}_{\mathcal{H}^*}) = \left(\mathsf{VIEW}^{\mathsf{REAL}}_{\mathcal{A},\mathcal{A}_{\mathcal{H}^*},\Pi}(1^\lambda, \boldsymbol{x}), \mathsf{OUT}^{\mathsf{REAL}}_{\mathcal{A},\Pi}(1^\lambda, \boldsymbol{x})\right).$$

Similarly we can define the following random variables for an ideal-world execution:

$\mathsf{OUT}^{\mathsf{IDEAL}}_{\mathcal{A},f}(1^\lambda, \boldsymbol{x})$ is the output of the non-active parties in $\mathcal{H}$.

$\mathsf{VIEW}^{\mathsf{IDEAL}}_{\mathcal{A},f}(1^\lambda, \boldsymbol{x})$ is $\mathcal{A}$'s simulated view.

$\mathsf{VIEW}^{\mathsf{IDEAL}}_{\mathcal{A},\mathcal{A}_{\mathcal{H}^*},f}(1^\lambda, \boldsymbol{x})$ is $\mathcal{A}_{\mathcal{H}^*}$'s simulated view.

As before we can formalize the global view of the ideal world execution:

$$\mathsf{IDEAL}^{f,\mathcal{A},\mathcal{A}_{\mathcal{H}^*}}_{1^\lambda,\boldsymbol{x},y_{\mathcal{A}},y_{\mathcal{H}^*}} = \left(\mathsf{VIEW}^{\mathsf{IDEAL}}_{\mathcal{A},f}(1^\lambda, \boldsymbol{x}), \mathsf{VIEW}^{\mathsf{IDEAL}}_{\mathcal{A},\mathcal{A}_{\mathcal{H}^*},f}(1^\lambda, \boldsymbol{x}), \mathsf{OUT}^{\mathsf{IDEAL}}_{\mathcal{A},f}(1^\lambda, \boldsymbol{x})\right)$$

We define the following projections:

$$\mathsf{IDEAL}^{f,\mathcal{A},\mathcal{A}_{\mathcal{H}^*}}_{1^\lambda,\boldsymbol{x},y_{\mathcal{A}},y_{\mathcal{H}^*}}(\mathcal{A}) = \left(\mathsf{VIEW}^{\mathsf{IDEAL}}_{\mathcal{A},f}(1^\lambda, \boldsymbol{x}), \mathsf{OUT}^{\mathsf{IDEAL}}_{\mathcal{A},f}(1^\lambda, \boldsymbol{x})\right)$$

and

$$\mathsf{IDEAL}^{f,\mathcal{A},\mathcal{A}_{\mathcal{H}^*}}_{1^\lambda,\boldsymbol{x},y_{\mathcal{A}},y_{\mathcal{H}^*}}(\mathcal{A}_{\mathcal{H}^*}) = \left(\mathsf{VIEW}^{\mathsf{IDEAL}}_{\mathcal{A},\mathcal{A}_{\mathcal{H}^*},f}(1^\lambda, \boldsymbol{x}), \mathsf{OUT}^{\mathsf{IDEAL}}_{\mathcal{A},f}(1^\lambda, \boldsymbol{x})\right).$$

7.2.1. FaF Security.

We say that a protocol Π computes a functionality f with (t, h^*)-FaF security if the two following simulators exist for any adversary $\mathcal{A}$ that statically corrupts at most t parties:

- A simulator $\mathcal{S}_{\mathcal{A}}$ which simulates $\mathcal{A}$'s view in the real world, and
- A simulator $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ which simulates the view of any subset $\mathcal{H}^*$ of size at most h^* of the honest parties, such that when given $\mathcal{S}_{\mathcal{A}}$'s entire state, $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ can generate a view that is indistinguishable from the real world view of $\mathcal{H}^*$.

We say that $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ is given the entire state of $\mathcal{S}_{\mathcal{A}}$ because in the real world, nothing stops an adversary from sending her entire view to one (or more) honest parties.

FaF Functionality with GOD In an ideal evaluation of the function f, the parties interact with the functionality as follows:

Inputs. Each party P_i is given input x_i. Adversary $\mathcal{A}$ is given auxiliary input $y_{\mathcal{A}} \in \{0,1\}^*$ and x_i for all $i \in \mathcal{I}$. Adversary $\mathcal{A}_{\mathcal{H}^*}$ is given auxiliary input $y_{\mathcal{H}^*} \in \{0,1\}^*$ and x_i for all $i \in \mathcal{H}^*$.

Parties Send Input. All non-active parties (i.e. the honest and passive parties) $i \in [n] \setminus \mathcal{I}$ send their inputs x_i to the functionality. $\mathcal{A}$ chooses inputs x_i' for $i \in \mathcal{I}$ as the input of each corrupt party and sends it to the functionality. For non-active parties i, we define $x_i' := x_i$.

Computation. The functionality computes $z = (z_1, \ldots, z_n) = f(x_1', \ldots, x_n')$ and sends z_i to each party i.

$\mathcal{A}_{\mathcal{H}^*}$ **receives $\mathcal{A}$'s state.** $\mathcal{A}_{\mathcal{H}^*}$ receives $\mathcal{A}$'s randomness, inputs, auxiliary input, and z_i for $i \in \mathcal{I}$.

Output. Each non-active party i outputs z_i, while the corrupted parties output nothing. $\mathcal{A}_{\mathcal{H}^*}$ and $\mathcal{A}$ output some function of their view.

Weak and Strong FaF Definitions In the following, we use $\equiv$ to denote computational indistinguishability.

Definition 7.1 (Weak FaF)**.** Let Π be a protocol for computing f. We say that Π computes f with computational *weak* (t, h^*)-FaF security (with GOD), if the following holds. For every non-uniform PPT adversary $\mathcal{A}$ controlling a set $\mathcal{I} \subset [n]$ of size at most t in the real world, there exists a non-uniform PPT simulator $\mathcal{S}_{\mathcal{A}}$ controlling $\mathcal{I}$ in the ideal world; and for every subset of the remaining parties $\mathcal{H}^* \subset [n] \setminus \mathcal{I}$ of size at most h^* controlled by a non-uniform passive PPT adversary $\mathcal{A}_{\mathcal{H}^*}$ there exists a non uniform PPT simulator $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$, controlling $\mathcal{H}^*$ in the ideal world such that

$$\mathsf{IDEAL}^{\mathcal{S}_{\mathcal{A}}, \mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}}_{1^\lambda, \boldsymbol{x}, y_{\mathcal{A}}, y_{\mathcal{H}}}(\mathcal{S}_{\mathcal{A}}) \equiv \mathsf{REAL}^{\mathcal{A}, \mathcal{A}_{\mathcal{H}}}_{1^\lambda, \boldsymbol{x}, y_{\mathcal{A}}, y_{\mathcal{H}}}(\mathcal{A})$$

and

$$\mathsf{IDEAL}^{\mathcal{S}_{\mathcal{A}}, \mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}}_{1^\lambda, \boldsymbol{x}, y_{\mathcal{A}}, y_{\mathcal{H}}}(\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}) \equiv \mathsf{REAL}^{\mathcal{A}, \mathcal{A}_{\mathcal{H}}}_{1^\lambda, \boldsymbol{x}, y_{\mathcal{A}}, y_{\mathcal{H}}}(\mathcal{A}_{\mathcal{H}^*})$$

for any set of inputs $\boldsymbol{x} \in (\{0,1\}^*)^n$, any auxiliary inputs $(y_{\mathcal{A}}, y_{\mathcal{H}^*}) \in (\{0,1\}^*)^2$, and any large enough security parameter $\lambda \in \mathbb{N}$.

Definition 7.2 (Strong FaF)**.** For $\mathcal{A}, \mathcal{S}_{\mathcal{A}}, \mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ defined as in Definition 7.1, we say that Π computes f with computational strong (t, h^*)-FaF security (with GOD), if

$$\mathsf{IDEAL}^{\mathcal{S}_{\mathcal{A}}, \mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}}_{1^\lambda, \boldsymbol{x}, y_{\mathcal{A}}, y_{\mathcal{H}}} \equiv \mathsf{REAL}^{\mathcal{A}, \mathcal{A}_{\mathcal{H}}}_{1^\lambda, \boldsymbol{x}, y_{\mathcal{A}}, y_{\mathcal{H}}}$$

for any set of inputs $\boldsymbol{x} \in (\{0,1\}^*)^n$, any auxiliary inputs $(y_{\mathcal{A}}, y_{\mathcal{H}^*}) \in (\{0,1\}^*)^2$, and any large enough security parameter $\lambda \in \mathbb{N}$.

The main difference is that in the strong notion of FaF security we want the simulated views of $\mathcal{A}$ and $\mathcal{A}_{\mathcal{H}^*}$ to be indistinguishable from the real views *even when taken together*.

7.3. Relation of FaF to Other Notions

Somewhat surprisingly, Alon *et al.* show that standard security against a static adversary making $t + h^*$ active corruptions does not imply (t, h^*) FaF security. Informally, this is because the simulator $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ for the honest parties is not allowed to choose which input to send to the ideal functionality, so it does not have as much power as the standard security simulator $\mathcal{S}$. Security against a (t, h^*) mixed adversary making t active corruptions and h^* passive corruptions also does not imply (t, h^*) FaF security. This is because the mixed adversary simulator can decide the active parties' inputs based on the passive parties' inputs; however, the FaF simulator $\mathcal{S}_{\mathcal{A}}$ does not know any of the honest parties' inputs when simulating.

On the other hand, Alon *et al.* show that security against an *adaptive* adversary making $t + h^*$ active corruptions *does* imply (t, h^*) FaF security. This is because a simulator for an adaptive adversary needs to be able to handle corruptions which occur after the end of the protocol execution, at which point it cannot choose the input even for actively corrupt parties. We observe that the proof given by Alon *et al.* also shows that security against an adaptive *mixed* (t, h^*) adversary making t active corruptions and h^* passive corruptions implies (t, h^*) FaF security, and we use this in our strong FaF construction.

Here, we further explore the relationship between FaF security and other security notions. First, we show the other direction: that (t, h^*) FaF security does not imply security against a (t, h^*) mixed adversary making t active corruptions and h^* passive corruptions, making the FaF and mixed adversary models incomparable. We do so by giving an example (Example 1) of a protocol that achieves (t, h^*) FaF security but not (t, h^*) mixed adversary security. We also consider $(t, t + h^*)$ Best of Both Worlds (BoBW) security, where the same protocol must tolerate either t active corruptions *or* $t + h^*$ passive corruptions (but not both). We show that BoBW is incomparable to both FaF and mixed adversaries.

Example 1 ($\Pi_{\neg\mathsf{MA}}$). Consider a function $f(x_1, \ldots, x_n)$ such that the output of f does not reveal the set of inputs $(x_1, \ldots, x_n)$ such as the XOR function. So we have that $f(x_1, \ldots, x_n) = x_1 \oplus \ldots \oplus x_n$ Suppose a protocol Π_{FaF} computes any f with (t, h^*) FaF security [1]. Now, consider a function $g((x_1, \rho_1), \ldots, (x_n, \rho_n))$, where each party P_i has an additional input ρ_i. g returns $(x_1, \ldots, x_n)$ to everyone if at least $t + 1$ of the ρ_i's are equal, and returns $f(x_1, \ldots, x_n)$ to everyone otherwise. The following protocol $\Pi_{\neg\mathsf{MA}}$ computes $f(x_1, \ldots, x_n)$ with (t, h^*) FaF security but not with security against a (t, h^*) mixed adversary.

1. Each party P_i chooses ρ_i uniformly at random from a large space.
2. The parties use Π_{FaF} to compute $g((x_1, \rho_1), \ldots, (x_n, \rho_n))$.

[1] Here, it is implicity assumed that the values of (t, h^*) are such that they admit FaF security.

3. Each party outputs the value returned by Π_{FaF}.

Theorem 7.1. *Protocol $\Pi_{\neg\mathsf{MA}}$ (Example 1) computes $f(x_1, \ldots, x_n)$ with (t, h^*) FaF security but not with security against a (t, h^*) mixed adversary making t active corruptions and h^* passive corruptions.*

Proof. $\Pi_{\neg\mathsf{MA}}$ achieves FaF security, because the adversary cannot possibly guess the honest parties' randomly chosen values ρ_i, and so cannot exploit the additional leakage given by the output of g.

However, the mixed adversary knows h^* passive party inputs and randomly chosen ρ_i's, and can choose one of those to set corrupt parties' ρ_i's to. This allows the mixed adversary to learn all parties' inputs. This is clearly insecure due to our assumption that the set of inputs are not revealed by the output of f. □

Remark 2. We observe that since the above reduction (Example 1) is information-theoretic, plugging in the statistically-secure FaF protocol of [AOP20] to instantiate Π_{FaF} would yield a statistically-secure protocol $\Pi_{\neg\mathsf{MA}}$ that satisfies (t, h^*) FaF security but not (t, h^*) mixed security. This shows a separation between statistical FaF and mixed security at the protocol level, which was left as an open question in [AOP20] (the only known separation at the protocol level was for computational security).

Theorem 7.2. *If Π_{FaF} from Example 1 is replaced with a protocol Π_{BoBW} which has $(t, t + h^*)$ BoBW security, then protocol $\Pi_{\neg\mathsf{MA}}$ (Example 1) computes $f(x_1, \ldots, x_n)$ with $(t, t + h^*)$ BoBW security, but not with security against a (t, h^*) mixed adversary making t active corruptions and h^* passive corruptions.*

Proof. $\Pi_{\neg\mathsf{MA}}$ achieves BoBW security, since an adversary making just t corruptions cannot guess honest parties' ρ_i values and thus cannot exploit the additional leakage, and an adversary making $t + h^*$ corruptions cannot dishonestly choose passive parties' values ρ_i to be equal.

However, as in the proof of Theorem 7.1, the mixed adversary knows h^* passive party inputs and randomly chosen ρ_i's, and can choose one of those to set corrupt parties' ρ_i's to. □

We next show that BoBW security does not imply FaF security, by giving an example (Example 2) of a protocol that achieves $(t, t + h^*)$ BoBW security but not (t, h^*) FaF security.

Example 2 ($\Pi_{\neg\mathsf{FaF}}$). Consider a function $f(x_1, \ldots, x_n)$ as in Example 1. Suppose a protocol Π_{BoBW} that computes any function with $(t, t+h^*)$ BoBW security. The following protocol $\Pi_{\neg\mathsf{FaF}}$ computes $f(x_1, \ldots, x_n)$ with $(t, t + h^*)$ BoBW security, but not with (t, h^*) FaF security.

1. The parties use Π_{BoBW} to compute $f(x_1, \ldots, x_n)$.

2. If a party P_i receives a special "attack" message from t parties, it sends its input x_i to the *other* $n - t$ parties. (Note that *sending* an "attack" message is not part of the instructions.)

3. Each party outputs the value returned by Π_{BoBW}.

Theorem 7.3. *Protocol $\Pi_{\neg\mathsf{FaF}}$ (Example 2) computes $f(x_1, \ldots, x_n)$ with $(t, t+h^*)$ BoBW security, but not with (t, h^*) FaF security.*

Proof. $\Pi_{\neg\mathsf{FaF}}$ achieves $(t, t + h^*)$ BoBW security: if there are only passive corruptions, no party will send an "attack" message, and thus the second step of $\Pi_{\neg\mathsf{FaF}}$ will never come into play. If there are only t active corruptions, they are able to trigger the attack, but will not learn the honest parties' inputs because those inputs are only sent to parties who didn't send attack messages. $\Pi_{\neg\mathsf{FaF}}$ does not achieve (t, h^*) FaF security: the t corrupt parties can easily trigger an attack, causing all the honest parties to learn one another's inputs. This violates security due to our assumption that the output of f does not reveal any party's input. □

It remains to show that neither FaF or mixed adversary security imply BoBW security. This follows from the fact that in both FaF and mixed adversary security, the simulator can choose the inputs of the actively corrupt parties. However, in BoBW security, in the case where the adversary only makes passive corruptions, the simulator is unable to choose the inputs of any parties.

Example 3 ($\Pi_{\neg\mathsf{BoBW}}$). Consider a function $f(x_1, \ldots, x_n)$ as in the previous examples, and a protocol Π_{FaF} that computes f with (t, h^*) FaF security. Now, consider a function $g((x_1, y_1), \ldots, (x_n, y_n))$, where each party P_i has an additional input y_i. g returns $y_i \wedge f(x_1, \ldots, x_n)$ to each party P_i. It returns $\perp$ to everyone else. The following protocol $\Pi_{\neg\mathsf{BoBW}}$ computes $g((x_1, y_1), \ldots, (x_n, y_n))$ with (t, h^*) FaF security but not with $(t, t+h^*)$ BoBW security.

1. The parties use Π_{FaF} to compute $z = f(x_1, \ldots, x_n)$.

2. Each party P_i outputs $y_i \wedge z$.

Theorem 7.4. *Protocol $\Pi_{\neg\mathsf{BoBW}}$ (Example 3) computes $g((x_1, y_1), \ldots, (x_n, y_n))$ with (t, h^*) FaF security but not with $(t, t + h^*)$ BoBW security.*

Proof. $\Pi_{\neg\mathsf{BoBW}}$ achieves (t, h^*) FaF security: a simulator can always set one of the actively corrupt parties' auxiliary inputs y_i to be 1 to learn the output of f.

However, it does *not* achieve BoBW security: in the case where the adversary can only make passive corruptions, if all parties' auxiliary inputs y_i are 0, the simulator does not learn the output of f [2], which it needs in order to simulate successfully. □

[2]We assume that the function f is such that the output of f depends on the inputs of all parties.

Theorem 7.5. *If Π_{FaF} from Example 3 is replaced with a protocol Π_{MA} which has (t, h^*) mixed adversary security, then protocol $\Pi_{\neg\mathsf{BoBW}}$ (Example 3) computes $g((x_1, y_1), \ldots, (x_n, y_n))$ with (t, h^*) mixed adversary security but not with $(t, t + h^*)$ BoBW security.*

The proof is the same as the proof of Theorem 7.4.

Remark 3. We design the function g in such a way that *any* party can learn the output of f by setting their auxiliary input y_i to 1. This gives us FaF and mixed adversary security; no matter whom the adversary actively corrupts, the simulator can use that party to learn the output of f, and simulate for the rest.

7.4. Building Block: Decentralized Threshold FHE

We recap the definitions of d-out-of-n decentralized threshold fully homomorphic encryption (dTFHE) as presented by Boneh et al. [BGGJKRS18].

Syntax A dTFHE scheme is a tuple of PPT algorithms (DistGen, Enc, Eval, PDec, Combine, SimPDec) with the following syntax:

$\mathsf{DistGen}(1^\lambda, 1^\kappa, i; \rho_i) \to (\mathtt{pk}_i, \mathtt{sk}_i)$**:** On input the security parameter λ, a depth bound κ, party index i and randomness ρ_i, the distributed setup outputs a public-secret key pair $(\mathtt{pk}_i, \mathtt{sk}_i)$ for party i. The public key of the scheme is denoted by $\mathtt{pk} = (\mathtt{pk}_1 \parallel \mathtt{pk}_2 \parallel \ldots \parallel \mathtt{pk}_n)$.

$\mathsf{Enc}\,(\mathtt{pk}, m; \rho) \to \mathtt{c}$**:** On input a public key $\mathtt{pk}$ and a plaintext m in the message space $\mathcal{M}$, the randomized algorithm outputs a ciphertext $\mathtt{c}$.

$\mathsf{Eval}\,(\mathtt{pk}, C, \mathtt{c}_1, \ldots, \mathtt{c}_k) \to \mathtt{c}$**:** On input a public key $\mathtt{pk}$, a circuit $C : \mathcal{M}^k \to \mathcal{M}$ of depth at most κ, and a set of k ciphertexts $\mathtt{c}_1, \ldots, \mathtt{c}_k$ (where $k = \mathsf{poly}(\lambda)$), the evaluation algorithm outputs an encrypted evaluation $\mathtt{c}$.

$\mathsf{PDec}\,(\mathtt{pk}, \mathtt{sk}_i, \mathtt{c}) \to \mathtt{d}_i$**:** On input the public key $\mathtt{pk}$, a ciphertext $\mathtt{c}$ and a secret key $\mathtt{sk}_i$ the algorithm outputs a partial decryption $\mathtt{d}_i$.

$\mathsf{Combine}\,(\mathtt{pk}, \{\mathtt{d}_i\}_{i\in S}) \to m \backslash \perp$**:** On input a public key $\mathtt{pk}$ and a set partial decryptions $\{\mathtt{d}_i\}_{i\in S}$ where $S \subseteq [n]$, the combination algorithm outputs a plaintext m or the symbol $\perp$.

$\mathsf{SimPDec}\,(\mathtt{c}, \mathtt{pk}, \{\mathtt{sk}_i\}_{i\in\mathcal{I}}, z) \to \{\mathtt{d}_i\}_{i\in[n]\setminus\mathcal{I}}$**:** On input a ciphertext $\mathtt{c}$, the public key $\mathtt{pk}$, the secret keys of at most d parties, and the target plaintext z, the simulated decryption algorithm outputs partial decryptions on behalf of the rest of the parties which are consistent with $\mathtt{c}$ decrypting to z.

Properties As in a standard homomorphic encryption scheme, we require that a dTFHE scheme satisfies correctness and security, which we describe informally below and provide the formal definition after the informal introduction.

Correctness. Informally, a dTFHE scheme is said to be *correct* if combining at least $d + 1$ partial decryptions of any honestly generated ciphertext output by the evaluation algorithm returns the correct evaluation of the corresponding circuit on the underlying plaintexts.

Semantic Security. Informally, a dTFHE scheme satisfies *semantic security* if no PPT adversary can distinguish between encryptions of a pair of (adversarially chosen) plaintext messages m_0 and m_1 of the same length, even given the secret keys corresponding to a subset $\mathcal{I}$ of the parties for any set $\mathcal{I}$ of size at most d. Since we use the dTFHE scheme as a tool in our semi-malicious MPC construction[3], we define the notion with respect to a semi-malicious adversary $\mathcal{A}$.

Simulation Security. Informally, a dTFHE scheme satisfies *simulation security* if there exists an efficient algorithm SimPDec that takes as input a ciphertext c, the public key $\mathtt{pk}$, the secret keys of at most d parties and the target plaintext z, and outputs a set of partial decryptions on behalf of the rest of the parties such that its output is computationally indistinguishable from the output of the real algorithm PDec that outputs partial decryptions of the ciphertext c using the corresponding secret keys for the same subset of parties. Similar to semantic security, we define this notion with respect to a semi-malicious adversary $\mathcal{A}$.

After the informal, high level description of the desired properties we present the formal definitions below:

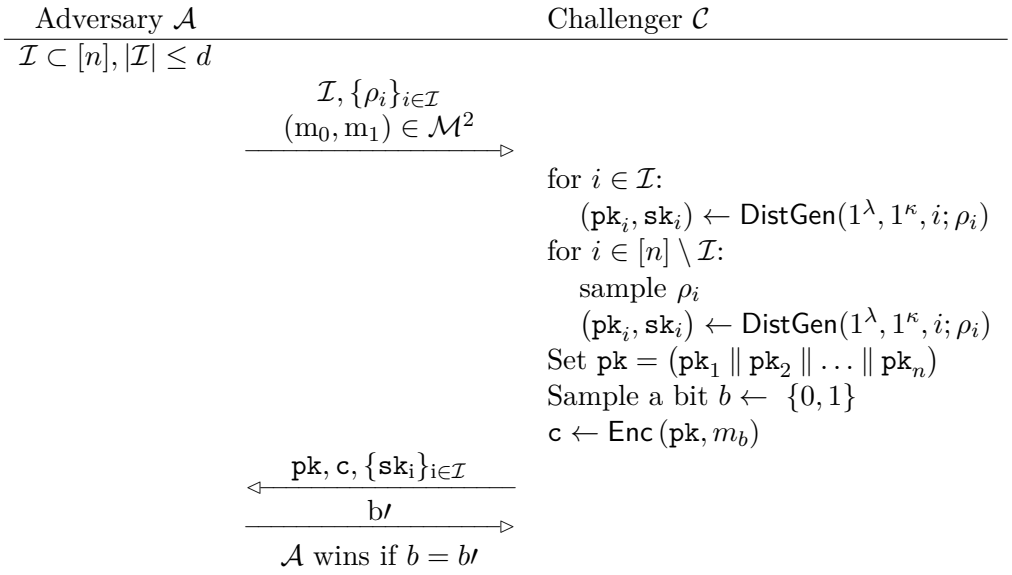
Correctness. A dTFHE scheme is *correct* if for all sufficiently large λ, all $k = \mathsf{poly}(\lambda)$, circuits $C : \mathcal{M}^k \to \mathcal{M}$ of depth at most κ and $m_i \in \mathcal{M}$ for $i \in [k]$, the following condition holds:

Let $(\mathtt{pk}_j, \mathtt{sk}_j) \leftarrow \mathsf{DistGen}(1^\lambda, 1^\kappa, j)$ for all $j \in [n]$, $\mathtt{pk} = (\mathtt{pk}_1 || \ldots || \mathtt{pk}_n)$; let $\mathsf{c}_i \leftarrow \mathsf{Enc}(\mathtt{pk}, m_i)$ for all $i \in [k]$; compute $\mathsf{c} \leftarrow \mathsf{Eval}(\mathtt{pk}, C, \mathsf{c}_1, \ldots, \mathsf{c}_k)$. For any $S \subseteq [n]$, $|S| > d$,

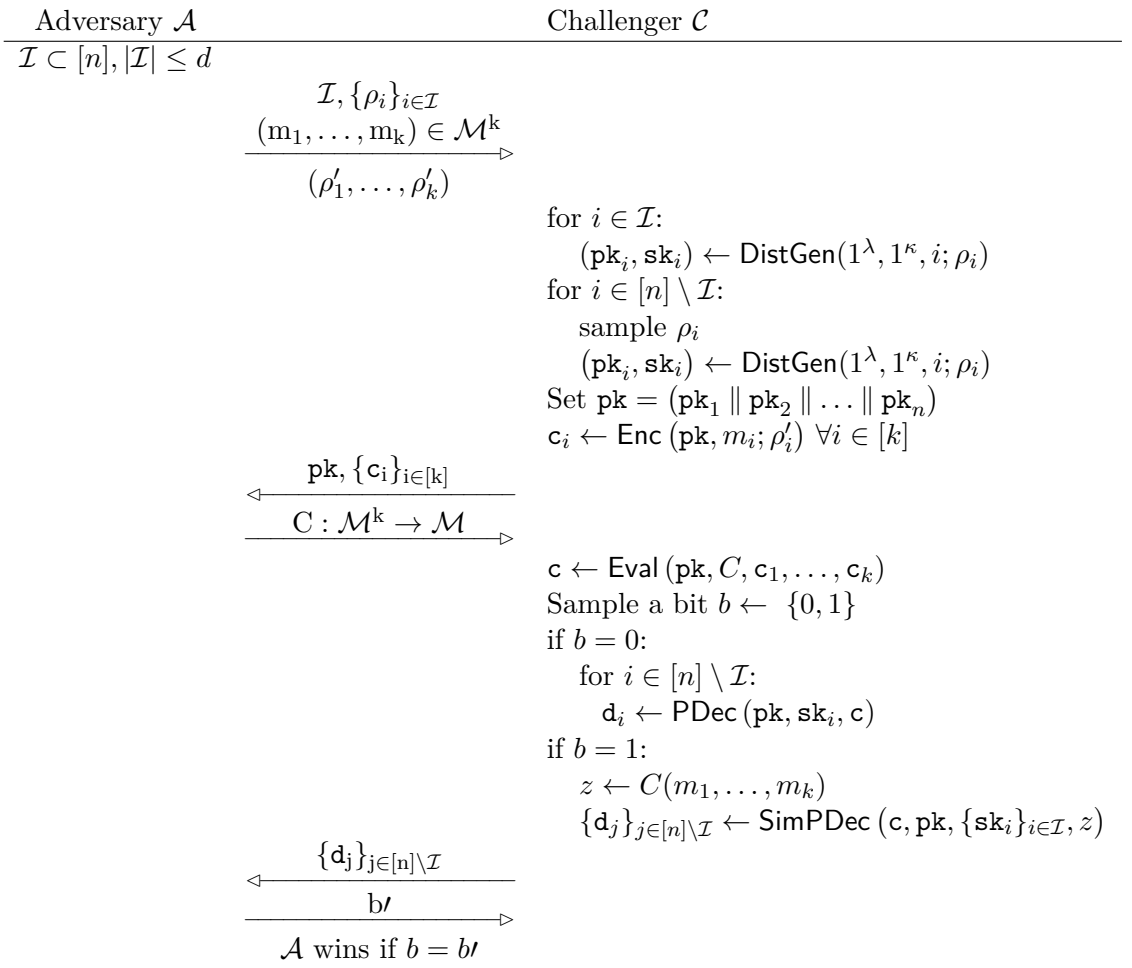
$$\mathsf{Pr}[\mathsf{Combine}(\mathtt{pk}, \{\mathsf{PDec}(\mathtt{pk}, \mathtt{sk}_j, \mathsf{c})\}_{j \in S}) = C(m_1, \ldots, m_k)] \geq 1 - \mathsf{negl}(\lambda).$$

Semantic Security. A dTFHE scheme is *semantically secure* if for all sufficiently large security parameters λ, all depth bound κ and any PPT semi-malicious adversary $\mathcal{A}$, there exists a negligible function negl such that the probability that $\mathcal{A}$ wins the game below is less than $\frac{1}{2} + \mathsf{negl}(\lambda)$.

[3] where semi-malicious security [AJLTVW12] refers to security against an adversary who needs to follow the protocol specification, but has the liberty to decide the input and random coins in each round.



Simulation Security. A dTFHE scheme satisfies *simulation security* if there exists a simulator SimPDec such that for all sufficiently large security parameters λ, all depth bound κ, and any PPT semi-malicious adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the game below is less than $\frac{1}{2} + \mathsf{negl}(\lambda)$.



7.5. Three-Round MPC with Weak FaF and Guaranteed Output Delivery

In this section, we present a three-round MPC construction in the CRS model (where it is assumed that parties have access to a common reference string at the beginning of the protocol execution) that achieves weak FaF security and GOD when $n > 2t + h^*$. This is round-optimal, following the impossibility of two-round MPC with weak FaF security and GOD shown in [AOP20] (which holds even in the CRS model). Specifically, their result shows that there are functionalities that cannot be computed with $(1, 1)$ weak FaF security and GOD in less than three rounds, for any $n \geq 3$ [4].

Our construction is based on the three-round construction of Gordon *et al.* [GLS15] that achieves standard security with GOD against $t < n/2$ active corruptions. At a high-level, this construction uses the tool of distributed threshold fully homomorphic encryption

[4]The proof in [AOP20] is a simple observation regarding the impossibility of computing the AND functionality with GOD in two rounds against 2 corrupted parties, proved by Gennaro et al. [GIKR02]; which holds in the common reference string (CRS) model.

scheme (dTFHE) with threshold t and proceeds as follows. First, the distributed setup allows the parties to obtain their individual public / secret key pairs. Each of them broadcasts their public key. Next, the parties broadcast encryptions of their input, which can be homomorphically evaluated to compute an encryption of the output. In the last round, the parties compute and broadcast partial decryptions of the output ciphertext, which can be combined to obtain the output.

We observe that the above construction admits (t, h^*) weak FaF security and GOD when $n > 2t + h^*$, if a dTFHE scheme with threshold $(t + h^*)$ is used instead. Intuitively, security of such a dTFHE scheme ensures that the joint view of the active and passively corrupt parties comprising of $(t + h^*)$ secret keys does not reveal any information about the inputs of the honest parties (beyond the output of computation). Correctness of such a scheme ensures that even if up to t parties abort, guaranteed output delivery is achieved because the partial decryptions sent by the remaining $n - t > t + h^*$ parties suffice to compute the output.

Similar to the work of Gordon *et al.* [GLS15], we present a three-round construction $\Pi^{\mathsf{sm}}_{\mathsf{wFaF}}$ that is secure against semi-malicious adversaries. Semi-malicious security was introduced by Aashrov *et al.* [AJLTVW12] and subsequently used in many works as a stepping-stone on the way to achieving active security. Recall that a semi-malicious adversary needs to follow the protocol specification, but has the liberty to decide the input and random coins in each round. Additionally, the parties controlled by the semi-malicious adversary may choose to abort at any time. To upgrade a semi-malicious construction to achieve active security, the general round-preserving compiler of Asharov *et al.* uses UC NIZKs (non-interactive zero-knowledge proofs) in the CRS model. We should point out that since FaF security becomes relevant only in settings with more than 3 parties, the standard security notion for 2-party NIZK used as a building block would not affect FaF security of the overall protocol.

We give a formal description of the protocol $\Pi^{\mathsf{sm}}_{\mathsf{wFaF}}$ below.

Inputs: Each party P_i has an input $x_i \in \{0,1\}^\lambda$.

Output: $f(x_1, \ldots, x_n)$, where the function f is represented by a circuit C.

Tools: A $(t + h^*)$-out-of-n dTFHE scheme $(\mathsf{DistGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{PDec}, \mathsf{Combine}, \mathsf{SimPDec})$. Such a scheme can be built based on LWE [BGGJKRS18; GLS15].

Round 1: Each party P_i does the following:

- Computes $(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow \mathsf{DistGen}(1^\lambda, 1^\kappa, i; \rho_i)$ using randomness ρ_i.
- Broadcasts $\mathtt{pk}_i$.

Round 2: All parties set $\mathtt{pk} := (\mathtt{pk}_1 || \ldots || \mathtt{pk}_n)$ (where a default public key is used corresponding to parties who have aborted in the first round).

Each party P_i does the following:

- Computes the encryption of its input as $\mathsf{c}_i \leftarrow \mathsf{Enc}(\mathtt{pk}, x_i)$.

- Broadcasts c_i.

Round 3: Each party P_i does the following:

- Computes the homomorphic evaluation of the circuit C on the ciphertexts as $\mathsf{c} \leftarrow \mathsf{Eval}(\mathtt{pk}, C, \mathsf{c}_1, \ldots, \mathsf{c}_n)$, where c_j is computed using default input and randomness if P_j aborted during the previous rounds.
- Computes her own partial decryption as $\mathtt{d}_i \leftarrow \mathsf{PDec}(\mathtt{pk}, \mathtt{sk}_i, \mathsf{c})$.
- Broadcasts the partial decryption $\mathtt{d}_i$.

Output Computation: Let $S \subseteq [n]$ denote the set of parties who have not yet aborted. Each party combines the partial decryptions broadcast by parties in S to obtain the output as $z \leftarrow \mathsf{Combine}(\mathtt{pk}, \{\mathtt{d}_j\}_{j \in S})$.

Protocol $\Pi^{\mathsf{mal}}_{\mathsf{wFaF}}$. Let $\Pi^{\mathsf{mal}}_{\mathsf{wFaF}}$ denote the three-round construction obtained by applying the compiler of Asharov *et al.* [AJLTVW12] to the three-round protocol $\Pi^{\mathsf{sm}}_{\mathsf{wFaF}}$ in the semi-malicious setting. In particular, in every round of $\Pi^{\mathsf{mal}}_{\mathsf{wFaF}}$, each party executes the actions of the corresponding round of $\Pi^{\mathsf{sm}}_{\mathsf{wFaF}}$ along with a non-interactive zero-knowledge proving that she is following the protocol consistently with respect to certain random coins. This compiler is round-preserving and preserves security of the underlying construction (i.e. if the underlying protocol achieves GOD, so does the compiled protocol).

We state the formal theorem below.

Theorem 7.6. *Let f be an efficiently computable n-party function and let $n > 2t + h^*$. Assuming a setup with CRS and the existence of a $(t+h^*)$-out-of-n decentralized threshold fully homomorphic encryption scheme, the three-round protocol $\Pi^{\mathsf{mal}}_{\mathsf{wFaF}}$ achieves (t, h^*) weak FaF security with guaranteed output delivery.*

Proof. To prove the theorem, we construct the simulators $\mathcal{S}_{\mathcal{A}}$ and $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ for $\Pi^{\mathsf{sm}}_{\mathsf{wFaF}}$ in the semi-malicious setting. This suffices to complete the proof, since active security would directly follow from the result of [AJLTVW12].

We begin with the description of $\mathcal{S}_{\mathcal{A}}$. Let $\mathcal{I}$ and $\mathcal{H}^*$ denote the set of indices of the t semi-malicious corrupt parties and the remaining parties respectively.

First Round: $\mathcal{S}_{\mathcal{A}}$ simulates public keys of honest parties by honestly generating key pairs.

Second Round: $\mathcal{S}_{\mathcal{A}}$ simulates input ciphertexts by broadcasting $\mathsf{c}_j \leftarrow \mathsf{Enc}(\mathtt{pk}, 0)$ on behalf of P_j, where $j \in [n] \setminus \mathcal{I}$.

Third Round: $\mathcal{S}_{\mathcal{A}}$ does the following:

- Computes $\mathsf{c} \leftarrow \mathsf{Eval}(\mathtt{pk}, C, \mathsf{c}_1, \ldots, \mathsf{c}_n)$ honestly, where c_i is computed using default input and randomness if P_i ($i \in \mathcal{I}$) aborted during the previous rounds.
- Reads the witness tape of the semi-malicious adversaries to learn the inputs x_i and secret keys $\mathtt{sk}_i$ for each $i \in \mathcal{I}$. If P_i has aborted, x_i is set as the default input.

- Invokes the ideal functionality $\mathcal{F}$ on inputs x_i for $i \in \mathcal{I}$ and receives the output z.
- Runs the simulated decryption algorithm to obtain partial decryptions as $\{\mathtt{d}_j\}_{j\in[n]\setminus\mathcal{I}} \leftarrow \mathsf{SimPDec}\left(\mathsf{c}, \mathtt{pk}, \{\mathtt{sk}_i\}_{i\in\mathcal{I}}, z\right)$.
- Broadcasts the partial decryption $\mathtt{d}_j$ on behalf of party P_j $(j \in [n] \setminus \mathcal{I})$.

Consider the following sequence of hybrids:

Hyb_0**:** Same as the real world execution of $\Pi^{\mathsf{sm}}_{\mathsf{wFaF}}$.

Hyb_1**:** Same as Hyb_0, except that the partial decryptions for honest parties are computed as $\mathsf{SimPDec}\left(\mathsf{c}, \mathtt{pk}, \{\mathtt{sk}_i\}_{i\in\mathcal{I}}, z\right)$ instead of $\mathsf{PDec}(\mathtt{pk}, \mathtt{sk}_i, \mathsf{c})$. It follows from simulation security of the dTFHE scheme that this hybrid is indistinguishable from the previous hybrid.

Hyb_2**:** Same as Hyb_1, except that the input ciphertexts of honest parties are computed as $\mathsf{c}_j \leftarrow \mathsf{Enc}(\mathtt{pk}, 0)$ using a dummy input 0 for $j \in [n] \setminus \mathcal{I}$, instead of using the actual input x_j. It follows from the semantic security of the dTFHE scheme that this hybrid is indistinguishable from the previous hybrid.

Since Hyb_2 corresponds to the ideal execution and every pair of consecutive hybrids are indistinguishable, this completes the proof that the view of $\mathcal{A}$ in the real world is indistinguishable from her view in the ideal world execution.

Next, suppose we fix the adversary $\mathcal{A}$ corrupting the parties in $\mathcal{I}$ where $\mathcal{I}$ is of size at most t, and let $\mathcal{H}^* \subseteq [n] \setminus \mathcal{I}$ of size at most h^*. The passive simulator $\mathcal{S}_{\mathcal{A}_{\mathcal{H}^*}}$ works very similarly to $\mathcal{S}_{\mathcal{A}}$. The only difference is that the messages the simulator sends to the adversary on behalf of the parties in $\mathcal{H}^*$ are the actual messages computed as per protocol specifications. For instance, the input ciphertexts will be computed as encryptions of the real inputs of parties in $\mathcal{H}^*$ (unlike in $\mathcal{S}_{\mathcal{A}}$ where it was computed as encryptions of dummy input 0). Similarly, the partial decryptions of the parties in $\mathcal{H}^*$ would be computed honestly using their secret keys (not as output of $\mathsf{SimPDec}$). Lastly, the sequence of hybrids and indistinguishability can be argued as above; it follows from the semantic and simulation security of the dTFHE scheme having threshold $(t + h^*)$.

□

Remark 4. Note that the protocol described in this section is not strongly FaF secure; this is because the simulator $\mathcal{S}_{\mathcal{A}}$ does not know the honest parties' inputs, and instead encrypts a default value on their behalf. However, this simulation cannot then be consistent with an honest party's simulated view, since their view must include their real input and randomness that maps that input to the ciphertext they broadcast.

Figure 7.3.: Getting (t, h^*)-FaF Security from BGW

7.6. Optimal-Threshold MPC with Strong FaF and Guaranteed Output Delivery

Alon *et al.* [AOP20] showed that even weak FaF with GOD is impossible if $2t + h^* \geq n$. Their proof holds even if arbitrary correlated randomness is available to the parties. So, the best that we could possible hope for is strong FaF with GOD and $2t + h^* + 1 = n$.

In this section we prove that BGW with Beaver triple preprocessing and augmented with adaptive zero knowledge proofs achieves exactly this. We do this in three steps, as described in Figure 7.3. First, in Section 7.6.1, we prove that BGW with Beaver triple preprocessing achieves security with GOD against adaptive mixed (fail-stop / passive) adversaries. Second, in Section 7.6.2, we use the compiler of Canetti *et al.* [CLOS02] to show that our protocol against adaptive mixed (fail-stop / passive) adversaries can be augmented with adaptive commitments and zero knowledge proofs of correct behavior in order to achieve security against adaptive mixed (*active* / passive) adversaries. Third, we invoke a theorem from Alon *et al.* [AOP20] to argue that any such protocol achieves GOD with strong FaF security.

We restate the theorem of Alon *et al.* [AOP20] which connects adaptive security to strong FaF below. Alon *et al.* prove that $(t + h^*)$ adaptive security implies (t, h^*) FaF; however, their proof can be strengthened (with no modifications necessary) to show that *adaptive mixed security with a corruption budget of t active corruptions and h^* passive corruptions* implies (t, h^*) FaF. As mentioned in 7.3, security against a (t, h^*) mixed adversary making t active corruptions and h^* passive corruptions does not imply (t, h^*) FaF security because the mixed adversary simulator can decide the active parties' inputs based on the passive parties' inputs but the FaF simulator $\mathcal{S}_{\mathcal{A}}$ does not know the honest parties' inputs. However, in the case of an adaptive mixed adversary, any party that is corrupted by an adaptive malicious adversary after the protocol has terminated can essentially be viewed as a passive corruption since the adversary cannot control their input.

Theorem 7.7 ([AOP20], Theorem 5.3)**.** *Let* $\mathsf{type} \in \{\text{computational}, \text{statistical}, \text{perfect}\}$*, and let* Π *be an n-party protocol computing some n-party functionality f with* type *adaptive mixed security with a corruption budget of t active corruptions and h^* passive corruptions. Then* Π *computes f with* type *strong (t, h^*)-FaF-security.*

7.6.1. Adaptive BGW Against Mixed (Fail-Stop / Passive) Adversaries

We first recall BGW (Section 7.6.1), and how Beaver triples can be used to improve the corruption threshold (Section 7.6.1).

Brief Overview of BGW Without Preprocessing

Let $\mathbb{F}$ be a finite field. A secret value s is secret shared using a polynomial $f_s(x) \in \mathbb{F}[x]$ of degree d such that $f_s(0) = s$ and each party P_i holds $f_s(i)$. The evaluation of the circuit then proceeds gate by gate. In order to add two secret shared values x and y, each party P_i can locally add the shares $f_x(i)$ and $f_y(i)$ that they are holding to get $f_{x+y}(i) = f_x(i) + f_y(i)$. This is a valid sharing of $x + y$, because f_{x+y} is of the same degree as f_x and f_y and $f_{x+y}(0) = f_x(0) + f_y(0) = x + y$. To reconstruct an output z, all parties broadcast their share $f_z(i)$, and everyone interpolates the polynomial.

Multiplication gates pose more of a challenge. If a party P_i computes $f'_{xy}(i) = f_x(i)f_y(i)$, she gets a point on a polynomial f'_{xy} such that $f'_{xy}(0) = xy$, which is what we wanted. The caveat is that f'_{xy} is of degree $2d$; if the degree keeps growing in this way, it will be too high to admit interpolation given only $n - t$ points (which is necessary if we would like to withstand fail-stop corruptions). Additional work needs to be done to reduce the degree of this polynomial: $2d + 1$ parties P_i need to reshare their points $f'_{xy}(i)$ using a new d-degree polynomial (that is, party P_i will pick a random polynomial r_i of degree t such that $r_i(0) = f'_{xy}(i)$ and send $r_i(j)$ to all other parties P_j). Each party P_j can locally compute $f_{xy}(j) = \sum_{i \in \mathcal{R}} \lambda_i r_i(j)$ where the λ_i's are the appropriate Lagrange coefficients.

Threshold Requirements Now, consider the (t, h^*)-FaF setting. In order to withstand fail-stop corruptions, even during degree reduction, we need $2d < n - t$. In order to have FaF security, we need $t + h^* \leq d$. We thus need

$$t + h^* < \frac{n - t}{2}$$

$$\Rightarrow 3t + 2h^* < n,$$

which is not optimal.

Brief Overview of BGW With Preprocessing

In order to approach the optimal threshold, we change the way we do multiplication to rely on *Beaver triple pre-processing* [Bea92]. Since this is work that explores the feasibility of FaF we can assume that the Beaver triples are generated by a trusted third party. We leave open the question of how such triples can be generated with security against a FaF adversary, eliminating the need for setup by a trusted third party. We give each party shares $f_a(i), f_b(i), f_c(i)$ of a randomly chosen a and b, and of their product $c = ab$. We use these shares in order to multiply x and y as follows. Each party P_i computes $f_\delta(i) = f_x(i) - f_a(i)$ and $f_\epsilon(i) = f_y(i) - f_b(i)$ and broadcasts these values. All parties reconstruct $\delta = x - a$ and $\epsilon = y - b$, and compute $f_{xy}(i) = f_c(i) + \epsilon f_x(i) + \delta f_y(i) - \delta\epsilon$.

We can see that f_{xy} has the same degree d, and that $f_{xy}(0) = c + \epsilon x + \delta y - \delta\epsilon = c + (y-b)x + (x-a)y - (x-a)(y-b) = c + xy - xb + yx - ya - xy + xb + ya - ab = xy$, because $c = ab$.

Threshold Requirements As in Section 7.6.1, in order to have FaF security, we need $t + h^* \leq d$. However, in order to withstand fail-stop corruptions, now we only need $d < n - t$. Putting these together, we need $t + h^* < n - t \Rightarrow 2t + h^* < n$, which is optimal.

Adaptive Security of BGW With Preprocessing

We now prove that BGW with Preprocessing with $2t + h^* < n$ and with $d = t + h^*$ is adaptively secure.

Theorem 7.8. *The construction summarized in Section 7.6.1 with $2t + h^* < n$ and with $d = t + h^*$ achieves security with guaranteed output delivery against an adaptive adversary with a budget of t fail-stop corruptions and h^* passive corruptions.*

We follow the blueprint of Damgård and Nielsen [DN14] for proving the adaptive security of BGW. We start by describing a simulator $\mathcal{S}_{static}$ for a static adversary, who is given the inputs of the passive and the fail-stop corrupted parties (which are similar to passive corruptions except that the adversary can choose to abort them at any step in the real world and substitute their input with a default input in the ideal world). $\mathcal{S}_{static}$ interacts with the adversary on behalf of the set $\mathcal{H}$ of the $n - t - h^*$ honest parties. She shares the input 0 on behalf of honest parties $P_i \in \mathcal{H}$. If a fail-stop party P_i fails to share an input, that party implicitly gives the degree 0 sharing of 0, where every share is 0 (thereby, its default input can be considered as 0). $\mathcal{S}_{static}$ forwards these inputs to the ideal functionality to obtain the output. $\mathcal{S}_{static}$ follows the protocol on behalf of the honest parties up until it's time to reconstruct the output. When it's time to reconstruct the output, $\mathcal{S}_{static}$ computes the difference $\delta = z - z'$, where z' is the computed output (shared on the polynomial $f_{z'}$), and z is the output dictated by the ideal functionality. $\mathcal{S}_{static}$ chooses a random polynomial Δ of degree d such that $\Delta(0) = \delta$ and $\Delta(i) = 0$ for all $i \in \mathcal{I}$. Notice that $f_{z'} + \Delta$ is a sharing f_z of the desired output $z = z' + \delta$. $\mathcal{S}_{static}$ uses shares $f_z(i)$ on behalf of honest parties $P_i \in \mathcal{H}$. (We make use of an assumption that the output z is produced by a multiplication gate. This forces the polynomial f_z used for the output to be a random degree d polynomial with the only constraint being that $f_z(0) = z$.)

We now describe the simulator $\mathcal{S}$ for an adaptive adversary. $\mathcal{S}$ starts out much like $\mathcal{S}_{static}$, by interacting with the adversary on behalf of the initial set $\mathcal{H}$ of honest parties, and maintaining a record of their views (including their shares of all intermediate values). However, unlike $\mathcal{S}_{static}$, at any point $\mathcal{S}$ may be asked to explain the view of one of these honest parties P_i, in the event that P_i becomes corrupt. It is insufficient for $\mathcal{S}$ to hand over the current simulated view of P_i, since $\mathcal{S}$ used the input 0 on behalf of P_i; this makes the simulated view clearly distinguishable from the real view, where the real input would have been used. So, $\mathcal{S}$ must adjusts the shares she has stored to account for the

use of the real input. Upon the corruption of party P_i, the simulator makes the following adjustments:

Shares of Input x_i: When party P_i is corrupted, $\mathcal{S}$ learns the real input x_i. Let $\mathcal{H}$ be the set of parties who were honest before the corruption of party P_i, and $\mathcal{I}$ be the set of parties who were corrupt. Note that $|\mathcal{I}| < t + h^* = d$; so, the views of the parties in $\mathcal{I}$ contain no information about $\{f_{x_i}(j)\}_{j\in\mathcal{H}}$, even if she knew x_i. $\mathcal{S}$ cannot change the shares of parties in $\mathcal{I}$, so she picks a random difference polynomial Δ s.t. $\Delta(0) = x_i$ and $\Delta(i) = 0$ for $i \in \mathcal{I}$. $\mathcal{S}$ updates the sharing polynomial as $f_{x_i} := f_{x_i} + \Delta$.

Addition or Multiplication by a Constant: In this case we only have local computation, so $\mathcal{S}$ simply recomputes the shares of the honest parties that were affected by the change to f_{x_i}.

Multiplication: Recall that for the multiplication of x by y each party j has published her share of δ, which is $f_\delta(j) = f_x(j) - f_a(j)$, and her share ϵ, which is $f_\epsilon(j) = f_y(j) - f_b(j)$ (where a, b and c is the Beaver triple s.t. $c = ab$). We will consider only x, a and δ; the case for y, b and ϵ is analogous. Since the adversary already saw $\delta = x - a$, and x might have changed, $\mathcal{S}$ must adjust a accordingly. Let x_{old} be the old value of x (shared on $f_{x_{old}}$), and a_{old} be the old value of a (shared on $f_{a_{old}}$). $\mathcal{S}$ defines $a := a_{old} + (x - x_{old})$, and lets $f_a := f_{a_{old}} + f_x - f_{x_{old}}$. Observe that $\delta = x_{old} - a_{old} = x_{old} - (a - (x - x_{old})) = x_{old} - a + x - x_{old} = x - a$, as desired. The shares on f_a are now consistent with the shares of δ previously published.

Note that we have changed the values of a and b, but not the value of c; so, it may no longer be the case that $c = ab$. However, this is not a problem, since the view of the adversary has no information about c (as she has insufficient shares).

Output Reconstruction: For an output z, the adversary has already seen all of the points on f_z. So, now we need to fix P_i's view to be consistent with $f_z(i)$. Recall that we assume that the output is produced by a multiplication gate, which uses a Beaver triple a, b, c. Let c_{old} be the old value of c (shared on $f_{c_{old}}$). Let a (shared on f_a) and b (shared on f_b) be the rest of that Beaver triple, and let x (shared on f_x) and y (shared on f_y) be the two inputs to the multiplication gate. Recall that $\delta = x - a$ and $\epsilon = y - b$ are fixed. $\mathcal{S}$ defines $c := z - \Big(\epsilon x + \delta y - \delta\epsilon\Big)$, and $f_c := f_z - (\epsilon f_x + \delta f_y - \delta\epsilon)$. This is consistent with what the adversary has seen.

As before, it may no longer be the case that $c = ab$; however, this is not a problem, since the adversary will have seen too few shares to be able to tell.

7.6.2. Adaptive BGW Against Mixed (Active / Passive) Adversaries

We now discuss how the security of the construction in Section 7.6.1 can be boosted to achieve guaranteed output delivery against a mixed adaptive adversary controlling t parties *actively* and h^* parties passively. This can be done by using the generic compiler

of Canetti *et al.* [CLOS02] that transforms a protocol secure against adaptive fail-stop corruptions to a protocol secure against adaptive active corruptions. At a high-level, this compiler follows the GMW compiler paradigm [GMW87a] where the parties (a) run an augmented coin-tossing protocol to obtain their respective uniformly distributed random tapes and commitments to other parties' random tapes, (b) commit to their inputs, (c) run the underlying fail-stop adaptively secure MPC protocol, while proving in each round using zero-knowledge that the computations have been done correctly. We can use the same compiler to upgrade security of the construction in Section 7.6.1 (achieving adaptive security against t fail-stop corruptions and h^* passive corruptions) to adaptive security against t active corruptions and h^* passive corruptions with a minor simplification. Since our underlying protocol satisfies perfect correctness (i.e. the protocol results in the correct output when everyone executes the protocol steps honestly, irrespective of the choice of random tapes of the parties), the augmented coin-tossing protocol used to determine the random tapes of the parties in the compiler of Canetti *et al.* can be avoided. The rest of the compiler remains the same; it relies on adaptively secure commitment and zero-knowledge tools (which can be based on enhanced trapdoor permutations). Since FaF security becomes relevant only in settings with more than 3 parties, the standard security notion for 2-party NIZK used as a building block would not affect FaF security of the overall protocol.

We argue that this simplified compiler preserves guaranteed output delivery. This is because, whenever an actively corrupt party misbehaves in the compiled protocol (for instance, a party aborts or the zero-knowledge proof showing the correctness of her actions in round r of the underlying protocol fails), such a scenario can be translated to an analogous scenario in the underlying protocol where the same party is fail-stop corrupt and stops communicating in round r. It is now easy to see that since the underlying protocol achieved GOD against t fail-stop and h^* passive corruptions, the same guarantees must hold against t active and h^* passive corruptions in the compiled protocol as well.

We state the formal theorem below.

Theorem 7.9. *The construction summarized in Section 7.6.2 with $2t + h^* < n$ and with $d = t + h^*$ achieves security with guaranteed output delivery against an adaptive adversary with a budget of t active corruptions and h^* passive corruptions.*

Part III.

Improving Efficiency for MPC

8. Compressing Pseudorandom Permutation Correlations

8.1. Introduction

Modern secure multi-party computation protocols are split into two phases: a function-independent offline phase and a function-dependent online phase. In the offline phase, parties generate preprocessing material, taking the form of correlated randomness, which they store for use in the online phase. Then, in the online phase, the parties use the generated correlated randomness to aid in the computation of the function. This model improves the efficiency of the online phase (e.g., by minimizing communication and rounds of interaction between parties) at the cost of an expensive preprocessing phase.

Starting with the work of Boyle et al. [BCGI18; BCGIKS20b; BCGIKS19], the efficiency of the offline phase—namely the generation of correlated randomness—was improved significantly with the help of *pseudorandom correlation generators* (PCGs) and *pseudorandom correlation functions* (PCFs) [BCGIKS20a],[1] which allow parties to locally expand a short seed into a large amount of correlated *pseudo*randomness. This paradigm enables efficient deployments of multi-party computation protocols in the preprocessing model, by significantly reducing the availability and interaction overheads incurred on the parties to generate correlated randomness.

In recent years, many constructions of PCGs and PCFs have been introduced, for various types of useful pseudorandom correlations such as oblivious transfer (OT) correlations [BCGIKS19; BCGIKRS19] and Beaver triple correlations [BCGIKS20b; BCCD23]. On the high end, there are generic approaches to obtaining PCGs and PCFs for large classes of correlations based on multi-key FHE or homomorphic secret sharing for circuits [DHRW16; BCGIKS19], and even for all efficiently computable correlations by relying on indistinguishability obfuscation [DHRW16; ASY22], but these approaches do not result in concretely efficient constructions. On the low end, constructions from variants of the Learning Parity with Noise (LPN) assumption and group-based assumptions give concretely efficient PCGs (and even PCFs [OSY21; BCGIKS20a; BCGIKRS22]) but only support limited correlations, e.g., typically OT and/or Beaver triple correlations.

The additive barrier. Most known constructions of PCGs and PCFs are limited to *additive* sharings of the target correlations. That is, each party obtains an additive secret share of the pseudorandom target correlation. In particular, this barrier has prevented

[1] Roughly speaking, a PCF can be used to generate a fresh sample from the target correlation "on-the-fly" via an evaluation algorithm whereas PCGs require expanding all correlations in a single monolithic evaluation.

	Assumption	Target Correlation	# parties
PCF [DHRW16]	$i\mathcal{O}$	Any reverse-samplable	2
PCF [ASY22]	multi-key FHE + $i\mathcal{O}$	Any reverse-samplable	Any
PCF [ASY22]	multi-key FHE + $i\mathcal{O}$ + ROM	Any	Any
PCG (Section 8.4)	Quasi-Abelian SD	Permutations (biased)	3
PCF (Section 8.4)	HSS+PRF in NC^1	Permutations	3

Table 8.1.: Constructions of PCGs and PCFs for non-additive correlations. Reverse-samplable means that, given the corrupt parties' outputs of a correlation sample, the honest parties' outputs can be efficiently simulated in a way that is indistinguishable from the original sample.

realizing concretely efficient PCGs for many interesting correlations that cannot be inherently represented by an additive secret sharing of the target correlations. This includes the case where parties need to obtain a pseudorandom permutation correlation or Shamir secret shares of some target correlation. While a very natural requirement, even high-end primitives such as multi-key FHE cannot be adapted to produce such correlations. Indeed, the *only* known way of achieving such "non-additive" correlations requires using indistinguishability obfuscation ($i\mathcal{O}$) in conjunction with other assumptions [DHRW16; ASY22].

Our contributions. In this paper, we initiate the study of PCGs and PCFs for the pseudorandom permutation correlation.

With an n-party pseudorandom permutation correlation, each party can locally obtain a component of a pseudorandom permutation over the set $\{1, \ldots, n\}$, while guaranteeing the full permutation remains hidden to any coalition of $n-2$ colluding parties. A PCG or PCF for pseudorandom permutations allows generating a large batch of ℓ such correlations with low communication complexity, e.g., $\mathsf{polylog}(\ell)$ following a one-time setup.

We construct three-party ($n = 3$) PCGs and PCFs from assumptions that are not known to imply $i\mathcal{O}$. In particular, we show how to construct a PCG for permutations from the Quasi-Abelian Syndrome Decoding assumption (a standard variant of the LPN assumption) and a PCF for permutations from Homomorphic Secret Sharing (HSS) [BGI16a; BCGIO17] for branching programs in conjunction with a PRF in NC^1, which can be instantiated under DCR [OSY21; RS21], LWE [BKS19] or assumptions in class groups [ADOS22]. While our HSS-based approach is still primarily of theoretical interest, our LPN-based construction results in a concretely-practical PCG, which we benchmark with an implementation.

We summarize our results in Table 8.1 and compare with prior approaches.

Applications. Our results carry an important conceptual contribution: They demonstrate the first feasibility result (without resorting to $i\mathcal{O}$) for constructing PCGs/PCFs for non-additive, reverse-sampleable correlations. We hope that some of the ideas developped in this work will pave the way to future developments on this challenging goal. Beyond

their conceptual benefits, our constructions have concrete applications in anonymous communication protocols and single secret leader election. In particular, we find that we can replace the expensive "offline" preprocessing protocol of Studholme and Blake [SB07] (which is used to generate pseudorandom permutations) with a PCG, to instantiate a Dining-Cryptography network for anonymous broadcast with optimal communication overheads. In particular, we show that using a PCG, each party can locally determine where to write their message in a bulletin board, in each round. Here, the pseudorandomness of the permutation guarantees that no other party learns the association between messages and the other parties and the permutation itself guarantees each party writes to a uniquely assigned "slot" removing the need for redundancy (a common solution used in prior work [CBM15; GJ04]). We also show how a PCF for pseudorandom permutations enable non-interactive single secret leader election (SSLE) protocol, where the leader is elected by having the lowest permutation value. In turn, SSLE has applications to proof-of-stake cryptocurrencies and other problems in distributed systems [BEHG20].

8.2. Technical Overview

In this section, we provide a detailed technical overview of our approach and construction. In Section 8.2.1, we briefly overview the syntax of PCGs. In Section 8.2.2, we overview the main ideas and general approach behind our constructions. Then, in Sections 8.2.3 and 8.2.4, we overview our PCG and PCF constructions.

8.2.1. Background

Here we give a brief overview of PCGs and PCFs to aid in understanding the technical overivew. See Section 8.3 for formal definitions.

A PCG for a target two-party correlation $\mathcal{C}$ consists of two algorithms:

- $\mathsf{Gen}(1^n) \to (\mathsf{k}_0, \mathsf{k}_1)$: a randomized algorithm that generates a pair of short, correlated seeds $(\mathsf{k}_0, \mathsf{k}_1)$ given a security parameter n, and
- $\mathsf{Expand}(\mathsf{k}_\sigma) \to R_\sigma$: a deterministic algorithm which stretches a seed k_σ into a long output R_σ.

In the case of a PCF, Expand is replaced with an evaluation algorithm Eval that takes an additional input x and outputs a single instance of the correlation (computed as a function of x).

The security requirement for a PCG is that the joint outputs (R_0, R_1) should be indistinguishable from $\mathcal{C}$, not only to external parties but also to each of the parties that possesses only one of the seeds. PCFs have an analogous security requirement with the main difference being on-demand evaluation of R_0 and R_1.

8.2.2. Main ideas and approach

All our constructions are limited to the three-party setting and we call these parties Alice, Bob, and Carol. Both our PCG and PCF construction share a common template,

which is refined from a natural but failed strawman approach.

The strawman approach. Examining a three party permutation, we first observe that we can view the problem as generating additive sharing of the "zero correlation" (modulo 3), where the parties always obtain additive shares of zero. To see this, note that a permutation over the set $\{0, 1, 2\}$ can viewed as giving parties pseudorandom shares of zero, since the sum of the permutation values (when parsed as additive shares) is always 3, which is equivalent to zero in $\mathbb{F}_3$. Unfortunately, while all pseudorandom permutations can be viewed as shares of zero, not all pseudorandom shares of zero form a valid pseudorandom permutation. Specifically, in the three-party case, all shares of zero that consist of the same value in $\mathbb{F}_3$, result in parties having shares of zero but an invalid permutation (since all parties have the same output). Therefore, the problem of constructing a PCG for permutations (in the three party case) boils down to ensuring that all three "pseudorandom shares of zero" held by the parties are distinct.[2] However, eliminating these "bad" cases turns out to be non-trivial.

Guaranteeing distinct shares. Our first observation is that for just two parties, say Alice and Bob, we can guarantee distinct shares by having the parties generate shares of a pseudorandom *bit* $\mu \in \{0, 1\}$ over $\mathbb{F}_3$. Then, using shares of μ we can guarantee *distinct shares* between them as follows. Let $a \in \mathbb{F}_3$ be Alice's share of μ and $b \in \mathbb{F}_3$ be Bob's share of μ, such that $a + b \pmod 3 = \mu$. If Alice outputs a and Bob outputs $2 - b$, as their components of the pseudorandom permutation, then it is clear that the outputs of Alice and Bob are trivially distinct as $2 - b = a + 2 - \mu \pmod 3$. Furthermore, Alice cannot distinguish whether Bob outputs $a + 1$ or $a + 2$, ensuring privacy of the permutation so far. However, how can Carol get her share of the permutation? Ideally, we'd like to repeat the same process between Alice and Carol, such that Carol's output is distinct to both Bob's and Alice's output, and Alice's output/share stays the same. However, this presents a challenge: Carol's share must somehow correlate with Bob's share to ensure they also have distinct outputs *and* Alice's share needs to remain the same. Overcoming this turns out to be the crux of our solution to compressing pseudorandom permutations in the three party setting, and we solve this challenge in two different ways to arrive at our PCG and PCF constructions for permutations (overviewed in Sections 8.2.3 and 8.2.4, respectively). However, ignoring this problem for the time being, we first explain how this gives us a template for a three-party construction.

The general template. Notice that if Alice and Bob, and Alice and Carol, can each generate a pseudorandom *subtractive* sharing of a bit μ and of $1 - \mu$, respectively, such that Alice maintains the same share a in both cases, then Carol's share c will be such that $2 - c \pmod 3$ is guaranteed to be distinct from both Alice's and Bob's output.

[2] It suffices to ensure that the parties have shares $a + b + c = 0$ and $a \neq b$. This implies a, b and c are pairwise distinct, since if e.g., c was equal to b then we have $a = -2b = b \pmod 3$, a contradiction.

Specifically, if we expand the outputs of each party, we have that:

Alice outputs:	$a := x$
Bob outputs:	$b := x + 2 - \mu$
Carol outputs:	$c := x + 1 + \mu$

Note that a, b, c are pairwise distinct. This is because $a = b$ would imply $\mu = 2$; $a = c$ would imply $\mu + 1 = 0$; and $b = c$ would imply $2\mu = 1 \pmod 3$. All these statements are false for $\mu \in \{0, 1\}$. More precisely, Bob and Carol must be given the ability to generate shares of pseudorandom bits μ and of $1 - \mu$, respectively.

In summary, this brings us to a sufficient condition for compressing a pseudorandom permutation: Generating pseudorandom shares of a bit $\mu \in \mathbb{F}_3$ and pseudorandom shares of $1 - \mu$ while fixing one of the shares to be the same in both cases. Unfortunately, while this requirement may sound trivial to achieve, in reality it requires overcoming several roadblocks. The first challenge, as alluded to above, is that we need to ensure that Alice's share is identical for both the shares of μ and the shares of $1 - \mu$. The second challenge is that we need to generate this bit in a non-binary field $\mathbb{F}_3$, and efficient protocols for computing bits in such fields typically require interaction [IKNZ23]. We now explain these roadblocks in more detail and highlight the strategies we use to overcome them.

Challenges in fixing Alice's share. Ensuring that Alice's share is the same when generating pseudorandom shares of μ and $1 - \mu$ with Bob and Carol requires a notion of "programmability." Prior notions of programmable PCG [BCGIKS19; BCGIKS20b] (resp. PCF) for correlations over $\mathbb{F}_3$ could be used to guarantee that the pseudorandom bit μ is the same across both the share of μ and $1 - \mu$. However, even with a programmable PCG (resp. PCF), Alice will obtain two distinct and independent shares, one for the μ instance and one for the $1 - \mu$ instance, which prevents us from applying the template we described above. To resolve this, we introduce the notion of a "doubly-programmable" PCG (resp. PCF).

New tool: "Doubly-programmable" PCGs and PCFs. We observe that existing constructions of programmable PCGs (resp. PCFs) can be adapted to provide "one-more-level" of programmability by making the seed (resp. key) of one party sampled independently of the correlation. This makes it possible to fix a seed/key k_0 and then generate multiple keys $\mathsf{k}_1, \mathsf{k}_2$ for two different programmable correlations $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. More concretely, coming back to our use case, Alice can be given k_0 which allows her to generate a pseudorandom share x while Bob and Carol can be given k_1 and k_2, respectively, such that:

$$\mathsf{Expand}(\mathsf{k}_0) - \mathsf{Expand}(\mathsf{k}_1) = \mu \text{ and } \mathsf{Expand}(\mathsf{k}_0) - \mathsf{Expand}(\mathsf{k}_2) = 1 - \mu.$$

Formally, we define such "doubly-programmable" PCGs and PCFs as being $\mathcal{F}$-programmable, where (1) one of the seeds/keys can be sampled independently of the ultimate correlation and (2) the correlation itself can be programmed to be some function of the source of pseudorandomness. (In the above example, the programmed function in k_2 maps x to $1 - x$.)

Challenges in generating bits over a field. Generating a pseudorandom bit in a larger field has many applications beyond our use case for building a PCG for pseudorandom permutations. In particular, such a correlation is very useful for efficient multi-party computation protocols [IKNZ23; CS10; DFKNT06] as it enables efficient conversion of secret shares from one field to another. However, despite this usefulness, there are no constructions PCGs/PCFs for this target correlation. Moreover, we observe that such a correlation can only be computed by high degree polynomial, making it difficult to instantiate using known techniques, which are typically geared to degree-2 correlations.

To overcome this challenge, we show that in the case of $\mathbb{F}_3$, it is possible to generate shares of a pseudorandom bit by evaluating a degree-2 polynomial, provided we settle for a *biased* bit in $\mathbb{F}_3$. This observation forms the backbone of our PCG construction, which we show can be instantiated by building off of existing PCGs for degree-2 correlations in $\mathbb{F}_3$.

To realize a PCF for pseudorandom permutations, we resort to using homomorphic secret sharing (HSS) [BGI16a] to evaluate a high-degree polynomial computing a PRF outputting a single bit. While we can concretely reduce the degree of the evaluation function by using tailored low-degree PRFs, our approach to constructing PCFs for pseudorandom permutations does not fully get around the barrier described above.

8.2.3. Overview of our PCG construction

We now turn to explaining how we construct our PCG for permutations by constructing an $\mathcal{F}$-programmable PCG for pseudorandom bits.

Our starting point is existing PCG constructions for degree-2 correlations in $\mathbb{F}_3$, which were recently constructed by Bombar et al.[BCCD23; BBCCDS24a] from the Quasi-Abelian Syndrome Decoding (QASD) assumption and allow programming the output correlation. In particular, we observe that this PCG construction can be upgraded using a *programmable* distributed point function (DPF) [BGIK22] to realize an $\mathcal{F}$-programmable PCG for correlations in $\mathbb{F}_3$. In a nutshell, this follows directly from an observation made by Boyle et al. [BGIK22], where they show that existing PCGs based on DPFs (which includes the PCG of Bombar et al.[BBCCDS24a]) can be converted using a programmable DPF such that the PCG seed k_0 of one party can be generated independently of the second seed k_1. Moreover, the seed k_0 is independent of the computed correlation, which gives us all the necessary properties to obtain $\mathcal{F}$-programmablilty.

From squares to bits. Using the $\mathcal{F}$-programmable PCG for correlations in $\mathbb{F}_3$, we show how to convert a pseudorandom degree-2 correlation into a biased pseudorandom bit in $\mathbb{F}_3$. The main idea is to define a share of μ as a share of $z^2 \pmod 3$, for a pseudorandom $z \in \mathbb{F}_3$. In particular, a share of $z^2 \pmod 3$, is always in the set $\{0, 1\}$, since $2^2 \pmod 3 = 1$. However, the value of z^2 is biased towards 1. The reason for the bias is that z^2 is not evenly distributed over $\{0, 1\}$ but is instead uniformly distributed over the *multiset* $\{0, 1, 1\}$ given that $2^2 = 1 \pmod 3$. This bias also translates to the resulting permutation, making the resulting PCG share the same bias. Nonetheless, this still represents a non-trivial PCG for permutations, which we show can be used in applications of multi-round anonymous broadcast when coupled with a suitable randomness extractor.

Putting things together. In summary, we show that we can extend the PCG of Bombar et al.[BBCCDS24a] for degree-2 correlations over $\mathbb{F}_3$ to be $\mathcal{F}$-programmable, which then allows us to generate three PCG keys k_0, k_1, k_2, which are distributed to Alice, Bob, and Carol respectively, such that Alice and Bob obtain shares of a pseudorandom "ϵ-biased" bit μ and Alice and Carol obtain shares of $1 - \mu$ (or vice versa), while keeping Alice's share the same in both cases. This then allows us to construct an "ϵ-biased" PCG for permutations with $\epsilon = (1/6)$ bias. We describe this construction in Section 8.4.

8.2.4. Overview of our PCF construction

Now that we've described the PCG construction, we take a step back and observe that if we instead have an $\mathcal{F}$-programmable PCF for generating a pseudorandom bit in $\mathbb{F}_3$, then we'd be able to avoid the bias in the construction from Section 8.2.3 and instantiate the template directly. Specifically, we observe that homomorphic secret sharing (HSS) coupled with a PRF in NC^1 gives us a programmable PCF for a pseudorandom bit in $\mathbb{F}_3$. Then, to upgrade this PCF to be $\mathcal{F}$-programmable, we use the recent paradigm of Couteau et al. [CMPR23] of "HSS with simulatable memory shares" which, informally speaking, allows one of the parties to compute their memory share independently of the function and immediately gives us $\mathcal{F}$-programmability. Coupled with a PRF in NC^1, we get a mechanism for constructing a PCF for (unbiased) pseudorandom bits in $\mathbb{F}_3$. We also show that if we assume specific low-degree PRF candidate, then we can significantly reduce the number of HSS multiplications required, making the PCF closer to being concretely practical. As an application of our PCF construction, we show an immediate application to single secret leader election, where the party with the lowest permutation value is elected as the leader without revealing themselves to the other (possibly corrupted) parties.

8.3. Preliminaries

Notation. We let $\mathbb{N}$ denote the set of natural numbers. We denote a vector $\mathbf{u}$ using bold lowercase letters and let $\mathbf{u}[i]$ denote the i-th coordinate of $\mathbf{u}$. We denote by $\mathsf{poly}[](\cdot)$ the set of all polynomials and by $\mathsf{negl}[](\cdot)$ any negligible function. We let $x \xleftarrow{R} S$ denote a uniformly random sample drawn from S. We let $x \leftarrow \mathsf{Adv}$ denote assignment from a (possibly randomized) algorithm Adv and $x := y$ denote initialization of x to the value of y. By an *efficient* algorithm Adv we mean that Adv is modeled by a (possibly non-uniform) Turing Machine that runs in probabilistic polynomial time. We write $D_0 \approx_c D_1$ to mean that two distributions D_0 and D_1 are *computationally* indistinguishable to all efficient distinguishers Adv and $D_0 \approx_s D_1$ to mean that D_0 and D_1 are *statistically* indistinguishable.

8.3.1. Homomorphic Secret Sharing

Homomorphic Secret Sharing (HSS) [BGI16a] can be seen as a distributed analogue to homomorphic encryption. In contrast to the centralized model where a single server

computes on encrypted inputs, in HSS, shares of the input are distributed among multiple parties, who then compute a share of the output through local homomormpic computations.

Definition 8.1 (Homomorphic Secret Sharing [BGI16a])**.** Let n be a security parameter and $\mathcal{R}$ be a finite ring. A Homomorphic Secret Sharing (HSS) scheme for a class of programs $\mathcal{P}$ with input space $\mathcal{I} \subseteq \mathcal{R}$ and share space $\mathcal{S} \subseteq \mathcal{R}$, consists of three efficient (possibly randomized) algorithms $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ such that:

- $\mathsf{Setup}(1^n) \to (\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B))$. Takes as input the security parameter n. Outputs a public key pk and a pair of evaluation keys $(\mathsf{ek}^A, \mathsf{ek}^B)$.
- $\mathsf{Input}(\mathsf{pk}, x) \to (\llbracket x \rrbracket^A, \llbracket x \rrbracket^B)$. Takes as input the public key pk and an input $x \in \mathcal{I}$. Outputs a pair of HSS input shares $(\llbracket x \rrbracket^A, \llbracket x \rrbracket^B)$.
- $\mathsf{Eval}(\sigma, \mathsf{ek}^\sigma, (\llbracket x_1 \rrbracket^\sigma, \ldots, \llbracket x_n \rrbracket^\sigma), P) \to \langle y \rangle^\sigma$. Takes as input the party identifier σ, an evaluation key ek^σ, a vector of n HSS input shares, and an n-input program $P \in \mathcal{P}$. Outputs a subtractive share of the output $y = P(x_1, \ldots, x_n)$ in share space $\mathcal{S}$.

The above functionality must satisfy the following properties:

Correctness. For all security parameters $n \in \mathbb{N}$, any $n \in \mathbb{N}$, and any n-input program $P \in \mathcal{P}$ with input space $\mathcal{I} \subseteq \mathcal{R}$, it holds that for any $x_1, \ldots, x_n \in \mathcal{I}$:

$$\Pr\left[\begin{array}{l} y = P(x_1, \ldots, x_n) : \\ \end{array} \begin{array}{r} (\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B)) \leftarrow \mathsf{Setup}(1^n) \\ (\llbracket x_i \rrbracket^A, \llbracket x_i \rrbracket^B) \leftarrow \mathsf{Input}(\mathsf{pk}, x_i) \forall i \in [n] \\ \langle y \rangle^A \leftarrow \mathsf{Eval}(A, \mathsf{ek}^A, (\llbracket x_i \rrbracket^A)_{i \in [n]}, P) \\ \langle y \rangle^B \leftarrow \mathsf{Eval}(B, \mathsf{ek}^B, (\llbracket x_i \rrbracket^B)_{i \in [n]}, P) \\ y \leftarrow \langle y \rangle^A - \langle y \rangle^B \end{array}\right] \geq 1 - \mathsf{negl}.$$

Security. For all efficient adversaries Adv, and any $\sigma \in \{A, B\}$, there exists a negligible function $\mathsf{negl}[\cdot]$ such that:

$$\Pr\left[\begin{array}{l} b' = b : \\ \end{array} \begin{array}{r} (x_0, x_1, \mathsf{st}) \leftarrow \mathsf{Adv}(1^n) \\ (\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B)) \leftarrow \mathsf{Setup}(1^n) \\ b \xleftarrow{\mathrm{R}} \{0, 1\} \\ (\llbracket x_b \rrbracket^A, \llbracket x_b \rrbracket^B) \leftarrow \mathsf{Input}(\mathsf{pk}, x_b) \\ b' \leftarrow \mathsf{Adv}(\mathsf{st}, \mathsf{pk}, \mathsf{ek}^\sigma, \llbracket x_b \rrbracket^\sigma) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}.$$

HSS memory values. Let $\langle\!\langle x \rangle\!\rangle^\sigma$ denote an intermediate "memory share" of an HSS evaluation held by party σ. In existing HSS schemes (e.g., [BGI16a; CMPR23; DIJL23a; FGJS17]) memory shares can be added together and multiplied with an HSS input share to produce a new memory share. Also, there is additional efficient algorithm MemGen

that takes as input an index $\sigma \in [n]$, and evaluation key ek_σ and an input $x \in \mathcal{I}$, and outputs a memory share $\langle\langle x \rangle\rangle^\sigma$ of x.

Extended Evaluation. Moreover, HSS schemes support a so-called "extended evaluation", as given by [CMPR23. Lemma 1]. Informally, there is an additional evaluation algorithm $\mathsf{ExtEval}$ that is similar to Eval but takes one additional input $M_\sigma = \langle\langle z \rangle\rangle^\sigma$—a memory share of $z \in \mathcal{I}$—and outputs a $\langle y \rangle^\sigma$, such that upon reconstruction $\langle y \rangle^0 - \langle y \rangle^1$ one obtains the z-multiplied output $z \cdot P(x_1, \ldots, x_n)$. We will make use of this multiply our outputs by a $z \in \{-1, 1\}$.

More formally, there the syntax of $\mathsf{ExtEval}$ is as follows:

- $\mathsf{ExtEval}(\sigma, \mathsf{ek}^\sigma, \langle\langle \alpha \rangle\rangle^\sigma, (\llbracket x_1 \rrbracket^\sigma, \ldots, \llbracket x_n \rrbracket^\sigma), P) \to \langle y \rangle^\sigma$. Takes as input an index σ, an evaluation key ek^σ, a memory share $\langle\langle \alpha \rangle\rangle^\sigma$ of $\alpha \in \mathcal{I}$ and a vector of n HSS input shares, and an n-input program $P \in \mathcal{P}$. Outputs a subtractive share of the output $y = \alpha P(x_1, \ldots, x_n)$.

Correctness of this algorithm means that

$$\Pr\left[\begin{array}{r} (\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B)) \leftarrow \mathsf{Setup}(1^n) \\ (\llbracket x_i \rrbracket^A, \llbracket x_i \rrbracket^B) \leftarrow \mathsf{Input}(\mathsf{pk}, x_i) \forall i \in [n] \\ M^A \leftarrow \mathsf{MemGen}(A, \mathsf{ek}^A, z) \\ M^B \leftarrow \mathsf{MemGen}(B, \mathsf{ek}^B, z) \\ \langle y \rangle^A \leftarrow \mathsf{ExtEval}(A, \mathsf{ek}^A, M^A, (\llbracket x_i \rrbracket^A)_{i \in [n]}, P) \\ \langle y \rangle^B \leftarrow \mathsf{ExtEval}(B, \mathsf{ek}^B, M^B, (\llbracket x_i \rrbracket^B)_{i \in [n]}, P) \\ y \leftarrow \langle y \rangle^A - \langle y \rangle^B \end{array} : y = z \cdot P(x_1, \ldots, x_n)\right] \geq 1 - \mathsf{negl},$$

i.e., if we run it like Eval, but with the additionally generated memory share of a $z \in \mathcal{I}$, we get a subtractive share of $z \cdot P(x_1, \ldots, x_n)$.

Programmability of HSS memory values. Here, we follow the definition of [CMPR23. Sect. 4.3].

Definition 8.2 (HSS with Simulatable Memory Values)**.** Let $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ be an HSS scheme with input space $\mathcal{I}$. Then, HSS is *simulatable w.r.t. its memory values* if there are efficient algorithms Sim_0, Sim_1, where Sim_0 takes as input 1^n and outputs a memory value M_0, and Sim_1 takes as input a memory value M_0, an element $z \in \mathcal{I}$, and two keys $\mathsf{ek}^A, \mathsf{ek}^B$, and outputs a memory value M_1. Then we want:

Simulation Correctness. For any security parameter n, and any $z_1, z_2 \in \mathcal{I}$, we want that correctness w.r.t. extended evaluation still holds when the memory values are

simulated, i.e.

$$\Pr\left[\begin{array}{l} y = z \cdot P(x_1, \ldots, x_n) : \end{array} \begin{array}{r} (\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B)) \leftarrow \mathsf{Setup}(1^n) \\ (\llbracket x_i \rrbracket^A, \llbracket x_i \rrbracket^B) \leftarrow \mathsf{Input}(\mathsf{pk}, x_i) \forall i \in [n] \\ M^A \leftarrow \mathsf{Sim}_0(1^n) \\ M^B \leftarrow \mathsf{Sim}_1(M^A, z, (\mathsf{ek}^A, \mathsf{ek}^B)) \\ \langle y \rangle^A \leftarrow \mathsf{ExtEval}(A, \mathsf{ek}^A, M^A, (\llbracket x_i \rrbracket^A)_{i \in [n]}, P) \\ \langle y \rangle^B \leftarrow \mathsf{ExtEval}(B, \mathsf{ek}^B, M^B, (\llbracket x_i \rrbracket^B)_{i \in [n]}, P) \\ y \leftarrow \langle y \rangle^A - \langle y \rangle^B \end{array}\right] \geq 1 - \mathsf{negl}.$$

Simulation Security. There exists an efficient simulator $\mathcal{S}$ such that for all $(\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B))$ in the image of Setup,

$$\left\{ M_1 \;\middle|\; \begin{array}{r} M_0 \leftarrow \mathsf{Sim}_0(1^n) \\ M_1 \leftarrow \mathsf{Sim}_1(M_0, z, (\mathsf{ek}^A, \mathsf{ek}^B)) \end{array} \right\} \approx_c \mathcal{S}(\mathsf{pk}).$$

8.3.2. Programmable Function Secret Sharing and Distributed Point Functions

Function secret sharing (FSS) [BGI15; BGI16b] is an analogue of HSS for sharing functions instead of data. Concretely, FSS allows to split up a function f into shares $\mathsf{k}_0, \mathsf{k}_1$, such that each share individually hides the function f. The shares can be locally evaluated on an input x, such that the evaluations of both shares at the same x form subtractive shares of $f(x)$.

Definition 8.3 (Function Secret Sharing)**.** Let $\mathcal{F} = \{f : I \to \mathbb{G}\}$ be a class of function descriptions, where the description of each f specifies the input domain I and an Abelian group $(\mathbb{G}, +)$ as the output domain. A (2-party) *function secret sharing* (FSS) scheme for $\mathcal{F}$ is a pair of algorithms $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ with the following syntax:

- $\mathsf{FSS.Gen}(1^n, f)$ is a PPT algorithm that given security parameter n and description of $f \in \mathcal{F}$ outputs a pair of keys (K_0, K_1). We assume that the keys specify I and $\mathbb{G}$.
- $\mathsf{FSS.Eval}(b, K_b, x)$ is a polynomial-time algorithm that, given a key K_b for party $b \in \{0, 1\}$, and an input $x \in I$, outputs a group element $y_b \in \mathbb{G}$.

The scheme should satisfy the following requirements:

Correctness. For any $f \in \mathcal{F}$ and $x \in I$, we have $\Pr[(K_0, K_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^n, f) : \mathsf{FSS.Eval}(0, K_0, x) - \mathsf{FSS.Eval}(1, K_1, x) = f(x)] = 1$.

Security. Let $f_1, f_2, \ldots$ be any polynomial-size sequence of functions $f_i \in \mathcal{F}$. For $b \in \{0, 1\}$, consider the following two experiments with a PPT simulator Sim:

- $\mathsf{Real}_b(1^n)$: Run $(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^n, f_n)$ and output k_b
- $\mathsf{Ideal}_b(1^n)$: Output $\mathsf{k}_b = \mathsf{Sim}(1^n, \mathsf{Leak}(f_n))$

We say that FSS is secure if for each $b \in \{0,1\}$, there exists Sim such that, for any non-uniform adversary Adv of size poly, $\Pr[\mathsf{Adv}(\mathsf{Real}_b(1^n)) = 1] - \Pr[\mathsf{Adv}(\mathsf{Ideal}_b(1^n)) = 1] \leq \mathsf{negl}$.

In the constructions we use, the leakage function $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$ is given by $\mathsf{Leak}(f_n) = (I, \mathbb{G})$, namely it outputs a description of the input and output domains of f.

Remark 5 (Pseudorandomness of FSS Outputs). In some of our constructions, we rely on a property that outputs of FSS.Eval are indistinguishable from random, when the FSS keys remain secret. This property can be obtained generically, by including a PRF key in each FSS key and having each party add the same PRF evaluation to its output. However, we note that the property is already achieved by all the FSS constructions we use.

Distributed Point Functions and FSS for Sums of Point Functions.

Let N be a domain size, $\mathbb{G}$ an abelian group, $\alpha \in [N]$, and $\beta \in \mathbb{G}$. A *point function* is a function $f_{\alpha,\beta} : [N] \to \mathbb{G}$ that evaluates to 0 on all inputs $x \neq \alpha$, and $f_{\alpha,\beta}(x) = \beta$ if $x = \alpha$.

An FSS scheme for the class of point functions of domain size N is known as a *distributed point function* [GI14]. As in previous work [BCGIKS20b; BCCD23], it will be convenient for our constructions to use FSS for the class of functions that can be described as a sum of t point functions.

Definition 8.4 (FSS for sum of point functions (SPFSS))**.** Let N be a domain size, $\mathbb{G}$ an abelian group, $\alpha \in [N]$, and $\beta \in \mathbb{G}$. A *point function* is a function $f_{\alpha,\beta} : [N] \to \mathbb{G}$ defined by $f_{\alpha,\beta}(x) = 0$ whenever $x \neq \alpha$, and $f_{\alpha,\beta}(x) = \beta$ if $x = \alpha$. For $S = (s_1, \ldots, s_t) \in [N]^t$ and $\mathbf{y} = (y_1, \ldots, y_t) \in \mathbb{G}^t$, the *sum of point functions* $f_{S,\mathbf{y}} : [N] \to \mathbb{G}$ is defined as

$$f_{S,\mathbf{y}}(x) = \sum_{i=1}^{t} f_{s_i,y_i}(x).$$

An *SPFSS* scheme is an FSS scheme for the class of sums of point functions.

Programmable FSS.

We use 2-party FSS schemes with a special *programmability* property, meaning that one of the two keys can be sampled independently of the function f being shared. The following definition extends the concept of programmable distributed point function [BGIK22] to programmable FSS for general functions.

Definition 8.5 (Programmable FSS)**.** We say that an FSS scheme (FSS.Gen, FSS.Eval) is *programmable* if FSS.Gen can be split into two algorithms Gen_0 and Gen_1 with the following syntax:

- $\mathsf{Gen}_0(1^n, I, \mathbb{G}) \to \mathsf{k}_0$. On input the security parameter and input/output domains of the function f_n, this samples a key k_0.
- $\mathsf{Gen}_1(\mathsf{k}_0, f_n) \to \mathsf{k}_1$. on input key k_0 and the function f_n, this generates a key k_1.

It is easy to see that a programmable DPF immediately also implies programmable FSS for sums of t point functions.

Boyle et al. [BGIK22] construct a programmable distributed point function for any domain size $N = N(n) \in \mathsf{poly}$ based on a pseudorandom generator. Their construction starts by building a simple and efficient, but *leaky*, programmable DPF, and then upgrades this to a fully secure construction through privacy amplification techniques. While the final construction is not practical, in Section 8.6 we show how to leverage the practical, leaky version of their DPF in our construction to build a concretely efficient $\mathcal{F}$-programmable PCG.

8.3.3. Pseudorandom Correlation Generators

A pseudorandom correlation generator, or PCG, is a way of distributing short seeds to a set of parties, which can later be expanded to produce a large amount of correlated randomness. The type of correlated randomness obtained is specified as a correlation generator, which typically has the reverse-samplable property below. In the following definitions, we adapt the definition of n-party PCG [BCGIKS19] to the case where only one party is corrupted. This fits the two-party and three-party constructions we present.

Definition 8.6 (Reverse-sampleable n-Correlation Generator)**.** A PPT algorithm $\mathcal{C}$ is called a *reverse sampleable n-correlation generator*, if on input 1^n it outputs elements in $\{0,1\}^\ell \times \ldots \times \{0,1\}^\ell \in (\{0,1\}^\ell)^n$ for some length parameter $\ell \in \mathsf{poly}$, and there exists a PPT algorithm $\mathsf{RSample}$ such that for any $\sigma \in [n]$, the correlation obtained via:

$$\left\{ (R'_1, \ldots, R'_n) \;\middle|\; \begin{array}{r} (R_1, \ldots, R_n) \xleftarrow{\$} \mathcal{C}(1^n) \\ R'_\sigma := R_\sigma, (R'_i)_{i \in [n] \setminus \{\sigma\}} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma) \end{array} \right\}$$

is computationally indistinguishable from $\mathcal{C}(1^n)$.

Definition 8.7 (Pseudorandom n-Correlation Generator)**.** Let $\mathcal{C}$ be a reverse-sampleable n-correlation generator. A *pseudorandom correlation generator (PCG)* for $\mathcal{C}$ is a pair of algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ with the following descriptions:

- $\mathsf{PCG.Gen}(1^n)$ is a PPT algorithm that given a security parameter n, outputs seeds $(\mathsf{k}_1, \ldots, \mathsf{k}_n)$.
- $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ is polynomial time algorithm that given a party index $\sigma \in [n]$ and a seed k_σ, outputs a bit string $R_\sigma \in \{0,1\}^\ell$.

The algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ should satisfy the following:

Correctness. The correlation obtained via:

$$\{(R_1, \ldots, R_n) | (\mathsf{k}_1, \ldots, \mathsf{k}_n) \xleftarrow{\$} \mathsf{PCG.Gen}(1^n), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma) \text{ for } \sigma \in [n]\}$$

is computationally indistinguishable from $\mathcal{C}(1^n)$

Security. For any $i \in [n]$, the following two distributions are computationally indistinguishable:

$$\left\{ (\mathsf{k}_i, (R_j)_{j \neq i}) \;\middle|\; \begin{array}{r} (\mathsf{k}_1, \ldots, \mathsf{k}_n) \xleftarrow{\$} \mathsf{PCG.Gen}(1^n) \\ R_j \leftarrow \mathsf{PCG.Expand}(j, \mathsf{k}_j) \;:\; j \neq i \end{array} \right\}$$

and

$$\left\{ (\mathsf{k}_i, (R_j)_{j \neq i}) \;\middle|\; \begin{array}{r} (\mathsf{k}_1, \ldots, \mathsf{k}_n) \xleftarrow{\$} \mathsf{PCG.Gen}(1^n) \\ R_i \leftarrow \mathsf{PCG.Expand}(i, \mathsf{k}_i) \\ (R_j)_{j \neq i} \xleftarrow{\$} \mathsf{RSample}(i, R_i) \end{array} \right\}$$

where $\mathsf{RSample}$ is the reverse sampling algorithm for the correlation $\mathcal{C}$.

8.3.4. Pseudorandom Correlation Functions

A PCF [BCGIKS20a] can be seen as a PCG where each output of the target correlation can be computed "on-demand", instead of being expanded all at once. Furthermore, as in a PRF, the number of possible outputs (that is, the domain size) may be exponential in the security parameter.

The syntax of a PCF consists of a key generation algorithm $\mathsf{PCF.KeyGen}$, which outputs a set of correlated keys, as well as a $\mathsf{PCF.Eval}$ algorithm that uses a key to compute a single PCF output.

The security game and the formal definition of a PCF follow.

$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathsf{Adv},N,0}(n)}$:	$\underline{\mathsf{Exp}^{\mathsf{pr}}_{\mathsf{Adv},N,1}(n)}$:
for $i = 1$ **to** $N(n)$:	$(\mathsf{k}_1, \ldots, \mathsf{k}_n) \leftarrow \mathsf{PCF.KeyGen}(1^n)$
$\quad x_i \xleftarrow{\mathrm{R}} \{0,1\}^\ell$	**for** $i = 1$ **to** $N(n)$:
$\quad (R_1^{(i)}, \ldots, R_n^{(i)}) \leftarrow \mathcal{C}(1^n)$	$\quad x_i \xleftarrow{\mathrm{R}} \{0,1\}^\ell$
$b' \leftarrow \mathsf{Adv}(1^n, (x_i, R_1^{(i)}, \ldots, R_n^{(i)})_{i \in [N(n)]})$	$\quad$ **for** $\sigma \in [n]$:
return b'	$\quad R_\sigma^{(i)} \leftarrow \mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x_i)$
	$b' \leftarrow \mathsf{Adv}(1^n, (x_i, R_1^{(i)}, \ldots, R_n^{(i)})_{i \in [N(n)]})$
	return b'

Figure 8.1.: Pseudorandom $\mathcal{C}$-correlated outputs of a PCF.

Definition 8.8 (Pseudorandom Correlation Function [BCGIKS20a])**.** Let n be a security parameter, $\mathcal{C}$ be a reverse-sampleable n-correlation with output length $\ell(n) \in \mathsf{poly}$,

$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathsf{Adv},N,\sigma,0}(n)}$:
$(\mathsf{k}_1,\ldots,\mathsf{k}_n) \leftarrow \mathsf{PCF.KeyGen}(1^n)$
for $i = 1$ **to** $N(n)$:
  $x_i \xleftarrow{\mathrm{R}} \{0,1\}^\ell$
  $R^{(i)}_\sigma \leftarrow \mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x_i)$
  $(R^{(i)}_{\overline{\sigma}})_{\overline{\sigma}\neq\sigma} \leftarrow \mathsf{RSample}(1^n, \sigma, R^{(i)}_\sigma)$
$b' \leftarrow \mathsf{Adv}(1^n, \sigma, \mathsf{k}_\sigma, (x_i, (R^{(i)}_{\overline{\sigma}})_{\overline{\sigma}\neq\sigma})_{i\in[N(n)]})$
return b'

$\underline{\mathsf{Exp}^{\mathsf{sec}}_{\mathsf{Adv},N,\sigma,1}(n)}$:
$(\mathsf{k}_1,\ldots,\mathsf{k}_n) \leftarrow \mathsf{PCF.KeyGen}(1^n)$
for $i = 1$ **to** $N(n)$:
  $x_i \xleftarrow{\mathrm{R}} \{0,1\}^\ell$
  foreach $\overline{\sigma} \in [n] \setminus \{\sigma\}$:
    $R^{(i)}_{\overline{\sigma}} \leftarrow \mathsf{PCF.Eval}(\overline{\sigma}, \mathsf{k}_{\overline{\sigma}}, x_i)$
$b' \leftarrow \mathsf{Adv}(1^n, \sigma, \mathsf{k}_\sigma, (x_i, (R^{(i)}_{\overline{\sigma}})_{\overline{\sigma}\neq\sigma})_{i\in[N(n)]})$
return b'

Figure 8.2.: Security of game for a PCF. Here, RSample is the algorithm for reverse sampling the correlation $\mathcal{C}$, as defined in Definition 8.6.

and $n \leq k = k(n) \in \mathsf{poly}$ be an input length. A Pseudorandom Correlation Function (PCF) for $\mathcal{C}$ is defined by a pair of algorithms $(\mathsf{PCF.KeyGen}, \mathsf{PCF.Eval})$ with the following functionality:

- $\mathsf{PCF.KeyGen}(1^n) \to (\mathsf{k}_1,\ldots,\mathsf{k}_n)$. Takes as input the security parameter n. Outputs an n-tuple of keys $(\mathsf{k}_1,\ldots,\mathsf{k}_n)$.
- $\mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x) \to R_\sigma$. Takes as input $\sigma \in [n]$, a key k_σ, and input string $x \in \{0,1\}^k$. Outputs a string $R_\sigma \in \{0,1\}^\ell$.

We say $\mathsf{PCF} = (\mathsf{PCF.KeyGen}, \mathsf{PCF.Eval})$ is a (weak) PCF for the correlation $\mathcal{C}$, if the following properties hold:

Correctness / Pseudorandom $\mathcal{C}$-correlated outputs. For every $\sigma \in [n]$, all efficient adversaries Adv, and all $N \in \mathsf{poly}$, there exists a negligible function negl[] such that for all sufficiently large n,

$$\left|\Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathsf{Adv},N,0}(n) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathsf{Adv},N,1}(n) = 1]\right| \leq \mathsf{negl},$$

where $\mathsf{Exp}^{\mathsf{pr}}_{\mathsf{Adv},N,b}(n)$, for $b \in \{0,1\}$, is as defined in Figure 8.1. In particular, the adversary is given access to $N(n)$ samples.

Security. For all $\sigma \in [n]$ and all efficient adversaries Adv, there exists a negligible function negl[] such that for all sufficiently large n,

$$\left|\Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathsf{Adv},N,\sigma,0}(n) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathsf{Adv},N,\sigma,1}(n) = 1]\right| \leq \mathsf{negl},$$

where $\mathsf{Exp}^{\mathsf{sec}}_{\mathsf{Adv},N,\sigma,b}(n)$, for $b \in \{0,1\}$, is as defined in Figure 8.2 (again, with the adversary given $N(n)$ samples).

8.4. Constructions

In this section, we present the framework for building a PCG/PCF for pseudorandom permutations from a PCG/PCF for pseudorandom bits and then instantiate it from the QA-SD assumption and HSS. In Section 8.4.1, we first describe the $\mathcal{F}$-programmability definition that we will make use of in our constructions.

8.4.1. Doubly-Programmable PCGs

We now introduce our notion of doubly-programmable PCGs. We consider the following class of two-party correlations, where the parties' outputs form subtractive shares of some random variable.

Definition 8.9 (Two-party subtractive correlation generator)**.** Let $(\mathbb{G}, +)$ be an abelian group and $\ell, n \in \mathbb{Z}$ be functions of n. Further, let $\mathcal{F} = \{f_n : \mathbb{G}^\ell \to \mathbb{G}^n\}_n$ be a family of functions. An *subtractive correlation generator* for $\mathcal{F}$ is a correlation generator $\mathcal{C}_\mathcal{F}$, such that $\mathcal{C}_\mathcal{F}(1^n)$ uniformly samples $X \leftarrow \mathbb{F}^\ell$ and outputs $(R_0, R_1) \in \mathbb{F}^n \times \mathbb{F}^n$, such that R_0 and R_1 are distributed uniformly, conditioned on $R_0 - R_1 = f_n(X)$.

A doubly-programmable PCG allows PCG keys to be generated in a special way, such that both (1) the randomness $f_n(X)$, and (2) the R_0 output of party 0, can be reused across multiple instances. Furthermore, we allow a small "tweak" to be applied to $f_n(X)$ such that it can be modified slightly when reusing it. We formalise this notion by using a special seed from which the randomness in $f_n(X)$ is derived, and an additional input to party 1's key generation algorithm for performing the tweak specified by some function from a family $\mathcal{F}$.

Definition 8.10 ($\mathcal{F}$-Programmable)**.** A two-party PCG (resp. PCF) for an additive correlation is *$\mathcal{F}$-programmable* for a class of functions $\mathcal{F}$, if $\mathsf{PCG.Gen}$ (resp. $\mathsf{PCF.Gen}$) can be split into two distinct algorithms:

- $\mathsf{Gen}_0(1^n)$: a PPT algorithm that on input 1^n, outputs a key $\mathsf{k}_0 \in \{0,1\}^*$
- $\mathsf{Gen}_1(\mathsf{k}_0, \mathsf{seed}, f)$: a deterministic algorithm that on input a key k_0, $\mathsf{seed} \in \{0,1\}^n$ and function $f \in \mathcal{F}$, samples and outputs a second key k_1.

We require the following correctness property, for any $f \in \mathcal{F}$:

$$\Pr\left[\begin{array}{l} R_0 - R_2 = f(R_0 - R_1) : \\ \end{array} \begin{array}{r} \mathsf{seed} \xleftarrow{\$} \{0,1\}^n \\ \mathsf{k}_0 \xleftarrow{\$} \mathsf{PCG.Gen}_0(1^n) \\ \mathsf{k}_1 \leftarrow \mathsf{PCG.Gen}_1(\mathsf{k}_0, \mathsf{seed}, \mathsf{id}) \\ \mathsf{k}_2 \leftarrow \mathsf{PCG.Gen}_1(\mathsf{k}_0, \mathsf{seed}, f) \\ R_j \leftarrow \mathsf{PCG.Expand}(\mathsf{k}_j),\ j \in \{0,1,2\} \end{array}\right] \leq 1 - \mathsf{negl}$$

The $\mathsf{PCG.Gen}$ (resp. $\mathsf{PCF.Gen}$) algorithm then simply runs Gen_0, followed by Gen_1 on input a random seed and f as the identity function, to output a pair of keys $(\mathsf{k}_0, \mathsf{k}_1)$.

8.4.2. Permutation PCG From PCG for Biased Bits

In this section, we present our generic template for building a three-party PCG for permutation correlations, given a special form of two-party PCG for secret-shared, biased random bits.

Definition 8.11 (Two-party, ε-biased bits correlation generator)**.** An *ε-biased bits correlation generator* is a subtractive correlation generator $\mathcal{C}$ (from Definition 8.9), such that $\mathcal{C}(1^n)$ samples a vector $\mathbf{x} \leftarrow \{0,1\}^\ell$ where each element x_i is sampled from $\mathsf{Bernoulli}(1/2+\varepsilon)$ and outputs $(\mathbf{R}_0, \mathbf{R}_1) \in \mathbb{Z}_q^\ell \times \mathbb{Z}_q^\ell$, such that the $\mathbf{R}_i$'s are distributed uniformly, conditioned on $\mathbf{R}_0 - \mathbf{R}_1 = \mathbf{x}$.

Definition 8.12 ($\mathcal{D}$-biased permutation correlation generator)**.** An *$\mathcal{D}$-biased permutation correlation generator* is a correlation generator that samples ℓ permutations $(\pi_1, \ldots, \pi_\ell) \leftarrow \mathcal{D}^\ell$ where $\mathcal{D}$ is a distribution on the set of all permutations on l elements S_l. It outputs $\mathbf{R}_1, \ldots, \mathbf{R}_\ell$ where $\mathbf{R}_i(j) = \pi_j[i]$, meaning that in the j-th position of $\mathbf{R}_i$ there is the i-th element of permutation π_j.

We remark that this correlation is reverse-sampleable, as given by the following lemma:

Lemma 8.1. *Let $m, l \in \mathbb{N}$ and $\mathcal{C}$ be a $\mathcal{D}$-biased permutation correlation generator, for some efficiently sampleable distribution $\mathcal{D}$ on S_l. Then, $\mathcal{C}$ is reverse-sampleable.*

Proof. For this, we need to show that there is an efficient algorithm $\mathsf{RSample}$ that takes as input a party index σ and their correlation sample $R_\sigma =: (e_1, \ldots, e_\ell)$ with $e_i \in [l]$, and outputs a list of samples $(R_i)_{i \neq \sigma}$ that fulfills Definition 8.6. $\mathsf{RSample}$ proceeds by rejection sampling of $\mathcal{D}$ as follows. For each $i \in [m]$ it samples a permutation π using $\mathcal{D}$, until $\pi^{-1}(i) = e_i$. Note that the probability for each permutation assigned by $\mathcal{D}$ is constant, so this process is efficient. Also clearly, the output is identically distributed to the output of $\mathcal{C}$. ■

When we build a $\mathcal{D}$-biased correlation generator based on ε-biased bits, this will give the *following distribution* $\mathcal{D}_\varepsilon$ on S_3, the set of permutations on $\mathbb{F}_3$. It builds a permutation π by randomly sampling a biased bit $\mu \xleftarrow{R} \mathsf{Bernoulli}(1/2+\varepsilon)$, and one uniformly random index $i \xleftarrow{R} \mathbb{F}_3$ for $\pi[0]$, and then from the remaining two choices $\mathbb{F}_3 \setminus \{i\}$, assign the smaller one to $\pi[1]$, if $\mu = 1$, and the larger one, otherwise. Assign to $\pi[2]$ the remaining option.

Theorem 8.2. *Suppose that $(\mathsf{PRBits.Gen}_0, \mathsf{PRBits.Gen}_1, \mathsf{PRBits.Expand})$ is an $\mathcal{F}$-programmable PCG for ε-biased bits for the function class $\mathcal{F} = \{\mathsf{id}, (x \mapsto -x)\}$, for some $\varepsilon \in [0, 1/2]$. Then, the construction in Figure 8.3 is a PCG for $\mathcal{D}_\varepsilon$-biased, three-party permutation correlations, where $\mathcal{D}_\varepsilon$ is the distribution on S_3 given above.*

Proof. We argue for each property in turn:

Correctness. Let $(\mathsf{k}_0, \mathsf{k}_1, \mathsf{k}_2)$ be as output by $\mathsf{PCG.Gen}(1^n)$, and $R_i \leftarrow \mathsf{PCG.Expand}(i, \mathsf{k}_i)$, i.e. define $R'_i = \mathsf{PRBits.Expand}(i, \mathsf{k}_i)$ for $i \in \{0,1,2\}$ and then, we have $R_0 = R'_0$, $R_1 = (2, \ldots, 2) + R'_1$, $R_2 = (1, \ldots, 1) + R'_2$.

Now, the R'_i denote the original subtractive shares of the bit vectors as output by $\mathsf{PRBits.Expand}(i, \mathsf{k}_i)$. Then, by construction, $R'_0 - R'_1 = (\mu_1, \ldots, \mu_\ell)$ forms a vector of ε-biased bits, and $R'_0 - R'_2 = (-\mu_1, \ldots, -\mu_\ell)$, by $\mathcal{F}$-programmability (cf. Definition 8.10). Let $k \in [\ell]$, then $R'_1[k] = R'_0[k] - \mu_k$ and $R'_2[k] = R'_0 + \mu_k$. Note that $R_0[k] + R_1[k] + R_2[k] = R'_0[k] + R'_1[k] + R'_2[k] = R'_0[k] + (R'_0[k] - \mu_k) + (R'_0 + \mu_k) = 0$. Hence, they form an additive sharing of zero for each k. Also, observe that $R_0[k], R_1[k], R_2[k]$ are pairwise distinct, hence they form a permutation.

Using the pseudorandomness of the PCG output, R_0 is a fresh random share in $\mathbb{F}_3$, and hence, R_0 is computationally close to uniform over $\mathbb{F}_3$, as in $\mathcal{D}_\varepsilon$. Then, the two possibilities for R_1 are exactly determined by the ε-biased bit as generated by PRBits. This shows that it is indistinguishable from a sample of $\mathcal{D}_\varepsilon$ for each $k \in [\ell]$.

Security.

We prove security via a sequence of hybrid arguments.

Hybrid $\mathcal{H}_0$. This hybrid corresponds to the $(\mathsf{k}_i, (R_j)_{j \neq i})$ as computed in Figure 8.3.

Hybrid $\mathcal{H}_1$. In this hybrid, we reverse sample R'_j for $j \neq i$ from R'_i and compute $(R_j)_{j \neq i}$ as computed in Figure 8.3.

$\mathcal{H}_1 \approx_c \mathcal{H}_0$ by the security of the PRBits PCG.

Hybrid $\mathcal{H}_2$. In this hybrid, we change how the outputs $(R_j)_{j \neq i}$ are computed. In particular, we first sample uniformly random bits $(\mu_1, \ldots, \mu_\ell)$. Then, we proceed in one of three cases: (1) if $i = 0$ we let $R_1 := R'_0 + 2 - (\mu_1, \ldots, \mu_\ell)$ and let $R_2 := R'_0 + 1 + (\mu_1, \ldots, \mu_\ell)$, (2) if $i = 1$ we let $R_0 := R'_1 - 2 + (\mu_1, \ldots, \mu_\ell)$ and let $R_2 := R'_0 + 1 + (\mu_1, \ldots, \mu_\ell)$, or (3) if $i = 2$ we let $R_1 := R'_0 + 2 - (\mu_1, \ldots, \mu_\ell)$ and let $R_0 := R'_2 - 1 - (\mu_1, \ldots, \mu_\ell)$.

$\mathcal{H}_2 \approx_c \mathcal{H}_1$ by the pseudorandomness property of the PRBits PCG.

Hybrid $\mathcal{H}_3$. In this hybrid, we reverse sample $(R_j)_{j \neq i}$.

$\mathcal{H}_3 \approx_c \mathcal{H}_2$. In particular, note that in $\mathcal{H}_2$, the keys k_j's for $j \neq i$ are no longer used to compute the outputs. Therefore, by the correctness of the reverse sampling algorithm, the two hybrids are identical.

■

Obtaining a PCF instead of PCG.

This construction can be easily extended to obtain a pseudorandom correlation *function*, if we start with a two-party PCF for ε-biased bits with $\mathcal{F}$-programmability instead of a PCG.

8.4.3. Programmable PCG for $(1/6)$-Biased Bits from Quasi-Abelian Syndrome Decoding

We now show how to instantiate the previous template, by giving a construction of a programmable PCG for ε-biased bits, where $\varepsilon = \frac{1}{6}$, based on the hardness of the quasi-abelian syndrome decoding problem [BCCD23].

PCG for permutation correlations over $\mathbb{F}_3$

$\mathsf{Gen}(1^n)$

1 : $\mathsf{k}_0 \leftarrow \mathsf{PRBits.Gen}_0(1^n)$
2 : $\mathsf{seed} \xleftarrow{\mathrm{R}} \{0,1\}^n$
3 : $\mathsf{k}_1 \leftarrow \mathsf{PRBits.Gen}_1(\mathsf{k}_1, \mathsf{seed}, f_1 = \mathsf{id})$
4 : $\mathsf{k}_2 \leftarrow \mathsf{PRBits.Gen}_1(\mathsf{k}_1, \mathsf{seed}, f_2 = (x \mapsto -x))$
5 : **return** $(\mathsf{k}_0, \mathsf{k}_1, \mathsf{k}_2)$

$\mathsf{Expand}(i, \mathsf{k}_i)$

1 : $R'_i = \mathsf{PRBits.Expand}(i, \mathsf{k}_i)$
2 : **if** $i = 0$: **return** $R_i := R'_i$
3 : **if** $i = 1$: **return** $R_i := (2, \ldots, 2) + R'_i$
4 : **else return** $R_i := (1, \ldots, 1) + R'_i$

Figure 8.3.: PCG for permutation correlations over $\mathbb{F}_3$

8.4.3.1. QA-SD Background.

The main motivation for quasi-abelian syndrome decoding is to obtain a syndrome decoding problem over a ring structure $\mathcal{R}$, where $\mathcal{R}$ is isomorphic to m copies of $\mathbb{F}_q$ for some large m. Using a cyclotomic polynomial ring $\mathcal{R} = \mathbb{F}_q[X]/f(X)$ as in ring-LWE or ring-LPN, this can be done only for $q = \Omega(m)$. Instead, quasi-abelian syndrome decoding allows us to use any $q \geq 3$.

Let $\mathbb{G}$ be a finite abelian group of order N. We define a ring $\mathcal{R}$ as the group algebra of $\mathbb{G}$ with coefficients in the finite field $\mathbb{F}_q$, that is, the set $\mathbb{F}_q[\mathbb{G}]$ of formal linear combinations:

$$\mathbb{F}_q[\mathbb{G}] := \left\{ \sum_{g \in \mathbb{G}} a_g g \mid a_g \in \mathbb{F}_q \right\}$$

Ring multiplication is defined by the convolution:

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

The group algebra $\mathbb{F}_q[\mathbb{G}]$ is isomorphic to the vector space $\mathbb{F}_q^N$ via the map $\phi : \sum_{i=0}^{N-1} a_i g_i \mapsto (a_0, \ldots, a_{N-1})$. This allows us to define the *weight* of an element $a \in \mathbb{F}_q[\mathbb{G}]$ as the Hamming weight of $\phi(a)$. (Note that the weight is independent of the ordering of the group $\mathbb{G}$ chosen in the definition of ϕ.)

Definition 8.13 (Decisional quasi-abelian syndrome decoding)**.** Given a target weight t and compression paramater c, let $\mathcal{D}_t(\mathbb{F}_q[\mathbb{G}])$ be a noise distribution that samples weight-t

elements of $\mathbb{F}_q[\mathbb{G}]$. The decisional $\mathsf{QASD}_{t,c}$ problem for $\mathcal{D}_t$ is to distinguish between the following two distributions with noticeable advantage:

- $(\mathbf{a}', u)$, where $\mathbf{a}' \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]^{c-1}, u \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]$
- $(\mathbf{a}', \langle \mathbf{a}, \mathbf{e} \rangle)$, where $\mathbf{a} = (1, \mathbf{a}')$ and $\mathbf{a}' \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]^{c-1}$, $\mathbf{e} \xleftarrow{\$} \mathcal{D}_t(\mathbb{F}_q[\mathbb{G}])^c$

As in prior works [BCCD23], to improve efficiency we can use a *regular* noise distribution where each weight-t vector is divided into N/t consecutive blocks of weight-1.

Instantiating the group algebra. Following [BCCD23], we can instantiate $\mathbb{G}$ with a direct product of cyclic groups $\prod_{i=1}^{N} \mathbb{Z}/(q-1)\mathbb{Z}$. This implies that $\mathbb{F}_q[\mathbb{G}]$ is isomorphic to $\mathbb{F}_q[X_1, \ldots, X_N]/(X_1^{q-1} - 1, \ldots, X_N^{q-1} - 1)$ via an efficiently-computable isomorphism, and we have $|\mathbb{F}_q[\mathbb{G}]| = (q-1)^N$.

Let $\mathsf{QA\text{-}Eval}_{q,N} : \mathbb{F}_q[\mathbb{G}] \to \mathbb{F}_q^{(q-1)^N}$ be the map takes an element of the group algebra, viewed as a multivariate polynomial in $\mathbb{F}_q[X_1, \ldots, X_N]/(X_1^{q-1}-1, \ldots, X_N^{q-1}-1)$, and evaluates it at every possible point. That is, $\mathsf{QA\text{-}Eval}_{q,N}(f) = \big(f(x_1, \ldots, x_N)\big)_{(x_1,\ldots,x_n)\in\{0,\ldots,q-1\}^N}$. Since $\mathsf{QA\text{-}Eval}$ is a ring homomorphism, for any $f, g, h \in \mathbb{F}_q[\mathbb{G}]$, it holds that

$$\mathsf{QA\text{-}Eval}_{q,N}(fg + h) = \mathsf{QA\text{-}Eval}(f) * \mathsf{QA\text{-}Eval}(g) + \mathsf{QA\text{-}Eval}(h)$$

where $*$ denotes the entry-wise product. This allows us to directly convert additive shares of a product over $\mathbb{F}_q[\mathbb{G}]$ into a vector of shares of $(q-1)^N$ products over $\mathbb{F}_q$.

8.4.3.2. Construction.

We want to give the parties shares of a random square $x^2 \in \mathbb{F}_3$. By choosing $x = \langle \mathbf{a}, \mathbf{e} \rangle$, as in the QASD assumption, then we can distribute shares of x^2 to two parties by giving out shares of the tensor product vector $\mathbf{e} \otimes \mathbf{e}$. Since each $e_i \in \mathbb{F}_q[\mathbb{G}]$ is t-sparse, the product $e_i e_j$ is at most t^2-sparse, and can be shared using FSS for a sum of t^2 point functions. We need to also achieve the $\mathcal{F}$-programmability feature, so that instead of shares of x^2, we may additionally distributed shares of $f(x^2)$ for some function $f \in \mathcal{F}$. We will allow $\mathcal{F}$ to be the class of linear maps from $\mathbb{F}_q \to \mathbb{F}_q$ (applied to each coordinate of x^2 independently, viewing the group algebra as a vector space over $\mathbb{F}_q$). To do this, it suffices to tweak the non-zero values of each tensored noise terms $e_i e_j$ (denoted $\mathbf{b}^i \otimes \mathbf{b}^j$ in Figure 8.4), applying f to each entry. Since the shares of x^2 obtained in the Expand algorithm are computed as a linear function of the shares of the tensor product, this results in the same linear map being applied to x^2.

Theorem 8.3. *Assume that* SPFSS *is a secure* FSS *scheme for sums of point functions and that the* $\mathsf{QA\text{-}SD}$ *assumption holds. Then, the protocol described in Figure 8.4 is an* $1/6$*-biased bits correlation generator over* $\mathbb{F}_3$.

Proof. We prove each property in turn.

Correctness. We prove correctness via a hybrid argument.

Let $e^i = \sum_{j\in[0,t)} \mathbf{b}^i[j]\mathbf{A}^i[j]$.

Construction PRBits

Parameters: Noise weight $t = t(\lambda)$, compression factor c, field $\mathbb{F}_3$.
A programmable SPFSS scheme $\mathsf{SPFSS} = (\mathsf{SPFSS.Gen}_0, \mathsf{SPFSS.Gen}_1, \mathsf{SPFSS.Eval})$

Public Input: $c-1$ vectors over $\mathbb{F}_3$, corresponding to the output of $\mathsf{Eval}(a_i)$ for uniformly random $\mathbf{a} = (1, a_1, \ldots, a_{c-1})$.

$\mathsf{PRBits.Gen}_0(1^n, \mathsf{seed})$

1 : **foreach** $i, j \in [0, c)$:
2 : $\quad (K_0^{i,j}) \xleftarrow{\$} \mathsf{SPFSS.Gen}_0(1^n)$
3 : Let $\mathsf{k}_0 = ((K_0^{i,j})_{i,j\in[0,c)})$
4 : Output k_0

$\mathsf{PRBits.Gen}_1(\mathsf{seed}, \mathsf{k}_0, f)$

1 : **foreach** $i \in [0, c)$:
2 : $\quad \mathbf{A}^i \xleftarrow{\$} (g_1, \ldots, g_t)_{g_i\in\mathbb{G}}, \quad \mathbf{b}^i \xleftarrow{\$} (\mathbb{F}_3^*)^t$ // Using randomness from seed
3 : **foreach** $i, j \in [0, c)$:
4 : $\quad (K_1^{i,j}) \xleftarrow{\$} \mathsf{SPFSS.Gen}_1(1^n, \mathbf{A}^i \otimes \mathbf{A}^j, f(\mathbf{b}^i \otimes \mathbf{b}^j))$
5 : Let $\mathsf{k}_1 = ((K_1^{i,j})_{i,j\in[0,c)})$
6 : Output k_1

$\mathsf{PRBits.Expand}(\sigma, \mathsf{k}_\sigma)$

1 : For $i, j \in [0, c)$:
2 : $\quad u_{\sigma,i+cj} \leftarrow \mathsf{SPFSS.Eval}(\sigma, K_\sigma^{i,j})$
3 : $z_\sigma \leftarrow \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_\sigma \rangle$
4 : Output $\mathsf{QA\text{-}Eval}_{3,N}(z_\sigma)$

Figure 8.4.: PCG for biased bits over $\mathbb{F}_3$

Hybrid $\mathcal{H}_0$. In this hybrid, we define $z_0 = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_0 \rangle, z_1 = \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_1 \rangle$ as computed in the real protocol execution and output (z_0, z_1).

Hybrid $\mathcal{H}_1$. In this hybrid, instead of computing $\mathbf{u}_1$ with $\mathsf{SPFSS.Eval}$, we replace $\mathbf{u}_1$ with $\mathbf{u}_0 - \mathbf{e} \otimes \mathbf{e}$ and compute $z_1 \leftarrow \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_0 - \mathbf{e} \otimes \mathbf{e} \rangle$

$\mathcal{H}_1 \approx_c \mathcal{H}_0$ by the SPFSS correctness property.

Hybrid $\mathcal{H}_2$. In this hybrid, we replace $\mathbf{u}_0$ with a randomly sampled $\tilde{\mathbf{u}}_0$. Compute $\mathbf{u}_1$ and z_0, z_1 as before using the new $\tilde{\mathbf{u}}_0$. We have $\mathbf{u}_1 = \tilde{\mathbf{u}}_0 - \mathbf{e} \otimes \mathbf{e}$, $z_0 = \langle \mathbf{a} \otimes \mathbf{a}, \tilde{\mathbf{u}}_0 \rangle$, and $z_1 = \langle \mathbf{a} \otimes \mathbf{a}, \tilde{\mathbf{u}}_0 - \mathbf{e} \otimes \mathbf{e} \rangle$

$\mathcal{H}_2 \approx_c \mathcal{H}_1$ by the pseudorandomness of the outputs of $\mathsf{FSS.Eval}$ (cf.Remark 5).

Hybrid $\mathcal{H}_3$. Here, we compute the output z_1 as $z_0 - \langle \mathbf{a}, \mathbf{e} \rangle^2$. This output is identically distributed to the previous hybrid, since in $\mathcal{H}_2$ we have:

$$z_1 = \langle \mathbf{a} \otimes \mathbf{a}, \tilde{\mathbf{u}}_0 \rangle - \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{e} \otimes \mathbf{e} \rangle = z_0 - \langle \mathbf{a}, \mathbf{e} \rangle^2$$

Hybrid $\mathcal{H}_4$. Here, we sample $z_0, x \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]$ uniformly at random and compute $z_1 = z_0 - x^2$.

We have $\mathcal{H}_4 \approx_c \mathcal{H}_3$, due to the QASD assumption which says that $x = \langle \mathbf{a}, \mathbf{e} \rangle$ is indistinguishable from random.

Finally, we observe that this distribution is equivalent to a sample from the ideal $\frac{1}{6}$-biased bits correlation generator, as each $\mathbb{F}_3$ component of x^2 is zero with probability $\frac{1}{3}$ and one with probability $\frac{2}{3}$.

Security. Security is argued as a sequence of hybrid arguments. We use $\sigma = 1$ and note that the case for $\sigma = 0$ is symmetric. We want to show that:

$$\{(\mathsf{k}_1, x, z_0)\} \approx_c \left\{ (\mathsf{k}_1, \tilde{x}, \tilde{z}_0) \;\middle|\; \begin{array}{r} \tilde{x} \xleftarrow{\$} \mathbb{F}_3 \\ \tilde{z}_0 = \tilde{x}^2 + z_1 \end{array} \right\}.$$

Hybrid $\mathcal{H}_0$. This hybrid corresponds to the (k_1, x, z_0) computed as in the real protocol.

Hybrid $\mathcal{H}_1$. In this hybrid, we replace all the FSS keys K_1^{ij} in k_1 with simulated keys.

$\mathcal{H}_0 \approx_c \mathcal{H}_1$ via a straightforward hybrid argument replacing each of the FSS keys one-by-one with a simulated key, and relying on the correctness and security of the FSS scheme.

Hybrid $\mathcal{H}_2$. In this hybrid, we sample $\tilde{x}$ at random instead of computing is using the FSS key K_1^{ij}.

$\mathcal{H}_1 \approx_c \mathcal{H}_2$ by the $\mathsf{QA\text{-}SD}$ assumption.

Hybrid $\mathcal{H}_3$. In this hybrid, we undo the changes made in $\mathcal{H}_1$ and switch back to the original FSS keys.

$\mathcal{H}_3 \approx_c \mathcal{H}_2$ via a straightforward hybrid argument replacing each of the FSS keys one-by-one, and relying on the correctness and security of the FSS scheme.

At this point, we have that $\mathcal{H}_3$ is distributed exactly as $(\mathsf{k}_1, \tilde{x}, \tilde{z_0})$, concluding the proof.

Programmability. The construction is programmable (that is, we have $z_0 - z_2 = f(z_0 - z_1)$) due to the programmability of the FSS. In particular, this follows from the fact that by scaling the non-zero values for the point functions defined by $\mathbf{b}^i \otimes \mathbf{b}^i$ by some constant $\Delta \in \mathbb{F}_q$ at key generation time, the secret shared output becomes a share of $\Delta \cdot x^2$ instead of x^2. The programmability of the FSS ensures that we then obtain the necessary $\mathcal{F}$-programmability feature, since we can keep the share z_0 independent (generated by Gen_0) and get $z_0 - z_2 = \Delta(z_0 - z_1)$. This immediately generalizes to linear functions f.

In addition, as was discussed earlier, for random $x \in \mathbb{F}_3$, x^2 is 1 for $x = 1, 2$ (2/3 of the times) and 0 for $x = 0$ (1/3 of the times). This means that x^2 is distributed according to $\mathsf{Bernoulli}(1/2 + 1/6)$ over $\mathbb{F}_3$ and so our construction is a 1/6-biased bits correlation generator over $\mathbb{F}_3$. ■

8.4.4. Unbiased PCF for Permutations Constructions

In this section, we describe a construction that is based on HSS with simulatable memory shares to obtain a PCF for unbiased permutation correlations.

For this, let P be an HSS program for a PRF with 1-bit output. Denote it by F_κ, where κ is the PRF key. To run it, we need HSS shares of the key, generated via $([\![\kappa]\!]^A, [\![\kappa]\!]^B) \leftarrow \mathsf{HSS.Input}(\mathsf{pk}, \kappa)$ [BCMPR24]. Then, Alice gets $[\![\kappa]\!]^A$, and both, Bob and Carol get $[\![\kappa]\!]^B$.

Now, when Alice and either Bob or Carol would run $\mathsf{Eval}(\sigma, \mathsf{ek}^\sigma, [\![\kappa]\!]^\sigma, P_x)$, they obtain, upon reconstruction, a pseudorandom bit $\mu = F.\mathsf{Eval}(\kappa, x)$. Now, remember that we want Alice to obtain a share that, together with the share of Bob, is a share of a pseudorandom bit μ, and, together with the share of Carol, is a share of $1 - \mu$. Hence, we want to compute $\alpha \cdot \mu$, where in one case, $\alpha = 1$, and in the other case, $\alpha = -1$. Then, it suffices that Carol subtracts 1 from her subtractive share, thereby converting it to a share of $1 - \mu$.

To do this, we can provide the α as memory shares and evaluate P_x for an PRF input x, using our extended evaluation (cf. Section 8.3.1). And, as we want to keep Alice's share fixed, we do so by using memory shares M^A, M^B that are simulatable, i.e. Alice uses $M^A \leftarrow \mathsf{Sim}_0(1^n)$ and to get a memory share, whose value is not yet fixed, but can be determined by generating a share using Sim_1.

Later, this is programmed twice, to encode certain $\alpha_B \in \mathcal{I}$ but also $\alpha_C \in \mathcal{I}$ using Sim_1, for use with Bob and Carol, respectively. Then, $\mathsf{ExtEval}$ will output a share of $\alpha_p \cdot F.\mathsf{Eval}(\kappa, x)$ for $p \in \{B, C\}$, where $F.\mathsf{Eval}(\kappa, x)$ is pseudorandom bit as output by the PRF. As motivated above, we choose $\alpha_B = 1$ and $\alpha_C = -1$.

Theorem 8.4. *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ *be an HSS with simulatable memory shares (with algorithms* $\mathsf{Sim}_0, \mathsf{Sim}_1$*) and with input space* $\mathcal{I} = \mathbb{F}_3$ *and share space* $\mathcal{S} = \mathbb{F}_3$*. Moreover, let* F *be a pseudorandom function and for* $x \in \{0,1\}^k$*, let* P_x *be the HSS program that runs* $F.\mathsf{Eval}(\cdot, x)$ *when given the PRF key as input. Then,* PCF *of Figure 8.5*

is a PCF for the 3-correlation generator $\mathcal{C}$ given by Definition 8.12, where $\mathcal{D}$ is the uniform distribution over S_3 and $m \in \mathbb{N}$ is the output length of F. (For simplicity of notation, we identify S_3 as the set of permutations over the elements of $\mathbb{F}_3$.)

Proof. We show each property in turn.

Correctness. Let $(\mathsf{k}_0, \mathsf{k}_1, \mathsf{k}_2)$ as output by $\mathsf{PCF.Gen}(1^n)$. Let k_i be parsed as $(\mathsf{pk}, [\![\kappa]\!]_i, \mathsf{ek}_i, M^i)$ for $i \in \{0,1,2\}$. Moreover, let $R_i \leftarrow \mathsf{PCF.Eval}(i, \mathsf{k}_i, x)$, for some $x \in \{0,1\}^k$. More concretely,

$$
\begin{aligned}
R_0 &= \mathsf{ExtEval}(0, \mathsf{ek}^A, M^A, [\![\kappa]\!]^A, P_x) \\
R'_1 &= \mathsf{ExtEval}(1, \mathsf{ek}^B, M^B, [\![\kappa]\!]^B, P_x) \\
R'_2 &= \mathsf{ExtEval}(2, \mathsf{ek}^B, M^C, [\![\kappa]\!]^B, P_x) - 1 \\
R_1 &= 2 + R'_1 \\
R_2 &= 2 + R'_2.
\end{aligned}
$$

We need to show that $(R_0, R_1, R_2) \in \mathbb{F}_3^3$ as obtained in this manner, is a uniformly random permutation over the elements of $\mathbb{F}_3$.

By the simulation correctness of the extended evaluation, we know that by construction, with overwhelming probability, $R_0 - R'_1 = F.\mathsf{Eval}(\kappa, x) \in \{0,1\}$, and $R_0 - R'_2 = 1 - F.\mathsf{Eval}(\kappa, x) \in \{0,1\}$. Also, because the distinguisher only gets the 3-tuple, by the security of the HSS scheme, they cannot learn anything about κ, and hence, $F.\mathsf{Eval}(\kappa, x)$ is computationally indistinguishable from a random bit. Then, using $R'_1 = R_0 - \mu$, and $R'_2 = R_0 + \mu - 1$, for $\mu := F.\mathsf{Eval}(\kappa, x)$, we get $R_0 + R_1 + R_2 = R_0 + (2 + R'_1) + (2 + R'_2) = R_0 + 2 + (R_0 - \mu) + (R_0 + 1 + \mu) = 3R_0 + 2 - \mu + 1 + \mu = 0 \pmod 3$, i.e., the R's form an additive sharing of zero, as required.

Next, we show that they are pairwise distinct. This is because $R_0 = R_1$ implies $R_0 - R'_1 = \mu = 2$, which is false. (The other cases are implied and/or similar.) Hence, (R_0, R_1, R_2) forms a permutation. It remains to show that it is uniformly generated. As R_0 is a fresh random subtractive share, it follows if the HSS satisfies the additional property that the marginal distribution of an output share is computationally close to random. While this does not follow automatically from the HSS definition, it is folklore any HSS can be trivially modified to satisfy this slightly stronger property by letting $\mathsf{HSS.Input}$ additionally output a fresh PRF key (for a PRF with range $\mathbb{F}_3$) in both input shares, and letting $\mathsf{HSS.Eval}$ sum the PRF output (evaluated on the RMS program viewed as a string) to its output. This preserves correctness (the PRF evaluations cancel out when subtracting the shares) and guarantees that each share is individually pseudorandom. With this change, R_0 is computationally close to uniform over $\mathbb{F}_3$. Then, given this R_0, we show that the two possibilities for R_1 are exactly determined by the pseudorandom bit of the PRF, so it is indistinguishable from being uniformly chosen from $\mathbb{F}_3 \setminus \{R_0\}$.

Security. Here, we show PCF security for each $\sigma \in \{0,1,2\}$. Note that the proof is different for the different σ's, because for $\sigma = 0$, the memory share M^A that is part of

the key does not need to be simulated. We prove the case where $\sigma = 0$. The other cases follow via an analogous proof.

Hybrid $\mathcal{H}_0$. This hybrid consists of $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(n)$ game as defined in Figure 8.2.

Hybrid $\mathcal{H}_1$. In this hybrid, we change how the output $R_1^{(i)}$ of Bob is computed (for all $i \in [N]$). First, we reconstruct the PRF key κ from the HSS input shares contained in the PCF keys and then evaluate the PRF on input x_i to get the output μ_i. Then, we use Alice's output $R_0^{(i)}$ to compute Bob's share $R_1^{(i)}$ as $R_0^{(i)} + 2 - \mu_i$.

We note that $\mathcal{H}_1$ is identical to $\mathcal{H}_0$, since the change is merely syntactic. Now, in $\mathcal{H}_1$, Bob's share is no longer computed using k_1.

Hybrid $\mathcal{H}_2$. In this hybrid, we change how the output of Carol is computed. As before, we first reconstruct the PRF key κ from the HSS input shares contained in the PCF keys and then evaluate the PRF on input x_i to get the output μ_i. Then, we use Alice's output $R_0^{(i)}$ to compute Carol's share $R_1^{(i)}$ as $R_0^{(i)} + 1 + \mu_i$.

We note that $\mathcal{H}_2$ is again identical to $\mathcal{H}_1$, since the change is merely syntactic. Now, in $\mathcal{H}_2$, Carol's share is no longer computed using k_2.

Hybrid $\mathcal{H}_3$. In this hybrid, we replace the HSS input shares of κ with an input share of 0.

$\mathcal{H}_3 \approx_c \mathcal{H}_2$ by the security of HSS.

Hybrid $\mathcal{H}_4$. In this hybrid, we replace the pseudorandom bit μ_i (as output in $\mathcal{H}_3$ using the PRF) with a uniformly random bit.

$\mathcal{H}_4 \approx_c \mathcal{H}_3$ by the pseudorandomness of the PRF.

Hybrid $\mathcal{H}_5$. This hybrid consists of $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(n)$ game as defined in Figure 8.2.

$\mathcal{H}_5$ is identical to $\mathcal{H}_4$ given that in $\mathcal{H}_4$, we already sample $R_1^{(i)}$ and $R_2^{(i)}$ entirely from Alice's output $R_0^{(i)}$.

■

8.5. Applications

In this section, we describe two applications of our constructions. The first application is for slot assignment in Dining Cryptographer networks (DC-nets). The second application is to single secret leader election. While currently limited to the three party setting, future constructions of PCGs and PCFs for pseudorandom permutations capable of handling more parties would automatically extend these applications to a multi-party setting.

HSS-based PCF for pseudorandom permutations over $\mathbb{F}_3$

$\mathsf{Gen}(1^n)$

1 : $(\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B)) \leftarrow \mathsf{HSS.Setup}(1^n)$
2 : $\kappa \leftarrow F.\mathsf{KeyGen}(1^n)$
3 : $([\![\kappa]\!]^A, [\![\kappa]\!]^B) \leftarrow \mathsf{HSS.Input}(\mathsf{pk}, \kappa)$
4 : $M^A \leftarrow \mathsf{Sim}_0(1^n)$
5 : $M^B \leftarrow \mathsf{Sim}_1(M^A, \alpha_B := 1, (\mathsf{ek}^A, \mathsf{ek}^B))$
6 : $M^C \leftarrow \mathsf{Sim}_1(M^A, \alpha_C := -1, (\mathsf{ek}^A, \mathsf{ek}^B))$
7 : Set $\mathsf{k}_0 := (\mathsf{pk}, [\![\kappa]\!]^A, \mathsf{ek}^A, M^A)$
8 : Set $\mathsf{k}_1 := (\mathsf{pk}, [\![\kappa]\!]^B, \mathsf{ek}^B, M^B)$
9 : Set $\mathsf{k}_2 := (\mathsf{pk}, [\![\kappa]\!]^B, \mathsf{ek}^B, M^C)$
10 : **return** $(\mathsf{k}_0, \mathsf{k}_1, \mathsf{k}_2)$

$\mathsf{Eval}(i, \mathsf{k}_i, x)$

1 : **parse** $\mathsf{k}_i = (\mathsf{pk}, k', \mathsf{ek}, M)$
2 : $\langle \mu \rangle_i \leftarrow \mathsf{ExtEval}(i, \mathsf{ek}, M, k', P_x)$ ▷ P_x is the program for $F.\mathsf{Eval}(\cdot, x)$
3 : **if** $i = 0$: **return** $\langle \mu \rangle_i$
4 : **if** $i = 1$: **return** $2 + \langle \mu \rangle_i$
5 : **else return** $1 - \langle \mu \rangle_i$

Figure 8.5.: HSS-based PCF construction

8.5.1. Anonymous broadcast via DC-nets

In a DC-net [Cha88; GJ04], parties use secret sharing to hide the provenance of a message. DC-nets form the backbone of anonymous broadcast protocols [CBM15; ECZB21; NSD22; APY20]. The main idea is that each party P_i with message $m_i \in \{0,1\}^\ell$ generates a one-hot vector $\mathbf{v}_i$ with the message m_i in the j-th coordinate, where j is chosen randomly by party P_i. The vector $\mathbf{v}_i$ is then additively secret-shared with all other parties. Given all the secret shares, the parties can locally generate a secret share of the combined vector $\mathbf{v}$ by adding all the secret shared vectors together. That is, they compute a share of $\mathbf{v} := \sum_{i=1}^{n} \mathbf{v}_i$. Finally, by broadcasting their shares of $\mathbf{v}$ (or posting them to a bulletin board), the parties obtain the vector $\mathbf{v}$ in the clear, which reveals the set of messages while hiding the provenance of each message. In particular, DC-nets provide unlinkability between parties and their message since the position of each party's message in $\mathbf{v}$ was chosen randomly and independently.

A practical challenge with the DC-net architecture is handling *message collisions*—an artifact of each party randomly selecting the "slot" of $\mathbf{v}$ they write their message into. Concretely, the problem is that if two parties P_i and P_j write their messages in the same non-zero coordinate of the vector $\mathbf{v}$, then the corresponding messages are added together resulting in a "clobbered" output. The standard solution to this problem is to either (1) run a special protocol to allow each party to "reserve" a random slot in the output vector [SB07; NSD22] or (2) increase the size of the vector to make the probability of a collision sufficiently small [CBM15; APY20]. However, these solutions are undesirable given that they both add communication and computation overheads on the parties. As highlighted by Golle and Juels [GJ04]: "There is no good non-interactive means of enabling all players to select distinct message position."

Here, we show that our constructions of PCGs and PCFs for permutations, provide such a *non-interactive* solution to address collisions, as originally imagined by Golle and Juels: Each party obtains a pseudorandom index automatically assigning them to a unique slot. Therefore, after a one-time setup protocol, the parties can use a DC-net without the risk of collisions. Moreover, this approach avoids the need for increasing the size of the message vector or running an allocation protocol at each round.

While this application to DC-nets is immediate for our unbiased PCF construction, we need to be a little more careful with our biased PCG construction. We show that in the context of DC-nets, where parties broadcast over multiple rounds, we can use a randomness extractor to make the permutation unbiased, while only modestly increasing the communication overhead on the parties (in particular, the parties only need to broadcast a few additional bits in each round, independent of the message size).

Dealing with bias via the Von Neumann extractor. We recall that the bias of the permutation comes from the pseudorandom bit μ being biased towards one. The Von Neumann extractor [Von+63] is a simple way of unbiasing any Bernoulli distribution and works as follows: Given a sequence of biased random bits, examine each consecutive pair of bits and discard all pairs where the xor of the two bits is zero. The remaining bits are then guaranteed to be unbiased. Our idea is to use the multi-round DC-net setting to allow parties to unbias the bits "on-the-fly" at round r for use in round $r+1$. The

main idea is that the parties can use the biased bit PCG from Figure 8.3 to generate a sequence of $2k$ biased bits at round r. Then, at the end of round r, they reveal k sums corresponding to the sum of each pair of consecutive bits (the sum is computed over $\mathbb{F}_3$). Then, at round $r+1$, the parties use the permutation generated using the first bit μ from a pair whose bits sum to one. By the guarantees of the Von Neumann extractor, all the bits in these pairs are distributed as unbiased pseudorandom bits (if the sum is $1 \in \mathbb{F}_3$, then the two bits in the pair are not equal). Moreover, because the parties only reveal the sum of the bits in the pair, no party learns whether the value of the first bit in the pair is zero or one.[3]

Overall, the protocol proceeds as follows:

One-time Setup. In the one-time setup, the i-th party is given a PCG key k_i for Figure 8.3. The setup additionally distributes a uniformly random starting permutation to all parties. The round number r is initialized to zero. Given the PCG key k_i, the i-th party runs $\mathsf{PRBits.Expand}$ to generate a sequence of $\ell \cdot 2k$ biased pseudorandom bits and obtains the associated $2k$ pseudorandom biased permutation outputs.

Step 1: Broadcast. In the $r+1$-st round, the parties use the r-th permutation as their slot assignment for the DC-net protocol and compute shares of the vector $\mathbf{v}$ containing all their messages. Then, each party parses the r-th block of $2k$ consecutive shares of bits output by $\mathsf{PRBits.Expand}$ as a sequence of k pairs, and broadcasts the k shares obtained by summing the bits in each pair in this sequence. Each party outputs their share of $\mathbf{v}$ along with their shares of the k pair sums.

Step 2: Generating the next permutation. With the k sums revealed, let j denote the index of the first sum that equals 1 in the sequence of k sums. The parties select the $2j$-th permutation output by $\mathsf{PCG.Expand}$ for the r-th round, which corresponds to selecting the pseudorandom permutation generated using the $2j$-th bit output by $\mathsf{PRBits.Expand}$.

Analysis. For each pair that sums to 1, the parties do not learn which bit in the pair was zero vs. one. Therefore, the sum of the two consecutive shares reveals whether or not the two bits are equal in $\mathbb{F}_3$, and nothing else. Then, by the properties of the Von Neumann extractor, we have the guarantee that all the pairs that sum to 1 are unbiased. By selecting the first such pair, we have the guarantee that both the bits in that pair are unbiased. Therefore, the permutation associated with the first bit of that sequence is also guaranteed to be unbiased.

Determining the value of k. We still need to analyze how large k needs to be in order to guarantee, with high probability, that at least one pair has two differing bits (so that the sum of the bits is one). We compute k using a simple Chernoff bound. The probability that a bit is 1 is $\frac{2}{3}$. Therefore, the probability of getting two differing consecutive bits is $(\frac{1}{3} \cdot \frac{2}{3}) + (\frac{2}{3} \cdot \frac{1}{3}) = \frac{4}{9}$. Let X be the random variable representing the number of pairs of consecutive different bits in k trials. We want to find k such that $\Pr[X > 0] \geq 1 - 2^n$ so

[3] Indeed, all they learn is that the two bits are distinct. Moreover, all pairs that sum to 0 or 2 are discarded and therefore do not affect the protocol.

as to guarantee a negligible correctness failure in the security parameter. The expected value of X is $4k/9$ and by the Chernoff bound, we have that: $\Pr[X = 0] \leq e^{-2k/9}$. Therefore, by solving for k such that $2^{-n} \leq e^{-2k/9}$, we get that we need $k \geq \frac{9}{2} n \ln(2)$ so that $\Pr[X > 0] \geq 1 - 2^{-n}$. For $n = 40$, this requires $k \approx 120$ to ensure that the parties can extract a single unbiased bit from the k pairs of bits in each round.

8.5.2. Single Secret Leader Election

Here, we describe an application of our PCF construction to single secret leader election (SSLE) [BBHP22; CFG23; BEHG20; CFG22; CCMBM24]. The problem of SSLE appears in many distributed systems contexts, and in particular, in proof-of-stake based cryptocurrencies. Informally, SSLE requires a set of parties to secretly agree on a single leader, such that only the elected leader knows that they were elected. Additionally, SSLE requires the leader to be able to prove, at some later point in time, that they were elected. The definition of SSLE is complex and so we refer the reader to Boneh et al. [BEHG20] for the full formalization. Here, we sketch how a PCF for pseudorandom permutations coupled with a NIZK [BFM88] gives us a simple SSLE protocol.

The basic idea is that we can let the leader be decided by having the lowest permutation value (i.e., zero). By the security of the PCF, no party (besides the leader) learns who was elected in the round. Then, the only missing component is the ability for the elected leader to prove to the other parties that they obtained the lowest permutation value (i.e., that the output of $\mathsf{PCF.Eval}$ is zero). This can be achieved by having the output of the setup give all the parties commitments to the set of keys $\{\mathsf{k}_0, \mathsf{k}_1, \mathsf{k}_2\}$, which then allows the i-th party to prove via a NIZK that the output of $\mathsf{PCF.Eval}$ (using key k_i) results in zero.

Overall, the SSLE protocol with n parties proceeds in three steps:

One-time Setup. In the one-time setup, the i-th party is given a PCF key k_i and a commitment to all other parties' PCF keys k_j for $j \in [n]$. The round number r is initialized to zero.

Step 1: Election. In the election step (which is repeated over many rounds), each party runs $\mathsf{PCF.Eval}$ using the round number r as input. The party with the zero output is determined as the leader at round r.

Step 2: Proof. In the proof step, the elected leader (say, party i) generates a NIZK to prove that, relative to the commitment of k_i, the output of $\mathsf{PCF.Eval}$ using key k_i and input r outputs zero.

Step 3: Verification. The parties use the public commitment of k_i to verify the NIZK generated by party i in round r.

Remark 6 (Requirement for a PCF). We crucially rely on the PCF to ensure that the elected leader can produce an efficient NIZK that doesn't grow with the total number of rounds. If a proof is not required, then a PCG would suffice to instantiate SSLE. Moreover, we note that our (biased) PCG construction gives us a perfectly *unbiased* SSLE protocol (even though the permutation itself is biased) since the party with the zero share is pseudorandomly distributed across all three parties.

8.6. Optimizations and Evaluation

In this section, we first describe an optimization to the PCG construction when instantiated using a with an leaky programmable DPF.

8.6.1. Optimizing Programmable PCGs for Biased Bits

We discuss how to optimize the concrete efficiency of the programmable PCG for (1/6)-biased bits described in Section 8.4.3. The construction of Section 8.4.3 is very similar to the original PCG for OLE from QA-SD of Bombar et al. [BCCD23] (adapted to generate shares of a pseudorandom square instead of shares of a pseudorandom product). Thanks to the followup work of Bombar et al. [BBCCDS24a] that introduces a number of algorithmic and low-level optimizations, this PCG was shown to achieve a very good concrete performance over small fields.[4]

The core bottleneck of our construction is its use of the *programmable* DPFs from [BGIK22] instead of the (much faster, non-programmable) DPFs from [BGI16b]: as the authors of [BGIK22. page 5] write themselves, *"our construction [...] cannot offer negligible privacy error ε with good concrete efficiency."* However, the starting point of [BGIK22] is a programmable DPF with noticeable privacy error which *does* achieve good concrete efficiency. We make the following observation, which we believe to be useful beyond the context of the current work:

> *Using the programmable DPF of Boyle et al. [BGIK22] with a large privacy error ε in a PCG construction still results in a secure PCG scheme, at the cost of assuming a stronger flavor of the syndrome decoding assumption.*

We note that replacing standard DPFs with a programmable DPF in PCG constructions has additional benefits that go beyond our target application:

(1) When generating random point functions, the programmable DPF of [BGIK22] admits a simple 2-round key distribution protocol (while the DPF from [BGI16b] requires $\log n$ round for a domain of size n) and

(2) allows one of the key shares to be extremely small (a 128 bit string).

These advantages translate directly to the PCG setting, yielding PCGs featuring a 2-round seed distribution protocol and one optimally-short key. Combined with our observation, this yields a simple approach to build reasonably efficient PCGs (e.g., for OLEs) with these appealing features.

[4] Bombar et al. [BBCCDS24a] report generating 12 millions OLEs over $\mathbb{F}_4$. While our setting is slightly different (we work over $\mathbb{F}_3$ and use programmable DPFs) most of their optimizations carry over directly to our setting.

8.6.1.1. Brief overview of the programmable DPF.

The main building block is a punctuable PRF. A puncturable pseudorandom function (PPRF) [KPTZ13; BW13; BGI14] is a PRF F such that given an input x, and a PRF key k, one can generate a *punctured* key, denoted $k\{x\}$, which allows evaluating F at every point except for x (i.e., there is an algorithm $F.\mathsf{Eval}$ such that $F.\mathsf{Eval}(k\{x\}, x') = F_K(x')$ for all $x' \neq x$), and such that $F_k(x)$ is indistinguishable from random given $k\{x\}$. The seminal tree-based construction of Goldreich et al. [GGM86] over a domain $[n]$ yields a PPRF [BGI14] with key size n and punctured key size $n \cdot \log n$; it admits an efficient 2-round distributed key generation protocol [Ds17].

At a high level, the full-domain evaluation of the PDPF of Boyle et al. [BGIK22] proceeds as follows. Given the target domain size n, let $N = T \cdot n$, for a parameter T to be determined later. Let P_0, P_1 denote two parties who will receive PDPF keys K_0 and K_1 respectively.

- $\mathsf{Gen}_0(1^n)$: Sample a key $K_0 = K$ for a PPRF $F_K : [N] \to [n]$.
- $\mathsf{Gen}_1(y)$: To puncture at a point $y \in [n]$, randomly select an input x such that $F_K(x) = y$ (output failure if there is no such input), and return the key $K_1 = (x, K\{x\})$ punctured at x.
- $\mathsf{Eval}(K_0)$: given K, initialize $\mathbf{u}_0 \leftarrow (0, \ldots, 0)$. For $i = 1$ to N, compute $j \leftarrow F_K(i)$ and add 1 to $u_{0,j}$.
- $\mathsf{Eval}(K_1)$: initialize $\mathbf{u}_1 \leftarrow (0, \ldots, 0)$. For every $i \in [N] \setminus \{x\}$, compute $j \leftarrow F_K(i)$ and add 1 to $u_{1,j}$.

It is easy to verify that $\mathbf{u}_0 - \mathbf{u}_1$ is a unit vector, with a 1 at position y. The scheme has non-negligible privacy error towards P_1, as x leaks a preimage of y.

8.6.1.2. Generating biased random unit vectors via PDPF.

We suggest a simple modification of the above approach, tailored to the setting where one wants to generate succinct shares of a *random* unit vector, as is typically the case in PCG constructions. Consider the following protocol:

- Sample a key $K_0 = K$ for a PPRF $F_K : [N] \to [n]$.
- Sample a *random input* x and return $(x, K\{x\})$.
- Evaluation with K_0: given K, initialize $\mathbf{u}_0 \leftarrow (0, \cdots, 0)$. For $i = 1$ to N, compute $j \leftarrow F_K(i)$ and add 1 to $u_{0,j}$.
- Evaluation with K_1: initialize $\mathbf{u}_1 \leftarrow (0, \cdots, 0)$. For every $i \in [N] \setminus \{x\}$, compute $j \leftarrow F_K(i)$ and add 1 to $u_{1,j}$.

Compared with the construction of Boyle et al. [BGIK22], we do not select an input x mapping to a target y; rather, we select a uniformly random x and hope that it will induce

a sufficiently well-distributed $y = F_K(x)$. This construction does still leak information about the secret unit vector, but the leakage is of a different nature and occurs in the opposite direction. That is, the construction does not leak any information anymore to P_1, which is is easy to see: Given $(x, K\{x\})$, the value $y = F_K(x)$ looks uniformly random to P_1 by the security of the PPRF. However, it provides the following leakage to P_0: P_0 learns that y was sampled from the biased distribution $\mathcal{D}_{\mathbf{u}_0}$ that returns each element i of $[n]$ with probability $u_{0,i}/N$. Indeed, $u_{0,i}$ corresponds exactly to the number of inputs x such that $F_K(x) = i$, hence when sampling a random x from $[N]$, the probability that $F_K(x) = i$ is exactly $u_{0,i}/N$.

While this is a noticeable leakage, in the context of sampling noise vectors to use in a PCG construction, it appears relatively harmless! Indeed, it translates to assuming the hardness of (regular) syndrome decoding where the noise entries of the noise vector have been sampled according to the biased distribution above. As T becomes sufficiently large, by a straightforward Chernoff bound, it is easy to see that the distribution approaches the uniform distribution, and so we expect the resulting variant of the syndrome decoding assumption to remain secure.

8.6.1.3. The randomly-biased syndrome decoding assumption.

We formalize the above observation by introducting the *randomly-biased* syndrome decoding assumption. Given parameters n, T, we let $\mathbf{p} \xleftarrow{\text{R}} \mathsf{W}(n, T)$ denote the procedure that randomly distributes $n \cdot T$ balls into n bins and output the vector $\mathbf{p} = (p_1, \cdots, p_N)$ where p_i denotes the fraction of balls in the i-th bin. Given a weight vector $\mathbf{p}$, we let $\mathcal{D}_{\mathbf{p}}$ denote the distribution over $[n]$ that samples each $i \in [n]$ with probability p_i.

Definition 8.14 (Randomly-Biased Regular Syndrome Decoding Assumption)**.** Let $\mathcal{R}$ be a finite ring, and $(t, k, n) = (t(n), k(n), n(n))$ be polynomials in the security parameter n with $k, t < n$. Let $m \leftarrow n - k$. Let $\mathcal{M} = \mathcal{M}_{n,m}$ denote a distribution over $\mathcal{R}^{m \times n}$. Let $T = T(n)$ denote a polynomial. The T-randomly-biased regular syndrome decoding assumption over $\mathcal{M}$ with dimension k, codeword length n, and noise t, denoted (t, k, n)-$\mathsf{SD}(\mathcal{M})$, states that for every polynomial-time algorithm $\mathcal{A}$, it holds that

$$\left| \Pr\left[b = 1 : \begin{array}{r} (\mathbf{p}_1, \cdots, \mathbf{p}_t) \xleftarrow{\$} \mathsf{W}(n/t, T)^t \\ (\mathbf{e}_1, \cdots, \mathbf{e}_t) \xleftarrow{\$} \mathcal{D}_{\mathbf{p}_1} \times \cdots \times \mathcal{D}_{\mathbf{p}_t} \\ H \xleftarrow{\$} \mathcal{M},\ \mathbf{e} \leftarrow (\mathbf{e}_1 || \cdots || \mathbf{e}_t) \\ b \leftarrow \mathcal{A}(\mathbf{p}_1, \cdots, \mathbf{p}_t, H, H \cdot \mathbf{e}^\intercal) \end{array} \right] - \Pr\left[b = 1 : \begin{array}{r} (\mathbf{p}_1, \cdots, \mathbf{p}_t) \xleftarrow{\$} \mathsf{W}(n/t, T)^t \\ H \xleftarrow{\$} \mathcal{M},\ \mathbf{x} \xleftarrow{\$} \mathcal{R}^m \\ b \leftarrow \mathcal{A}(\mathbf{p}_1, \cdots, \mathbf{p}_t, H, \mathbf{x}) \end{array} \right] \right|$$

is negligible.

The definition above is very general as we formulate the assumption with respect to an arbitrary distribution $\mathcal{M}$ over the parity-check matrices of some linear code over a ring $\mathcal{R}$, and captures in particular the QA-SD assumption used in our work. It is relatively straightforward to show that, when plugging our variant of the PDPF from Boyle et al. [BGIK22] as a replacement for the DPF of [BGI16b] in all existing PCGs and PCFs based on (some variant of) the regular syndrome decoding assumption, the security

D	PCG.Expand	**Key Size** (per party)
2^{18}	4 minutes	0.71 MB
2^{20}	14 minutes	0.77 MB
2^{22}	48 minutes	0.84 MB

Table 8.2.: Performance of our PCG implementation on a `c5.metal` EC2 instance. We set the noise parameter to $t = 16$ and set the compression factor $c = 5$, following the parameter selection script of [BBCCDS24a].

analysis extends immediately under the randomly biased regular syndrome decoding assumption (for a suitable choice of the matrix distribution $\mathcal{M}$).

We leave a concrete cryptanalysis of this new assumption, and a formal reduction to more traditional assumptions, for future work. However, we note that even for relatively low values of T (e.g., $T < 100$), we are not aware of any nontrivial attacks against this assumption instantiated with the same parameters as for the standard regular syndrome decoding assumption. We believe that this could be formalized in the linear test framework of [BCGIKS20a; CRR21].

8.6.2. Implementation and parameters

We implement the biased PCG construction in C by adapting the open-source implementation of the FOLEAGE [BBCCDS24a] PCG for programmable OLE correlations. In particular, we modify their implementation by adapting it to work over $\mathbb{F}_3$ and use their parameter estimation script to derive a new set of parameters tailored to our construction. The parameters c (compression factor) and t (noise weight) influence the size of the PCG key and evaluation time. We set $c = 5$ and $t = 16$, which requires evaluating $(ct)^2 = 2916$ DPFs to compute PCG.Expand. The choice of $(c = 5, t = 16)$ corresponds to a conservative parameter choice based on the parameter estimation script of [BBCCDS24a]. We then replace their use of DPFs with programmable DPFs, to match Figure 8.4. In particular, we use the $T = 40$, which gives us correctness $1 - 2^{-40}$.

Our programmable DPF implementation takes advantage of the `AES-NI` instruction to implement a fast PRG G using fixed-key AES (from the OpenSSL library [The24]) and the Davies-Meyer transform. However, programmable DPFs to not allow for the optimizations exploited in the implementation of the FOLEAGE PCG [BBCCDS24a] and are overall concretely more expensive to evaluate. In particular, the SPFSS (sum of many DPFs) evaluation accounts for 98% of the total computation time of the PCG expansion.

8.6.3. Benchmarks

We run a set of benchmarks using AWS `c5.metal` (3.4GHz CPU). All experiments are averaged across ten trials and evaluated on a single core. We report the benchmark results in Table 8.2. The parameter $D = 2^n$ determines the number of bits we generate in total. Our choice of DPF range 2^{18}, 2^{20}, and 2^{22} correspond to the size of a regular noise block when $D = 2^{22}$, $D = 3^{24}$, and $D = 3^{26}$, respectively. Evaluating the PCG requires $(ct)^2$ calls to the DPF on domain size D/t (using the regular noise optimization described in [BBCCDS24a]). The DPF evaluation cost ends up being the dominant factor (approximately 97% of the total computation). However, we obtain an amortized performance of approximately 1,200 permutations per second.

Part IV.

Weaker Primitives for MPC

9. Structured-Seed Local Pseudorandom Generators and their Applications

9.1. Introduction

Pseudorandom generators (PRGs) are functions mapping n bits to $m(n) > n$ bits such that no polynomial-time algorithm can distinguish their output on a random input from a random m-bit string. Local pseudorandom generators (local PRGs) are pseudorandom generators where every output bit can be computed from a constant number of input bits (that is, they belong to the complexity class NC^0). The existence of local PRGs was first investigated in the work of Cryan and Miltersen [CM01]. The work of Applebaum, Ishai, and Kushilevitz [AIK04; AIK08] showed that pseudorandom generators in NC^0 with sublinear stretch ($m = n + o(n)$) exist under widely believed standard assumption for the case of PRG with sublinear stretch (such as factorization, or discrete logarithm), and under a specific intractability assumption related to the hardness of decoding "sparsely generated" linear codes for the case of PRG with linear stretch $m = \Theta(n)$.

In recent years, the existence of local pseudorandom generators with *polynomial stretch* ($m = n^{1+\varepsilon}$ for some constant $\varepsilon > 0$) has been shown to enjoy a variety of applications, ranging from secure computation with constant computational overhead [IKOS08], indistinguishability obfuscation [JLS21; JLS22], pseudorandom correlation generators and functions [BCGIO17; BCMPR24], public key encryption [BKR23] and sublinear secure computation [BCM23], to applications extending beyond the realm of cryptography such as hardness of learning [DV21]. Consequently, the existence of polynomial-stretch local PRGs and the cryptanalysis of existing candidates has been the subject of many works [Gol00; MST03; BQ09; App12; OW14; CEMT14; App15; ABR16; AL16; LV17; CDMRR18; AK19; OST19; Méa; YGJL21; Méa22; Üna23b; DMR23; Üna23a]. All existing candidates build upon a design originally suggested in [Gol00] that applies a well-chosen predicate P on constant-size subsets of the bits of the seed, where the subsets are chosen to form the hyperedges of a sufficiently expanding uniform hypergraph.

9.1.1. Our contribution

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 [IKOS08];
- Compact homomorphic secret sharing [BCM23];
- Hardness of learning DNFs [DV21].

Beyond introducing structured-seed local PRGs and providing a formal definition, we also introduce constructions of structured-seed local PRGs from well-studied cryptographic assumptions which are not known to imply the existence of standard local PRGs. Concretely, we focus on the *sparse learning parity with noise* assumption, an assumption introduced in the work of Alekhnovich [Ale03] which is equivalent to hardness of decoding random LDPC codes. We provide an extended coverage of various flavors of this assumption and show several constructions of structured-seed local PRGs from these variants. In particular, we obtain:

- A direct construction of structured-seed local PRG from the sparse-LPN assumption with regular noise distribution (where the noise is sampled as a concatenation of random one-hot vector), and
- A construction of structured-seed local PRG with inverse-polynomial security from the sparse-LPN assumption with more common noise distributions. This second construction is more involved and builds upon hashing schemes for balanced allocation.

As a consequence, we show that for the four applications above, the assumption of local PRGs can be replaced by the assumption that sparse-LPN with regular noise is hard (for the application to indistinguishability obfuscation, we require subexponential hardness of the assumption). For the application to hardness of learning, where inverse-polynomial security suffices, we further obtain hardness results from the sparse-LPN assumption without regular noise.

9.1.2. Concurrent work

In a recent work [RVV24], Ragavan, Vafa, and Vaikuntanathan also introduced the notion of structured-seed local pseudorandom generators and studied its application. Our work is concurrent and independent to theirs, and there is a significant overlap between our results: the core observation (that structured-seed local PRGs can replace local PRGs in some applications) and the definition of structured-seed local PRGs are essentially the same in both works. We outline a few differences:

- The work of [RVV24] focuses on the application of structured-seed local PRGs to indistinguishability obfuscation (iO). While we also consider iO, they provide a much more thorough coverage of this application and achieve stronger results (replacing local PRGs with structured-seed local PRG in the work of [JLS22] rather than in the work of [JLS21], hence avoiding the use of the LWE assumption).
- The other applications we consider are not considered in the work of [RVV24]. While our coverage of these applications is (for now) superficial, we plan to include a significantly more extended coverage of the application to hardness of learning in future versions of this work (since several non-trivial complications arise in this setting).
- Eventually, the work of [RVV24] focused on constructions from sparse-LPN with Bernoulli noise. In contrast, we consider other noise distributions, such as regular noise, and XOR noise (where the noise is sampled as a XOR of unit vectors). Consequently, our constructions of structured-seed local PRGs, while sharing a common intuition, differ significantly from theirs.

The current version of our paper is a work in progress. We are posting this working draft on ePrint due to the concurrent work of [RVV24] being online. In future versions of this work, we plan to include additional results, such as

- Exploring further the implications of structured-seed local PRGs for PAC-learning;
- Providing constructions of structured-seed local PRGs with improved parameters (smaller locality for a given stretch) from the Expand-Accumulate LPN assumption from [BCGIKRS22].

9.2. Preliminaries

Vector and Matrix. We denote vectors using bold font and matrices using caps. Given a vector $\mathbf{v}$, we write $\mathbf{v}[i]$ to denote its i-th entry, for a given set $S \in [n]$, we denote $\mathbf{v}[S]$ as a set including i-th entries of $\mathbf{v}$ for all $i \in S$. By default, all vectors are column vectors. We call a length-m one-hot vector as a unit vector over $\mathbb{F}_2^n$. We write $[n]$ to denote the set $\{1, \cdots, n\}$ and $\mathsf{unit}_n(i)$ to denote a unit vector of length n having non-zero i-th entry. For any distribution $\mathcal{D}$, we denote $x \leftarrow \mathcal{D}$ is the process of sampling uniformly x over the distribution $\mathcal{D}$.

Distribution. Given an algorithm $\mathcal{A}$ and a pair of distributions $(\mathcal{D}_0, \mathcal{D}_1)$, we write $\mathsf{Adv}_{\mathcal{A}}(\mathcal{D}_0, \mathcal{D}_1)$ to denote

$$\mathsf{Adv}_{\mathcal{A}}(\mathcal{D}_0, \mathcal{D}_1) = \left| \Pr_{x \leftarrow \mathcal{D}_0}[\mathcal{A}(x) = 0] - \Pr_{x \leftarrow \mathcal{D}_1}[\mathcal{A}(x) = 0] \right|.$$

We say the pair of distributions $(\mathcal{D}_0, \mathcal{D}_1)$ is polynomially indistinguishable and subexponentially indistinguishable if $\mathsf{Adv}_{\mathcal{A}}(\mathcal{D}_0, \mathcal{D}_1) \leq \mathsf{negl}(\lambda)$ for all sufficient large $\lambda \in \mathbb{N}$ and $\mathsf{Adv}_{\mathcal{A}}(\mathcal{D}_0, \mathcal{D}_1) \leq \exp(-\lambda^c)$ for a real number $c > 0$ respectively.

9.2.1. LPN Assumptions

The LPN assumption over the binary field $\mathbb{F}_2$ states, informally, that no adversary can distinguish $(A, A \cdot \mathbf{x} + \mathbf{e})$ from $(A, \mathbf{b})$, where A is a matrix sampled from some matrix distribution $\mathcal{M}$ over $\mathbb{F}_2^{n \times k}$, $\mathbf{x}$ is sampled uniformly from $\mathbb{F}_2^k$, and $\mathbf{e}$ is a *noise vector* sampled from some noise distribution $\mathcal{E}$ over (typically sparse) $\mathbb{F}_2$-vectors. The vector $\mathbf{b}$ is a uniform vector over $\mathbb{F}_2^n$. More formally, we define below the LPN assumption over $\mathbb{F}_2$ with dimension k and n samples, w.r.t. a matrix distribution $\mathcal{M}$ and a noise distribution $\mathcal{D}$:

Definition 9.1 (Learning parity with noise)**.** For any integer $k \in \mathbb{N}$, let $n = n(k)$ be a polynomial, and let $\mathcal{M} = \mathcal{M}_{n,k}$ be a distribution over $\mathbb{F}_2^{n \times k}$. Let $\mathcal{E} = \mathcal{E}_n$ be a noise distribution over $\mathbb{F}_2^n$. Then, we say that the $(\mathcal{M}, \mathcal{E})$-$\mathsf{LPN}$ problem is (T, ε, δ)-hard if for every probabilistic adversary $\mathcal{A} = (\mathcal{A}_n)_{n \in \mathbb{N}}$ of size at most $T = T(n)$, it holds that for all large enough k,

$$\Pr_{A \xleftarrow{\$} \mathcal{M}_{n,k}} \left[\mathsf{Adv}_{\mathcal{A}_k}(\mathcal{D}_0^A, \mathcal{D}_1^A) > \varepsilon \right] \leq \delta,$$

where $\mathcal{D}_0^A$ denotes the distribution $\{(A, A \cdot \mathbf{x} + \mathbf{e}) \mid \mathbf{x} \xleftarrow{\$} \mathbb{F}_2^k, \mathbf{e} \xleftarrow{\$} \mathcal{E}\}$ and $\mathcal{D}_1^A$ denotes $\{(A, \mathbf{b}) \mid \mathbf{b} \xleftarrow{\$} \mathbb{F}_2^n\}$.

9.2.1.1. Types of noise.

Denoting w a parameter which governs the average density of nonzero entries in a random noise vector $\mathbf{e}$, common choices of noise distribution are Bernoulli noise (each entry of $\mathbf{e}$ is sampled from a Bernoulli distribution with parameter w/n), exact noise ($\mathbf{e}$ is uniformly random over the set of vectors of Hamming weight w), and regular noise ($\mathbf{e}$ is a concatenation of w random unit vectors). Another natural choice of noise distribution is to set $\mathbf{e}$ to be the XOR of w random one-hot vectors of length n, as the independence of the one-hot vectors is often convenient in security analysis. We will call respectively *Bernoulli noise*, *exact noise*, *regular noise*, and *xor noise* these standard noise distributions, and denote:

- $\mathcal{B}_{n,w}$ the distribution over $\mathbb{F}_2^n$ where each entry is set to 1 with independent probability w/n;
- $\mathcal{S}_{n,w}$ the uniform distribution over the set of vectors of $\mathbb{F}_2^n$ with Hamming weight w;
- $\mathcal{R}_{n,w}$ the distribution obtained by sampling w one-hot vectors over $\mathbb{F}_2^{n/w}$ (assuming for simplicity that w divides n) and outputting their concatenation;
- $\mathcal{X}_{n,w}$ the distribution obtained by sampling w one-hot vectors over $\mathbb{F}_2^n$ and outputting their XOR.

When n is clear from the context, we drop it from the subscript and write $\mathcal{B}_w, \mathcal{S}_w, \mathcal{R}_w, \mathcal{X}_w$ respectively.

When $w \ll \sqrt{n}$ (the "low-noise" setting), the flavors of LPN with noise sampled from $\mathcal{S}_w$ and $\mathcal{X}_w$ are easily shown to be equivalent, since a random sample from $\mathcal{X}_w$ has Hamming weight exactly w with probability at least $1 - w^2/n$. In turn, LPN with noise sampled from $\mathcal{S}_w$ is known to be equivalent to LPN with noise sampled from $\mathcal{B}_w$ [Pie12]. While there are also reductions between LPN with regular noise and other variants, they typically induce a much larger loss in the parameters [LWYY24].

9.2.1.2. Discussion on the definition.

The reader may observe that Definition 9.1 differs slightly from the standard definition of LPN: the standard definition requires that the adversary should have at most advantage ε (for a negligible $\varepsilon = \varepsilon(n)$) in distinguishing $\{(A, \mathbf{b}) \mid A \xleftarrow{\$} \mathcal{M}_{n,k}, \mathbf{b} \xleftarrow{\$} \mathcal{D}_0^A\}$ from $\{(A, \mathbf{b}) \mid A \xleftarrow{\$} \mathcal{M}_{n,k}, \mathbf{b} \xleftarrow{\$} \mathcal{D}_1^A\}$ (that is, the distinguishing advantage is also quantified over the random choice of A). Definition 9.1 is more fine-grained: it separates the probability δ that the matrix A is a "good matrix" for the LPN problem from the probability ε that the adversary can (given A) distinguish the LPN samples from random. Setting (ε, δ) to be negligible recovers the standard LPN definition. However, this more fine-grained definition style is particularly helpful for the *sparse-LPN* assumption, which we cover in Section 9.4.

9.2.2. Useful Lemmas

Lemma 9.1 (Chernoff Bound)**.** *Let $n \in \mathbb{N}$ and $X_1, \cdots, X_n$ be independent random variables taking values in $\{0, 1\}$. Let X denote their sum and let $\mu \leftarrow \mathbb{E}[X]$. Then for any $\delta \in (0, 1)$,*

$$\Pr[X \geq (1+\delta)\mu] \leq \exp\left(-\frac{\delta^2\mu}{3}\right) \text{ and } \Pr[X \leq (1-\delta)\mu] \leq \exp\left(-\frac{\delta^2\mu}{2}\right).$$

Lemma 9.2 (Markov inequality)**.** *Let X be a positive random variable with finite expectation μ. Then for any $k > 0$, $\Pr[X \geq k] \leq \mu/k$.*

Lemma 9.3 (Piling-up lemma)**.** *For any $0 < r < 1/2$ and any integer N, given N i.i.d. random variables $X_1, \cdots, X_N$ over $\mathbb{F}_2$ with $\Pr[X_i = 1] = r$, it holds that $\Pr[\bigoplus_{i=1}^N X_i = 0] = 1/2 + (1-2r)^n/2$.*

9.3. Defining Structured-Seed Local PRGs

9.3.1. Noisy local circuits

The class NC_0 is the class of functions computable by uniformly generatable boolean circuits with constant depth and bounded fan-in (equivalently, boolean circuits where every output bit depends on a constant number of input bits).

9.3.1.1. The noisy-NC_0 model.

Let $\mathcal{N} = \{\mathcal{N}_n\}_{n\in\mathbb{N}}$ be a family of distribution over $\{0,1\}^n$. We introduce below the notion of $\mathcal{N}$-NC_0 circuits, which are (informally) NC_0 circuits C where the outputs of any gate can be flipped with some probability, where the randomness is picked from the random tape is filled with a sample from $\mathcal{N}_{|C|}$. More formally, we define the model of $\mathcal{N}$-NC_0 circuits as follows:

Definition 9.2. An $\mathcal{N}$-NC_0 circuit $C : \{0,1\}^n \to \{0,1\}^m$ is a bounded fan-in constant-depth circuit where the gates are *noisy*, in the following sense: on input $x \in \{0,1\}^n$, a random sample $r \xleftarrow{\$} \mathcal{N}_{|C|}$ is picked from $\mathcal{N}$. Then, during the evaluation of $C(x)$, the output of each gate g_i is xored to r_i. We denote $C(x : r)$ the result of evaluating the noisy circuit C on input x with randomness r.

Remark 7. The model of noisy circuits differs from that of randomized circuits, which are passed a random input in addition to their regular input, in two respects...

9.3.2. Noisy local PRGs

A cryptographic pseudorandom generator (PRG) is an algorithm $\mathsf{PRG} : \{0,1\}^n \to \{0,1\}^m$, with $n > m$, such that no polynomial-time adversary can distinguish between $\mathsf{PRG}(r)$ (for $r \xleftarrow{\$} \{0,1\}^n$) and a random $z \xleftarrow{\$} \{0,1\}^m$. The uniform string $r \in \{0,1\}^n$ is called the *seed* of the PRG. A pseudorandom generator is *local* if each output bit depends on a constant number of input bits; equivalently, PRG can be computed by an NC_0 circuit.

In the formal definition below and given a noise distribution $\mathcal{N}$, we expand the notion of local pseudorandom generator to also capture *$\mathcal{N}$-local PRGs*, where PRG is a randomized algorithm computable by an $\mathcal{N}$-NC_0 circuit. Formally,

Definition 9.3 (Noisy Local Pseudorandom Generator)**.** Let $m(\cdot)$ be a polynomial, and let $\mathcal{N} = \{\mathcal{N}_n\}_{n\in\mathbb{N}}$ be a family of efficiently samplable distributions over $\{0,1\}^n$. A noisy pseudorandom generator with stretch $m(\cdot)$ is a pair of uniform p.p.t. algorithms $(\mathsf{Setup}, \mathsf{PRG})$, with:

- $\mathsf{Setup}(1^n)$. A probabilistic algorithm that on input 1^n outputs public parameters pp.
- $\mathsf{PRG}(\mathsf{pp}, \mathsf{seed}; r)$. A probabilistic algorithm that on inputs a seed $\mathsf{seed} \in \{0,1\}^n$, public parameters pp, and running on random tape $r \in \mathsf{Supp}(\mathcal{N})$, outputs a string $y \in \{0,1\}^{m(n)}$.

A noisy local PRG must be *local* and *secure*:

Locality. A noisy PRG is *local* if $\mathsf{PRG}(\mathsf{pp}, \cdot)$ is implementable by an $\mathcal{N}$-NC_0 circuit for every pp in the support of $\mathsf{Setup}(1^n)$; we typically think of Setup as an efficient algorithm that samples the noisy local circuit computing $x \mapsto \mathsf{PRG}(\mathsf{pp}, x)$.

Security. A noisy local PRG is (T, ϵ, δ)-*secure* if for any *non-uniform* p.p.t adversary $\mathcal{A} = (\mathcal{A}_n)_{n\in\mathbb{N}}$ of size at most $T = T(n)$, for all $n \in \mathbb{N}$, $\Pr[\mathsf{Adv}_{\mathcal{A}_n}(\mathcal{D}_0, \mathcal{D}_1) > \varepsilon] \leq \delta$, where

for $b \in \{0,1\}$, $\mathcal{D}_b = \mathcal{D}_b^n$ denotes the family of distributions

$$\{(\mathsf{pp}, z_b) \mid \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^n), x \xleftarrow{\$} \{0,1\}^n, r \xleftarrow{\$} \mathcal{N}, z_0 \xleftarrow{\$} \{0,1\}^{m(n)}, z_1 \leftarrow \mathsf{PRG}(\mathsf{pp}, x; r)\}.$$

A noisy pseudorandom generator is (ε, δ)-secure if it is (T, ε, δ)-secure for every $T(n) = \mathsf{poly}(n)$, and that it is γ-secure if it is $(\varepsilon, \gamma - \varepsilon)$-secure for some $\varepsilon \leq \gamma$.

We say that a noisy local PRG has *polynomial stretch* if $m(n) = n^{1+\Omega(1)}$.

9.3.3. Structured-seed local PRGs

In this section, we introduce the notion of structured-seed local pseudorandom generator. A *structured-seed* pseudorandom generator relaxes the standard notion of pseudorandom generator to allow for more general distributions of seeds: instead of sampling r uniformly over $\{0,1\}^n$, we sample it as $r \xleftarrow{\$} \mathsf{SampleSeed}$, where

- (small size) the support $\mathsf{Supp}(\mathsf{SampleSeed})$ of $\mathsf{SampleSeed}$ is contained in $\{0,1\}^n$, and
- (efficiency) the running time of the sampler $\mathsf{SampleSeed}$ is much smaller than m.

We provide a formal definition below.

Definition 9.4 (Structured-Seed Local Pseudorandom Generator)**.** A structured-seed pseudorandom generator with a stretch $m(\cdot)$ is a triple of uniform p.p.t. algorithms $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$, with:

- $\mathsf{Setup}(1^n)$. A probabilistic algorithm that on inputs 1^n and outputs a public parameter pp.
- $\mathsf{SampleSeed}(\mathsf{pp})$. A probabilistic algorithm that on inputs pp and outputs a seed value $\mathsf{seed} \in \{0,1\}^n$.
- $\mathsf{PRG}(\mathsf{pp}, \mathsf{seed})$. A deterministic algorithm that on inputs seed, public parameter pp and outputs an evaluation value $y \in \{0,1\}^{m(n)}$.

A structured-seed pseudorandom generator is (T, ϵ, δ)-*secure* if for any *non-uniform* p.p.t adversary $\mathcal{A} = (\mathcal{A}_n)_{n \in \mathbb{N}}$ of size at most $T = T(n)$, for all $n \in \mathbb{N}$,

$$\Pr\left[\mathsf{Adv}_{\mathcal{A}_n}(\mathcal{D}_0, \mathcal{D}_1) > \varepsilon\right] \leq \delta,$$

where $\mathcal{D}_0 = \mathcal{D}_0^{n,n}$ denotes the family of distributions

$$\{(\mathsf{pp}, \mathsf{PRG}(\mathsf{pp}, r)) \mid \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^n), r \xleftarrow{\$} \mathsf{SampleSeed}(\mathsf{pp})\}$$

and $\mathcal{D}_1 = \mathcal{D}_1^{n,n}$ denotes the family of distributions

$$\{(\mathsf{pp}, z) \mid \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^n), z \xleftarrow{\$} \{0,1\}^{m(n)}\}.$$

Furthermore, a structured-seed PRG is said to be in NC^0, or *local*, if PRG is implementable by a uniformly efficiently generatable NC^0 circuit. We say that a structured-seed PRG has *polynomial stretch* if $m(n) = n^{1+\Omega(1)}$. Denoting $\mathsf{T_{ss}} = \mathsf{T_{ss}}(n, n, m)$ the (worst-case) running time of $\mathsf{SampleSeed}$ (implemented by a uniformly efficiently generatable family of boolean circuits), we say that a structured-seed PRG has *strong polynomial stretch* if $m(n) = \mathsf{T_{ss}}^{1+\Omega(1)}$.

We note that if a structured-seed PRG with input size n and stretch m is local, then there exists a locality parameter l such that for all $i \in [m]$, there exists a subset $S_i \subset [n]$ of size $|S_i| \leq l$ and a predicate P_i such that for all $x \in \mathsf{Supp}(\mathsf{SampleSeed}(\mathsf{pp}))$, defining $y = \mathsf{PRG}(x)$, we have $y_i = P_i(x[S_i])$.

Eventually, we consider a further relaxation of structured-seed local PRGs where we allow Setup to be inefficient:

Definition 9.5 (Non-Uniform Structured-Seed Local PRG)**.** We say that $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ is a *non-uniform* structured-seed local pseudorandom generator if it satisfies the conditions of Definition 9.4, except that $\mathsf{Setup}(1^n)$ is not required to run in polynomial time.

9.3.4. From weak to strong local PRGs

Theorem 9.4 ([AK19], Theorem 2.12)**.** *For every constants $d \in \mathbb{N}$, $a > 0$, and $c, c' > 1$, there exists a constant d' for which the following holds. Any ensemble of ε-secure d-local PRGs $G : \{0,1\}^n \to \{0,1\}^{n^c}$ with $\varepsilon = 1/n^a$ can be converted into an ensemble of $\mathsf{negl}(n)$-secure d'-local PRGs $G' : \{0,1\}^n \to \{0,1\}^{n^{c'}}$.*

9.4. The Sparse-LPN Assumption

9.4.1. The sparse-LPN assumption

In this work, we will mostly focus on a variant introduced in [Ale03] (commonly called the "Alekhnovich assumption" or "sparse-LPN assumption"), where $\mathcal{M}$ is a distribution of *sparse* matrices:

Notation 1. We denote by $\mathcal{W}^c = \mathcal{W}^c_{n,k}$ the distribution over $\mathbb{F}_2^{n \times k}$ that samples independently each row $\mathbf{r}$ of A as $\mathbf{r}^\intercal \xleftarrow{\$} \mathcal{S}_{c,k}$; that is, A is a uniformly random matrix with row-weight c over $\mathbb{F}_2^{n \times k}$.

With these notations, the Aleknovich assumption asserts that for a suitable constant $c \geq 3$, the $(\mathcal{M}^c, \mathcal{S}_w, \mathbb{F}_2)$-$\mathsf{LPN}(k, n)$ assumption holds.

It is not hard to see that δ cannot be negligible for sparse-LPN: with probability at least n^2/k^c, two rows $\mathbf{a}_i, \mathbf{a}_j$ of A will be identical, in which case there is a trivial distinguisher (as $\langle \mathbf{a}_i, \mathbf{x} \rangle + \mathbf{e}[i]$ is very likely to be equal to $\langle \mathbf{a}_j, \mathbf{x} \rangle + \mathbf{e}[j]$ by the sparsity of the noise). In fact, any small set of linearly-dependent rows of A yields a nontrivial distinguisher. Therefore, the standard formulation of sparse-LPN asserts that the $(\mathcal{M}^c, \mathcal{S}_w, \mathbb{F}_2)$-$\mathsf{LPN}(k, n)$ is $(\mathsf{poly}(n), \mathsf{negl}(n), \delta)$-hard for a suitable inverse-polynomial δ.

When A does not have a small set of linearly-dependent rows, the best-known attacks on sparse-LPN are the same as the best-known attacks on LPN. This is best explained through the framework of *linear tests*, a framework to heuristically analyze the hardness of variants of LPN which has roots in the works of Naor and Naor [NN90] and Mossel, Shpilka, and Trevisan [MST03], and which was explicitly put forth and stuied in the context of LPN in [ADINZ17; BCGIKS20a; CRR21]. The central observation of this framework is that most known attacks against LPN (such as BKW [BKW00; Lyu05], ISD [Pra62], and many more) share a common template, and that to defeat all attacks sharing this template, it suffices (in coding theoretic terms) that the *dual distance* of the code generated by A is large – *i.e.*, that A does not have a small set of linearly-dependent rows.

9.4.2. Security against linear tests

We briefly overview the linear test framework, and derive from the framework a concrete set of parameters for the sparse-LPN assumption. We stress that the framework is only a heuristic: there are known settings in which a variant of LPN can be broken by an attack that does not fit in the framework (see *e.g.* the discussions in [CRR21; BCCD23]). Nevertheless, the bounds provided by this framework are in line with the state-of-the-art cryptanalysis on sparse LPN, and provides a convenient heuristic to choose plausible parameters.

Notation 2. We call *dual distance* of a matrix M, and write $\mathsf{dd}(M)$, the largest integer d such that every subset of d rows of M is linearly independent.

Informally, the linear test framework models attacks where the adversary is unbounded and can arbitrarily use the LPN matrix A, but is restricted to computing a linear function of the vector $\mathbf{b}$. Concretely, let $\mathcal{A}$ be a (possibly unbounded) adversary. The attack proceeds in two stages:

1. $\mathcal{A}$ receives the LPN matrix A and outputs a nonzero *test vector* $\mathbf{v}$. Note that $\mathcal{A}$ can run in unbounded time, but sees only the matrix A.

2. In the second stage, the vector $\mathbf{b} \leftarrow A \cdot \mathbf{x} + \mathbf{e}$ is sampled. We say that $\mathcal{A}$ is successful if, with large probability over the random choice of A, the *bias* of the distribution induced by sampling $\mathbf{x}$ and the noise $\mathbf{e}$ and computing $\langle \mathbf{v}, A\mathbf{x} + \mathbf{e} \rangle$ (that is, evaluating the linear function picked by $\mathcal{A}$ on $\mathbf{b}$) is noticeable.

To formally state the definition, we recall the notion of bias of a distribution:

Definition 9.6 (Bias of a Distribution)**.** Given a distribution $\mathcal{D}$ over $\mathbb{F}_2^n$ and a nonzero vector $\mathbf{u} \in \mathbb{F}_2^n$, the bias of $\mathcal{D}$ with respect to $\mathbf{u}$, denoted $\mathsf{bias}_{\mathbf{u}}(\mathcal{D})$, is equal to

$$\mathsf{bias}_{\mathbf{u}}(\mathcal{D}) = \left| \mathop{\mathbb{E}}_{\mathbf{x} \leftarrow \mathcal{D}} [\mathbf{u}^\intercal \cdot \mathbf{x}] - \mathop{\mathbb{E}}_{\mathbf{x} \xleftarrow{\$} \mathbb{F}_2^n} [\mathbf{u}^\intercal \cdot \mathbf{x}] \right| = \left| \Pr_{\mathbf{x} \leftarrow \mathcal{D}} [\mathbf{u}^\intercal \cdot \mathbf{x} = 1] - \frac{1}{2} \right|.$$

The bias of $\mathcal{D}$, denoted $\mathsf{bias}(\mathcal{D})$, is the maximum bias of $\mathcal{D}$ with respect to any nonzero vector $\mathbf{u}$.

Definition 9.7 (Security against Linear Test)**.** For any integer $k \in \mathbb{N}$, let $n = n(k)$ be a polynomial, and let $\mathcal{M} = \mathcal{M}_{n,k}$ be a distribution over $\mathbb{F}_2^{n\times k}$. Let $\mathcal{E} = \mathcal{E}_n$ be a noise distribution over $\mathbb{F}_2^n$. Then, we say that the $(\mathcal{M}, \mathcal{E})$-LPN problem is (ε, δ)-*secure against linear tests* if if for any (possibly inefficient) adversary $\mathcal{A}$ which, on input a matrix $A \in \mathbb{F}_2^{n\times k}$, outputs a nonzero $\mathbf{v} \in \mathbb{F}_2^n$, it holds that

$$\Pr[A \xleftarrow{\$} \mathcal{M}, \mathbf{v} \xleftarrow{\$} \mathcal{A}(A) \ : \ \mathsf{bias}_{\mathbf{v}}(\mathcal{D}^A) \geq \varepsilon(n)] \leq \delta(n),$$

where $\mathcal{D}^A$ denotes the distribution induced by sampling $\mathbf{x} \xleftarrow{\$} \mathbb{F}_2^m$, $\mathbf{e} \leftarrow \mathcal{E}_n$, and outputting the LPN samples $A \cdot \mathbf{x} + \mathbf{e}$.

The following observation is folklore, and was made explicitly *e.g.* in [BCGIKS20a]:

Observation 1. Most existing attacks against LPN, including BKW [BKW00; Lyu05], ISD [Pra62], variants of Gaussian elimination [LF06; EKM17], statistical decoding attacks [Ove06], generalized birthday attacks [Wag02; Kir11], linearization attacks [BM97; Saa07], or attacks based on finding correlations with low-degree polynomials [ABGKR14; BR17], can be cast as instances of the linear test framework. Therefore, none of these attacks can provide a polynomial-time distinguisher against any LPN assumption that is provably (ε, δ)-secure against linear tests for negligible functions (ε, δ).

Given any test vector $\mathbf{v}$, observe that $\langle \mathbf{v}, A\mathbf{x}+\mathbf{e}\rangle = \langle \mathbf{v}, A\mathbf{x}\rangle + \langle \mathbf{v}, \mathbf{e}\rangle$. We recall a simple folklore observation, rooted in [NN90; MST03]: the distribution induced by $A \cdot \mathbf{x}$ is $\mathsf{dd}(A)$-wise independent by definition of $\mathsf{dd}(A)$. Hence, the bias of $\langle \mathbf{v}, A\mathbf{x}+\mathbf{e}\rangle = \langle \mathbf{v}, A\mathbf{x}\rangle + \langle \mathbf{v}, \mathbf{e}\rangle$ is zero if the Hamming weight of $\mathbf{v}$ is less than $\mathsf{dd}(A)$. But if $\mathsf{HW}(\mathbf{v}) \geq \mathsf{dd}(A)$, then $\langle \mathbf{v}, \mathbf{e}\rangle$ has low bias, because $\mathbf{e}$ "hits" a nonzero entry of $\mathbf{v}$ with large probability. Formally:

Lemma 9.5. *Let $\mathcal{M} = \mathcal{M}_{n,k}$ be a distribution over $\mathbb{F}_2^{n\times k}$ and $\mathcal{E}_n$ denote a noise distribution over $\mathbb{F}_2^n$. Then for any $d \in \mathbb{N}$, the the $(\mathcal{M}, \mathcal{E})$-LPN problem with dimension $k = k(n)$ and $n = n(n)$ samples is $(\varepsilon_d, \delta_d)$-secure against linear tests, where*

$$\varepsilon_d = \max_{\mathsf{HW}(\mathbf{v})>d} \mathsf{bias}_{\mathbf{v}}(\mathcal{E}_n), \qquad \textit{and} \qquad \delta_d = \Pr_{A\xleftarrow{\$}\mathcal{M}}[\mathsf{dd}(A) < d].$$

The quantity ε_d in Lemma 9.5 depends solely on the noise distribution and can be computed easily for standard types of noise:

Lemma 9.6. *For any integer d and noise distribution $\mathcal{E}_n$, we denote $\varepsilon_d(\mathcal{E}_n) = \max_{\mathsf{HW}(\mathbf{v})>d} \mathsf{bias}_{\mathbf{v}}(\mathcal{E}_n)$. Then*

- $\varepsilon_d(\mathcal{X}_{n,w}) \leq (1 - 2(d+1)/n)^w/2 \leq \exp(-2(d+1)w/n)/2$
- $\varepsilon_d(\mathcal{R}_{n,w}) \leq (1 - 2(d+1)/n)^w/2 \leq \exp(-2(d+1)w/n)/2$
- $\varepsilon_d(\mathcal{B}_{n,w}) \leq (1 - 2w/n)^{d+1}/2 \leq \exp(-2(d+1)w/n)/2$

The proof of the claim is a straightforward application of the piling-up lemma (Lemma 9.3): for $\mathcal{X}_{n,w}$ or $\mathcal{R}_{n,w}$, the distribution induced by $\langle \mathbf{v}, \mathbf{e}\rangle$ is a xor of w independent Bernoulli samples, each equal to 1 with probability $\mathsf{HW}(\mathbf{v})/n < d/n$. For $\mathcal{B}_{n,w}$, $\langle \mathbf{v}, \mathbf{e}\rangle$ is a xor of $\mathsf{HW}(\mathbf{v}) > d$ Bernoulli sample of rate w/n. The rightmost side of the inequalities follows from the standard inequality $(1 - 1/N)^N \leq \exp(-1)$. We note that a similar bound can be shown (with a slightly more tedious analysis) for $\mathcal{S}_{n,w}$.

9.4.3. The dual distance of random sparse matrices

We recall below a standard bound on the probability that random sparse matrices have large dual distance.

Theorem 9.7 (Most sparse matrices have large dual distance)**.** *For any constants $c \geq 3$ and $\eta \in (0,1)$, for any large enough $k = k(n)$, there is a constant $\gamma(c)$ such that for any $n \leq k^{(1-\eta)c/2+\eta}$,*

$$\Pr\left[A \xleftarrow{\$} \mathcal{W}^c \ : \ \mathsf{dd}(A) \geq \frac{k^\eta}{\gamma(c)} - 1\right] \geq 1 - \left(\frac{\gamma(c)}{k^\eta}\right)^{c-2}.$$

For example, setting $c = 3$ and $\eta = 1/5$, Theorem 9.7 yields that for $n = k^{1.4}$, a random c-sparse matrix A over $\mathbb{F}_2^{n\times k}$ has dual distance $\mathsf{dd}(A) = \Omega(k^{0.2})$ with probability at least $1 - O(k^{-0.2})$. For completeness, we provide the proof which is a direct adaptation (and slight generalization) of the analysis of [MST03. Section 5.3]. Part of the proof is taken essentially verbatim from [CRR21], and adapted to our parameter setting. Near-identical proofs of essentially the same theorems can be found in other works, *e.g.* [BCGIKRS23; DIJL23b].

Proof. Given a matrix $A \in \mathbb{F}_2^{n\times k}$, we denote by $(\mathbf{a}_1, \cdots, \mathbf{a}_n)$ the rows of A, and by $(\mathbf{a}^1, \cdots, \mathbf{a}^k)$ its columns. For any subset $S \subseteq [n]$ of the rows of A, we call *column-neighbors of S in A*, and write $\mathsf{N}^{\mathsf{c}}_A(S) \subseteq [k]$, the subset of the columns of A which have at least one 1 in a row of S. For any integer d, we say that A has the *d-unique column-neighbor property* if for any set $S \subseteq [n]$ with $|S| \leq d$, there exists a column A^j of A such that $A^j[S]$ contains exactly a single 1.

Lemma 9.8 ([MST03])**.** *If $A \in \mathbb{F}_2^{n\times k}$ has d-unique column-neighbor, then $\mathsf{dd}(A) \geq d - 1$.*

Proof. For any subset S of $[n]$ with $|S| \leq d$, there exists a column of A that has exactly one 1 in a row indexed by S, hence $\bigoplus_{i\in S} \mathbf{a}_i \neq 0^k$. □

In the following, we show that a randomly sampled sparse matrix A has d-right unique column-neighbor with high probability, for a certain d. This follows from the fact that A has certain expansion properties. We say that a matrix A is (d,α)-expanding if for every subset $S \subseteq [n]$ with $|S| \leq d$, it holds that $|\mathsf{N}_A(S)| > \alpha \cdot |S|$ (in other words, a matrix A is (d,α)-expanding if it is the adjacency matrix of a (d,α)-expanding bipartite graph).

Lemma 9.9 ([MST03])**.** *Let $A \in \mathsf{Supp}(\mathcal{W}^c)$ be a c-sparse matrix. If A is $(d, c/2)$-expanding, then it has d-unique column-neighbor.*

Proof. Assume that A does not have d-unique column-neighbor. Let $S \subseteq [n]$ be any subset with $|S| \leq d$. Then for every $j \in \mathsf{N}^{\mathsf{c}}_A(S)$, $\mathbf{a}^j[S]$ contains at least two 1's. Since the rows in S contain $c \cdot |S|$ 1's in total, this implies that $|\mathsf{N}^{\mathsf{c}}_A(S)| \leq (c/2) \cdot |S|$. □

To prove Theorem 9.7, it remains to show that a random sample from $\mathcal{W}^c$ is sufficiently expanding with high probability.

Lemma 9.10. *For any large enough $k = k(\lambda)$, any constants $c \geq 3$ and $\eta > 0$, there is a constant $\gamma(c)$ such that for any $n \leq k^{(1-\eta)c/2+\eta}$,*

$$\Pr\left[A \xleftarrow{\$} \mathcal{W}^c \ : \ A \text{ is } \left(\frac{k^\eta}{\gamma(c)}, c/2\right)\text{-expanding}\right] \geq 1 - \left(\frac{\gamma(c)}{k^\eta}\right)^{c-2}.$$

Proof. For any subset $S \subset [n]$ and any size $c \cdot |S|/2$ subset $T \subset [k]$, the probability over the random choice of $A \xleftarrow{\$} \mathcal{W}^c$ that $\mathsf{N}^{\mathsf{c}}_A(S) \subseteq T$ is at most $(c \cdot |S|/2k)^{c\cdot|S|}$. Since there are $\binom{n}{|S|}$ choices for S and $\binom{k}{c|S|/2}$ choices for T, the probability that A fails to be $(d, c/2)$-expanding is at most

$$\sum_{i=2}^{d} \binom{n}{i} \cdot \binom{k}{c \cdot i/2} \cdot \left(\frac{c \cdot i}{2k}\right)^{c\cdot i},$$

where the sum starts at 2 because any single row j always satisfies $\mathsf{N}^{\mathsf{c}}_A(\{j\}) = c > c/2$. Using the inequality $\binom{a}{b} \leq (ae/b)^b$, we get

$$\begin{aligned}
\sum_{i=2}^{d} \left(\frac{en}{i}\right)^i \cdot \left(\frac{2ek}{ci}\right)^{ci/2} \cdot \left(\frac{ci}{2k}\right)^{ci} &= \sum_{i=2}^{d} \left(\frac{en}{i} \cdot \left(\frac{2ek}{ci}\right)^{c/2} \cdot \left(\frac{ci}{2k}\right)^{c}\right)^i \\
&= \sum_{i=2}^{d} \left(\frac{n}{k} \cdot e^{c/2+1} \cdot (c/2)^{c/2} \cdot \left(\frac{i}{k}\right)^{c/2-1}\right)^i \\
&= \sum_{i=2}^{d} \left(\left(\frac{n}{k}\right)^{\frac{1}{c/2-1}} \cdot \left(\frac{c}{2}\right)^{\frac{c/2}{c/2-1}} \cdot e^{\frac{c/2+1}{c/2-1}} \cdot \frac{i}{k}\right)^{i\cdot(c/2-1)} \\
&\leq \sum_{i=2}^{d} \left(\gamma(c) \cdot \frac{i}{2k^\eta}\right)^{i\cdot(c/2-1)},
\end{aligned}$$

where $\gamma(c)$ denotes the constant $2 \cdot (c/2)^{\frac{c/2}{c/2-1}} \cdot e^{\frac{c/2+1}{c/2-1}}$. Now, setting $d = k^2/(\gamma(c) \cdot n)$, the above is upper bounded by

$$\left(\frac{\gamma(c)}{k^\eta}\right)^{c-2} + \left(\frac{3\gamma(c)}{2k^\eta}\right)^{3(c/2-1)} + \log^2 k \cdot \left(\frac{\gamma(c)^2 \log^2 k}{k^\eta}\right)^{2c-4} + d \cdot \left(\frac{1}{k^{\log k}}\right)^{c/2-1},$$

where the first two terms are the terms for $i = 2, 3$ in the sum, the third term bounds the terms $i = 4, \cdots, \log^2 k$ in the sum, and the last term bounds the sum of the remaining terms. For a large enough k and any integer $c > 2$, this sum is therefore dominated by its first term. The lemma follows. □

□

9.4.4. A parametrized version of the sparse-LPN assumption

Combining Lemma 9.6 (bounding ε_d) and Theorem 9.7 (to bound δ_d) yields a quantified estimate of the security of sparse-LPN against attacks from the linear test framework:

Lemma 9.11. *For any constants $c \geq 3$ and $\eta \in (0,1)$, for any noise distribution $\mathcal{E} \in \{\mathcal{X}_{n,w}, \mathcal{R}_{n,w}, \mathcal{B}_{n,w}\}$, the $(\mathcal{S}^c, \mathcal{E})$-LPN problem with dimension $k = k(n)$, noise $w = w(n)$, and $n = n(k) \leq k^{(1-\eta)c/2+\eta}$ samples is (ε, δ)-secure against linear tests, where*

$$\varepsilon = \frac{1}{2} \cdot \exp\left(-2\frac{w \cdot k^\eta}{\gamma(c) \cdot n}\right), \qquad \delta = \left(\frac{\gamma(c)}{k^\eta}\right)^{c-2},$$

for a constant $\gamma(c) = 2 \cdot (c/2)^{\frac{c/2}{c/2-1}} \cdot e^{\frac{c/2+1}{c/2-1}}$.

For example, $\gamma(3) \approx 1000$, $\gamma(7) \approx 70$, and $\gamma(10) \approx 67$ (note however that no effort has been made to optimize the constant, and the analysis is very loose). We are now ready to state a concrete parametrized version of the sparse-LPN assumption; the assumption basically states that there is no attack on sparse LPN that does significantly better than linear tests:

Assumption 1. For any constants $c \geq 3$ and $\eta \in (0,1)$, for any noise distribution $\mathcal{E} \in \{\mathcal{X}_{n,w}, \mathcal{R}_{n,w}, \mathcal{B}_{n,w}\}$, for every $T = 2^{o(n)}$, the $(\mathcal{S}^c, \mathcal{E})$-LPN problem with dimension $k = k(n)$, number of samples $n = n(k) \leq k^{(1-\eta)c/2+\eta}$, and noise $w = w(n,k) \geq n \cdot k^{(1-\eta)c/2}$ is (T, ε, δ)-secure, with

$$\varepsilon = \frac{\mathsf{poly}(T)}{2^{\Omega(n)}}, \qquad \delta = \frac{1}{\Omega(k^\eta)}.$$

We note that variants of the concrete assumption above have appeared on multiple occasions in the literature: [ADINZ17] makes a very similar assumption (Assumption 6 in [ADINZ17]) but for a fixed matrix A, and in the constant rate setting (saying that any circuit of size $T = \exp(\Omega_r(\mathsf{dd}(A)))$ has advantage at most $1/T$ against sparse LPN with noise rate $r = w/n$, where r is treated as a constant and $\Omega_r(\cdot)$ hides the dependency in r). [BCGIKRS23] also makes a very similar assumption (though they again only require the existence of a matrix A with large enough dual distance). Below, we provide two specific parameter settings consistent with the requirements of Assumption 1 that we will use in this work:

Parameter Set 1 (balancing (k, w)). For every constant $\eta \in (0,1)$, for any noise distribution $\mathcal{E} \in \{\mathcal{X}_{n,w}, \mathcal{R}_{n,w}, \mathcal{B}_{n,w}\}$, for every $T = 2^{o(n)}$, there is a constant $c(\eta) = 2/(1-\eta)$ such that the $(\mathcal{S}^c, \mathcal{E})$-LPN problem with dimension k, number of samples $n = k^{1+\eta}$, and noise $w = n \cdot k$ is $(T, 2^{-\Omega(n)}, 1/\Omega(k^\eta))$-secure.

Parameter Set 2 (minimizing w). For every constants $\gamma > 0$, for any noise distribution $\mathcal{E} \in \{\mathcal{X}_{n,w}, \mathcal{R}_{n,w}, \mathcal{B}_{n,w}\}$, for every $T = 2^{o(n)}$ and every $c \geq 2\log\gamma/(\log\gamma - 1)$, the $(\mathcal{S}^c, \mathcal{E})$-LPN problem with dimension k, number of samples $n = k^{1+\gamma/2}$ samples, and noise $w = n \cdot k^\gamma$ is $(T, 2^{-\Omega(n)}, 1/\Omega(k^{1-\gamma/2}))$-secure.

9.4.5. Amplifying advantage

Let $t = t(n, \varepsilon, \delta) \leftarrow n/(\delta\varepsilon^2)$. We prove below that, up to a $\mathsf{poly}(n, 1/\varepsilon, 1/\delta)$ loss in the runtime and number of samples, the (ε, δ)-hardness of sparse-LPN implies its $(\exp(-\Omega(n)), \exp(-\Omega(n)))$-hardness.

Lemma 9.12. *Assume that the $(\mathcal{S}^c, \mathcal{X}_{n,w})$-$\mathsf{LPN}$ problem with dimension $k = k(n)$, number of samples $n = k^{(1-\eta)c/2+\eta}$, and noise $w = n \cdot k^{(1-\eta)c/2}$ is $(T', \varepsilon, \delta)$-secure. Then the $(\mathcal{S}^c, \mathcal{X}_{n',w'})$-$\mathsf{LPN}$ problem with dimension k, number of samples $n' = n \cdot t$, and noise $w' = w \cdot t$ is $(T, \exp(-\Omega(n)), \exp(-\Omega(n)))$-secure, with $T' = \mathsf{poly}(T, n, 1/\varepsilon, 1/\delta)$.*

Proof. Let $\mathcal{A}$ be an algorithm running in time T solving the $(\mathcal{S}^c, \mathcal{X}_{n,w})$-$\mathsf{LPN}$ problem with dimension k, number of samples n, and noise w, such that $\Pr_A[\mathsf{Adv}_{\mathcal{A}}(\mathcal{D}_0^A, \mathcal{D}_1^A) > \varepsilon] > \delta$. Set $t \leftarrow n/(\delta\varepsilon^2)$. We show how to construct from $\mathcal{A}$ an algorithm $\mathcal{B}$ which is given black-box access to $\mathcal{A}$ and $(\mathsf{poly}(n, T, 1/\varepsilon, 1/\delta), 1 - \exp(-\Omega(n)), 1 - \exp(-\Omega(n)))$-solves the $(\mathcal{S}^c, \mathcal{X}_{n,w})$-$\mathsf{LPN}$ problem with dimension k, number of samples $n' = n \cdot t$, an noise $w' = w \cdot t$. We denote $\mathsf{Good}_\mu^{\mathcal{A}}$ the set of all matrices $A \in \mathbb{F}_2^{n \times k}$ such that $\mathsf{Adv}_{\mathcal{A}}(\mathcal{D}_0^A, \mathcal{D}_1^A) > \mu$.

On input a $(\mathcal{S}^c, \mathcal{X}_{n,w})$-$\mathsf{LPN}$ instance $(A, \mathbf{b}) \in \mathbb{F}_2^{n' \times k} \times \mathbb{F}_2^{n'}$, let c denote the index of the distribution from which $(A, \mathbf{b})$ was sampled (*i.e.* $c = 0$ if $\mathbf{b}$ is an LPN sample, and $c = 1$ if $\mathbf{b}$ is uniform). $\mathcal{B}$ proceeds as follows:

- Break $(A, \mathbf{b})$ into $n'/n = t$ smaller instances $(A_i, \mathbf{b}_i)_{i \le t}$ with $(A_i, \mathbf{b}_i) \in \mathbb{F}_2^{k \times n}$.
- Set $S \leftarrow \emptyset$. For each $i \le t$, test whether $A_i \in \mathsf{Good}_\varepsilon^{\mathcal{A}}$ as follows:
  - Repeat $\theta = 32n/\varepsilon^2$ times the following procedure: set $\mathsf{ctr} \leftarrow 0$. Sample a bit $\tilde{\sigma} \xleftarrow{\$} \{0,1\}$. If $\tilde{\sigma} = 0$, sample $(\mathbf{x}, \mathbf{e}) \xleftarrow{\$} \mathbb{F}_2^k \times \mathcal{X}_{n,w}|$ and set $\tilde{\mathbf{b}} \leftarrow A_i\mathbf{x} + \mathbf{e}$. If $\tilde{\sigma} = 1$, set $\tilde{\mathbf{b}} \xleftarrow{\$} \mathbb{F}_2^n$. If $\mathcal{A}(A_i, \tilde{\mathbf{b}}) = \tilde{\sigma}$, set $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$.
  - If $\mathsf{ctr} \ge \theta \cdot (1/2 + \varepsilon/4)$, declare A_i to be "good" and add i to S.
  - If $n/(2\varepsilon^2)$ matrices have been declared "good" (i.e., $|S| = n/(2\varepsilon^2)$), **break**.
- Set $B \leftarrow |S| \cdot (1/2 - \varepsilon/16)$. For each good A_i:
  - Sample $\mathbf{x}_i \xleftarrow{\$} \mathbb{F}_2^k$, $\mathbf{b}_i^1 \xleftarrow{\$} \mathbb{F}_2^n$, and set $\mathbf{b}_i^0 \leftarrow \mathbf{b}_i \oplus A_i\mathbf{x}_i$.
  - Flip a coin $c_i \xleftarrow{\$} \{0,1\}$ and compute $\sigma_i \leftarrow \mathcal{A}(A_i, \mathbf{b}_i^{c_i})$.
  - Output 1 if $\sum_{i \in S} \sigma_i \oplus c_i \le B$.

We prove that $\mathcal{B}$ is successful. We use a sequence of simple claims:

Claim 9.13. *If $A_i \in \mathsf{Good}_\varepsilon^{\mathcal{A}}$, then $\Pr[\mathcal{B}$ declares A_i good$] \ge 1 - \exp(-n)$.*

Proof. If $A_i \in \mathsf{Good}_\varepsilon^{\mathcal{A}}$, then $\Pr[\mathcal{A}(A_i, \tilde{\mathbf{b}}) = b] \ge 1/2 + \varepsilon/2$ by definition. It follows that $\mathbb{E}[\mathsf{ctr}] \ge (1/2 + \varepsilon/2) \cdot 32n/\varepsilon^2$. By a Chernoff bound 9.1,

$$\Pr[\mathsf{ctr} < (1/2 + \varepsilon/4) \cdot 32n/\varepsilon^2] < \exp\left(-32 \cdot \left(\frac{\varepsilon/2}{1+\varepsilon}\right)^2 \cdot \frac{n(1+\varepsilon)}{4\varepsilon^2}\right) \le \exp(-n).$$

□ □

Claim 9.14. *Let $S \subseteq [n'/n]$ be the subset of indices i such that $A_i \in \mathsf{Good}^{\mathcal{A}}_{\varepsilon}$. Then* $\Pr[|S| \leq n/(2\varepsilon^2)] \leq \exp(-n/8)$.

Proof. The A_i are sampled uniformly and independently from $\mathbb{F}_2^{n\times k}$, and $\Pr[A_i \in \mathsf{Good}^{\mathcal{A}}_{\varepsilon}] > \delta$ by assumption, hence $\mathbb{E}[|S|] > \delta \cdot n'/n = n/\varepsilon^2$. By a Chernoff bound, it follows that

$$\Pr[|S| \leq n/(2\varepsilon^2)] \leq \exp\left(-\left(\frac{1}{2}\right)^2 \frac{n}{2\varepsilon^2}\right) < \exp(-n/8).$$

□ □

Combining these two claims, it follows that $\mathcal{B}$ will declare at least $n/(2\varepsilon^2)$ matrices A_i to be good, except with probability at most $\exp(-n) + \exp(-n/8)$.

Claim 9.15. *If $A_i \notin \mathsf{Good}^{\mathcal{A}}_{\varepsilon/4}$, then $\Pr[\mathcal{B}$ declares A_i good$] \leq \exp(-4n/15)$.*

Proof. If $A_i \notin \mathsf{Good}^{\mathcal{A}}_{\varepsilon/4}$, then $\Pr[\mathcal{A}(A_i, \tilde{\mathbf{b}}) = b] \leq 1/2 + \varepsilon/8$ by definition. It follows that $\mathbb{E}[\mathsf{ctr}] \leq (1/2 + \varepsilon/8) \cdot 32n/\varepsilon^2$. By a Chernoff bound 9.1,

$$\Pr[\mathsf{ctr} \geq (1/2 + \varepsilon/4) \cdot 32n/\varepsilon^2] < \exp\left(-32 \cdot \left(\frac{\varepsilon}{4+\varepsilon}\right)^2 \cdot \frac{n(1/2+\varepsilon/8)}{3\varepsilon^2}\right) \leq \exp(-4n/15).$$

□ □

Therefore, by a straightforward union bound, with probability at least $1 - \exp(-n) - \exp(-n/8) - n\exp(-4n/15)/2 = 1 - \exp(-\Omega(n))$,

- $\mathcal{B}$ declares $n/(2\varepsilon^2)$ matrices to be good, and
- Every matrix A_i declared good by $\mathcal{B}$ belongs to $\mathsf{Good}^{\mathcal{A}}_{\varepsilon/4}$.

Then, for each A_i, observe that if $(A, \mathbf{b})$ is an LPN sample, $(A_i, \mathbf{b}_i^0 = \mathbf{b}_i \oplus A_i\mathbf{x}_i)$ is a uniformly random LPN sample, while if $\mathbf{b}$ is uniform, the $\mathbf{b}_i^0$ are uniform as well. That is,

- If $\mathbf{b}$ is uniform ($c = 1$), then $\mathbf{b}_i^0$ and $\mathbf{b}_i^1$ are identically distributed for every $i \in S$. Therefore, c_i is perfectly independent of σ_i, and $\sum_{i\in S} \sigma_i \oplus c_i$ is a sum of independent unbiased random coins.
- Else, if $\mathbf{b}$ is an LPN sample ($c = 0$), then distinguishing $(A_i, \mathbf{b}_i^0)$ from $(A_i, \mathbf{b}_i^1)$ is exactly distinguishing between $\mathcal{D}_0^{A_i}$ and $\mathcal{D}_1^{A_i}$. Since each $A_i \in \mathsf{Good}^{\mathcal{A}}_{\varepsilon/4}$ (with overwhelming probability), we have for every $i \in S$,

$$\Pr[\sigma_i \oplus c_i = 1 \mid c_i \xleftarrow{\$} \{0,1\}, \mathcal{A}(A_i, \mathbf{b}_i^{c_i})] \leq 1/2 - \varepsilon/8.$$

Therefore,

$$\begin{aligned}2 \cdot \Pr[\mathcal{B} \text{ fails}] &= \Pr\left[\sum_{i\in S} \sigma_i \oplus c_i > B \mid c = 0\right] + \Pr\left[\sum_{i\in S} \sigma_i \oplus c_i \leq B \mid c = 1\right] \\ &\leq \exp\left(-\frac{|S|\cdot(1/2-\varepsilon/8)}{3} \cdot \left(\frac{\varepsilon}{8-2\varepsilon}\right)^2\right) + \exp\left(-\frac{|S|\cdot 1/2}{2} \cdot \left(\frac{\varepsilon}{8}\right)^2\right) \\ &= \exp\left(-\frac{n}{48\cdot(8-2\varepsilon)}\right) + \exp\left(-\frac{n}{256}\right) \leq \exp(-\Omega(n)),\end{aligned}$$

where the first inequality follows by applying a Chernoff bound on both terms of the sum. This concludes the proof that $\mathcal{B}$ is successful. □ □

9.4.6. Variants: changing the noise or matrix distribution

We focused in the above on the noise distribution $\mathcal{X}$ since it is the most convenient to use in our constructions. However, the same analysis and conjectures apply equivalently to the other standard noise distributions $(\mathcal{B}, \mathcal{S}, \mathcal{R})$ (the analysis in the linear test framework gives identical bounds). Furthermore, since in our regime we typically have $n = k^{1+\eta}$ with $\eta < 1$, it is not too hard to provide tight reductions between $(\mathcal{S}^c, \mathcal{X}_{n,w})$-LPN and $(\mathcal{S}^c, \mathcal{S}_{n,w})$-LPN and/or $(\mathcal{S}^c, \mathcal{B}_{n,w})$-LPN.

More interestingly, one can also consider a different distributions over sparse matrices, in the hope of getting better bounds on the dual distance. And indeed, such a distribution was exhibited in the work of Applebaum and Kachlon [AK19]: Theorem 8.2 in [AK19] states that for every constant $\ell > 1$, $c > 4\ell$, there exists a *negligible-error* polynomial-time algorithm that samples matrices A over $\mathbb{F}_2^{n\times k}$, with $n = k^\ell$ rows of weight c with dual distance $\mathsf{dd}(A) \geq \Omega(k^\eta)$ for some suitable constant $\eta(\ell, c) \leq 1 - 4(\ell-1)/(c-4)$. On the flip side, this sampler achieves only a *slightly* negligible error probability.

9.4.7. Predicate-conditioned sparse-LPN

We now state a result that will prove useful in our analysis later. Let $\mathcal{P} = \{\mathsf{P} : \mathbb{F}_2^n \to \{0,1\}\}$ denote a family of predicates. For a noise distribution $\mathcal{E}$, let us denote $\mathcal{E}|_{\mathsf{P}}$ the distribution $\{\mathbf{e} \ : \ \mathbf{e} \xleftarrow{\$} \mathcal{E} \mid \mathsf{P}(\mathbf{e}) = 1\}$ (that is, $\mathcal{E}|_{\mathsf{P}}$ samples vectors from $\mathcal{E}$ conditioned on $\mathsf{P}(\mathbf{e}) = 1$). Fix a predicate P and consider the following assumption (for suitable parameters (T, ε, δ)):

Definition 9.8 ($\mathcal{P}$-conditioned sparse-LPN)**.** For any constants $c \geq 3$ and $\eta \in (0,1)$, for every $T = 2^{o(n)}$, we say that the $\mathcal{P}$-conditioned $(\mathcal{S}^c, \mathcal{X}_{n,w})$-LPN problem with dimension $k = k(n)$, number of samples $n = n(k) \leq k^{(1-\eta)c/2+\eta}$, and noise $w = w(n,k) \leq n\cdot k^{(1-\eta)c/2}$ is (T, ε, δ)-hard if for every probabilistic adversary $\mathcal{A} = (\mathcal{A}_n)_{n\in\mathbb{N}}$ of size at most $T = T(n)$, it holds that for all large enough k,

$$\Pr_{A\xleftarrow{\$}\mathcal{S}^c, \mathsf{P}\xleftarrow{\$}\mathcal{P}}\left[\mathsf{Adv}_{\mathcal{A}_k}(\mathcal{D}_0^{A,\mathsf{P}}, \mathcal{D}_1^{A,\mathsf{P}}) > \varepsilon\right] \leq \delta,$$

where $\mathcal{D}_0^{A,\mathsf{P}}$ denotes the distribution $\{(A, \mathsf{P}, A \cdot \mathbf{s} + \mathbf{e}) \mid \mathbf{x} \xleftarrow{\$} \mathbb{F}_2^k, \mathbf{e} \xleftarrow{\$} \mathcal{X}_{n,w}|_{\mathsf{P}}\}$ and $\mathcal{D}_1^{A,\mathsf{P}}$ denotes $\{(A, \mathsf{P}, \mathbf{b}) \mid \mathbf{b} \xleftarrow{\$} \mathbb{F}_2^n\}$.

9.4.7.1. Reducing $\mathcal{P}$-conditioned sparse-LPN to sparse-LPN.

In this section, we prove a reduction between $\mathcal{P}$-conditioned sparse-LPN and sparse-LPN. The quality of the reduction depends on the quantity $\mathsf{err}(\mathcal{P}) = \max_{\mathbf{e}} \Pr_{\mathsf{P} \xleftarrow{\$} \mathcal{P}}[\mathsf{P}(\mathbf{e}) \neq 1]$.

Lemma 9.16. *Assume that the $(\mathcal{S}^c, \mathcal{X}_{n,w})$-LPN problem with dimension $k = k(n)$, number of samples $n = k^{(1-\eta)c/2+\eta}$, and noise $w = n \cdot k^{(1-\eta)c/2}$ is $(T', \varepsilon, \delta)$-secure. Then the $\mathcal{P}$-conditioned $(\mathcal{S}^c, \mathcal{X}_{n,w})$-LPN problem with dimension $k = k(n)$, number of samples $n = k^{(1-\eta)c/2+\eta}$, and noise $w = n \cdot k^{(1-\eta)c/2}$ is $(O(T), \varepsilon + \mathsf{err}(\mathcal{P}), \delta)$-secure.*

Proof. Let $\mathcal{A}$ be a (T, ε, δ)-adversary against the $\mathcal{P}$-conditioned $(\mathcal{S}^c, \mathcal{X}_{n,w})$-LPN problem with dimension $k = k(n)$, number of samples $n = k^{(1-\eta)c/2+\eta}$, and noise $w = n \cdot k^{(1-\eta)c/2}$. We build an adversary $\mathcal{B}$ against the $(\mathcal{S}^c, \mathcal{X}_{n,w})$-LPN problem as follows: given as input a $(\mathcal{S}^c, \mathcal{X}_{n,w})$-LPN problem $(A, \mathbf{b})$, $\mathcal{B}$ samples $\mathsf{P} \xleftarrow{\$} \mathcal{P}$ and feeds $(A, \mathsf{P}, \mathbf{b})$ to $\mathcal{B}$. Observe that

- If $\mathbf{b}$ is uniformly random, then $(A, \mathsf{P}, \mathbf{b})$ is a valid random $\mathcal{P}$-conditioned instance.
- Else, if $\mathbf{b}$ is an LPN sample $\mathbf{b} = A \cdot \mathbf{x} + \mathbf{e}$, then the distribution of $(A, \mathsf{P}, \mathbf{b})$ *conditioned on* $\mathsf{P}(\mathbf{e}) = 1$ is a valid $\mathcal{P}$-conditioned LPN instance.

For $\sigma = 0, 1$, let us denote

$$p_\sigma(A, \mathsf{P}) = \Pr_{(A,\mathsf{P},\mathbf{b}) \xleftarrow{\$} \mathcal{D}_\sigma^{A,\mathsf{P}}}[\mathcal{A}(A, \mathsf{P}, \mathbf{b}) = 0].$$

We have $\mathsf{Adv}^{\mathcal{A}}(\mathcal{D}_0^{A,\mathsf{P}}, \mathcal{D}_1^{A,\mathsf{P}}) = |p_0(A, \mathsf{P}^t) - p_1(A, \mathsf{P}^t)|$. Then:

$$\begin{aligned}
&\Pr_A\left[\mathsf{Adv}^{\mathcal{B}}(\mathcal{D}_0^A, \mathcal{D}_1^A) > \varepsilon'\right] \\
&= \Pr_{A,\mathsf{P}}\left[\left|\Pr[\mathcal{A}(A, \mathsf{P}, \mathbf{b}) = \ \mid \mathbf{b} \text{ random}] - \Pr[\mathcal{B}(A, \mathsf{P}, \mathbf{b}) = \ \mid \mathbf{b} \text{ LPN}]\right| > \varepsilon'\right] \\
&\leq \max_{x \in [0,1]} \Pr_{A,\mathsf{P}}\left[|p_1(A, \mathsf{P}) - p_0(A, \mathsf{P}) \cdot (1 - \mathsf{err}(\mathcal{P})) - x \cdot \mathsf{err}(\mathcal{P})| > \varepsilon'\right] \text{ by the Bayes rule} \\
&\leq \Pr_{A,\mathsf{P}}[|p_1(A, \mathsf{P}) - p_0(A, \mathsf{P})| - \mathsf{err}(\mathcal{P}) > \varepsilon'] \text{ by triangle inequality} \\
&= \Pr_{A,\mathsf{P}}\left[\mathsf{Adv}^{\mathcal{A}}(\mathcal{D}_0^{A,\mathsf{P}}, \mathcal{D}_1^{A,\mathsf{P}}) > \varepsilon' + \mathsf{err}(\mathcal{P})\right] \leq \delta,
\end{aligned}$$

and we conclude the proof by setting $\varepsilon' = \varepsilon - \mathsf{err}(\mathcal{P})$. □

9.5. A Structured-Seed Local PRG from Sparse LPN

Our constructions rely on a simple method to compress a length-ℓ unit vector $\mathbf{u}$ into d smaller unit vectors $(\mathbf{u}_1, \cdots, \mathbf{u}_d)$ of length $\ell^{1/d}$ such that each entry of $\mathbf{u}$ can be reconstructed by retrieving a single entry from each $\mathbf{u}_i$.

9.5.1. Compressing unit vectors

Let $\mathsf{unit}_m(i)$ denote the procedure which, on input $i \in [m]$, outputs a length-m one-hot $\mathbb{F}_2$-vector with a 1 at position i. Conversely, let $\mathsf{nzi}(\mathbf{u})$ denote the procedure which, given as input a length-m one-hot $\mathbb{F}_2$-vector $\mathbf{u}$, returns its non-zero index i. We describe the compression and reconstruction algorithms below:

Algorithms 1 $(\mathsf{Comp}, \mathsf{Rec})$:

$\mathsf{Comp}(\mathbf{u}, d)$: on input a length-$\ell$ unit vector $\mathbf{u}$ and a compression factor d,

1. Set $\ell_d \leftarrow \lceil \ell^{1/d} \rceil$.
2. Compute $i \leftarrow \mathsf{nzi}(\mathbf{u})$ and write i over the ℓ_d-ary basis as $(i_1, \cdots, i_d) \in [\ell_d]^d$.
3. Output $(\mathbf{u}_1, \cdots, \mathbf{u}_d) \leftarrow (\mathsf{unit}_{\ell_d}(i_1), \cdots, \mathsf{unit}_{\ell_d}(i_d))$.

$\mathsf{Rec}(j, (\mathbf{u}_1, \cdots, \mathbf{u}_d))$: on input an index $j \in [\ell]$ and d one-hot $\mathbb{F}_2$ vectors $\mathbf{u}_i \in \mathbb{F}_2^{\ell_d}$, with $\ell_d^d \geq \ell$,

1. Write j over the ℓ_d-ary basis as $(j_1, \cdots, j_d) \in [\ell_d]^d$.
2. Return $b \leftarrow \prod_{i=1}^{d} u_i[j_i]$. // AND operation over $\mathbb{F}_2$.

We note in passing that since $\mathsf{Comp}(\mathbf{u}, d)$ starts by computing $i = \mathsf{nzi}(\mathbf{u})$, it never needs to store or read $\mathbf{u}$ in "expanded form" and can be passed i directly. We will make use of this observation when estimating the running time of our algorithms. From the description above, it is clear that $\mathsf{Rec}(j, (\mathbf{u}_1, \cdots, \mathbf{u}_d))$ always reads exactly d bits from its second input $(\mathbf{u}_1, \cdots, \mathbf{u}_d)$: for every j, $\mathsf{Rec}(j, \cdot)$ is d-local. Furthermore, it is easy to observe that on any input $(j, \mathsf{Comp}(\mathbf{u}, d))$, Rec correctly reconstructs the j-th bit of $\mathbf{u}$:

Claim 9.17. *For every d and length-ℓ one-hot $\mathbb{F}_2$-vector $\mathbf{u}$, for every $j \leq \ell$, it holds that*

$$\mathsf{Rec}(j, \mathsf{Comp}(\mathbf{u}, d)) = \mathbf{u}[j].$$

We say that $(\mathsf{Comp}, \mathsf{Rec})$ is *correct* to denote this property.

9.5.2. Warm-up: a structured-seed local PRG from regular sparse LPN

Let $(w, k) = (w(n), k(n))$ denote respectively the noise weight and dimension of an LPN instance, and let $n = n(k)$ be the (polynomial) number of samples, chosen such that w divides n. Let $c \geq 3$ and d be two constants. Let $\ell \leftarrow n/w$ and $\ell_d \leftarrow \lceil (n/w)^{1/d} \rceil$. We describe below a structured-seed local PRG whose security reduces to the hardness of the regular sparse-LPN assumption:

- Global parameters: two constants (c, d), the noise weight $w = w(n)$ of, the dimension $k = k(n)$, and the stretch $n = n(k)$. All global parameters are implicitly passed as inputs to all algorithms.

- $\mathsf{Setup}(1^n)$: sample $A \xleftarrow{\$} \mathcal{M}^c_{k,n}$. Let $(\mathbf{a}_1, \cdots, \mathbf{a}_n)$ denote the rows of A (of Hamming weight c). Output $\mathsf{pp} \leftarrow A$.
- $\mathsf{SampleSeed}(\mathsf{pp})$: sample $\mathbf{x} \xleftarrow{\$} \mathbb{F}_2^k$ and w unit vectors $(\mathbf{e}_1, \cdots, \mathbf{e}_w) \xleftarrow{\$} \mathcal{U}_{n/w} \times \cdots \times \mathcal{U}_{n/w}$. Output $\mathsf{seed} \leftarrow (\mathbf{x}, \mathsf{Comp}(\mathbf{e}_1, d), \cdots, \mathsf{Comp}(\mathbf{e}_w, d))$.
- $\mathsf{PRG}_{\mathsf{pp}}(\mathsf{seed})$: parse seed as $(\mathbf{x}, (\mathbf{u}_{1,1}, \cdots, \mathbf{u}_{1,d}), \cdots, (\mathbf{u}_{w,1}, \cdots, \mathbf{u}_{w,d}))$, where each $\mathbf{u}_{i,j}$ is a unit vector over $\mathbb{F}_2^{\ell_d}$ for every $i \le w$, $j \le d$. For $i = 1$ to n, write i as $(\alpha - 1)\cdot(n/w) + \beta$, with $\alpha \in [w]$ and $\beta \in [n/w]$. set $y_i \leftarrow \langle \mathbf{a}_i, \mathbf{x} \rangle + \mathsf{Rec}(\beta, (\mathbf{u}_{\alpha,1}, \cdots, \mathbf{u}_{\alpha,d}))$. Output $(y_1, \cdots, y_n)$.

Theorem 9.18. *Assuming the (T, ε, δ)-hardness of the $(\mathcal{M}^c, \mathcal{R})$-LPN problem, for any constant $d \ge 3$, $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ is a $(T - O(n), \varepsilon, \delta)$-secure structured-seed local pseudorandom generator with seed length $k + wd \cdot \lceil (n/w)^{1/d} \rceil$, stretch n, and locality $c + d$.*

For example, setting $k = w$ and $n(k) = k^{1.99}$ yields a PRG with seed length $s = O(k^{1+0.99/d})$ and stretch $k^{1.99} = \Omega(s^{1.99/(1+0.99/d)})$; for $d = 10$, this translates to a stretch $\Omega(s^{1.81})$. In general, as n approaches k^2 and d grows, the stretch becomes $\Omega(s^{2-\varepsilon_d})$ for an arbitrarily small constant ε_d.

Proof. Let $\mathbf{e}$ denote the regular vector obtained by concatenating $(\mathbf{e}_1, \cdots, \mathbf{e}_w)$. Note that $\mathbf{e}$ is distributed as a random sample from $\mathcal{R}_{n,w}$. By correctness of $(\mathsf{Comp}, \mathsf{Rec})$, $\mathsf{Rec}(\beta, (\mathbf{u}_{\alpha,1}, \cdots, \mathbf{u}_{\alpha,d})) = \mathbf{e}_\alpha[\beta] = \mathbf{e}[i]$ (the β-th entry of the α-th block of $\mathbf{e}$ is exactly the i-th entry of $\mathbf{e}$ as $i = (\alpha - 1) \cdot (n/w) + \beta$). Therefore, denoting $\mathbf{y} = (y_1, \cdots, y_n)$,

$$\mathbf{y} = (\langle \mathbf{a}_i, \mathbf{x} \rangle + \mathbf{e}[i])_{i \le n} = A \cdot \mathbf{x} + \mathbf{e}.$$

From there, it follows immediately that breaking the $(T - O(n), \varepsilon, \delta)$-security of $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ translates to breaking the (T, ε, δ)-hardness of the Alekhnovich assumption (the reduction is straightforward; the $O(n)$ term accounts for the cost of sampling the LPN instance and compressing the unit vectors). Eventually, $(c + d)$-locality follows from the fact that each $\mathbf{a}_i$ is c-sparse, hence the mapping $\mathbf{x} \mapsto \langle \mathbf{a}_i, \mathbf{x} \rangle$ is c-local, and the mapping $\mathsf{Rec}(\beta, \cdot)$ is d-local for any $\beta \in [n/w]$. The theorem follows. □

9.5.3. Removing regularity using 2-choice hashing

The construction presented in the previous section requires the sparse-LPN assumption to be secure when the noise has a regular structure. In this section, we explain how to lift this restriction using efficient hashing schemes for allocating elements into bins. We use 2-choice hashing [CRS03; SEK03] for the sake of concreteness, but we note that our construction can be framed generically using the language of batch codes [IKOS04]. Replacing regular noise with random sparse noise in the LPN variant using hashing or batch codes has been done in previous works [BCGI18; SGRR19; BBCCDS24b]. Here, we show how to integrate this approach into our structured-seed local PRG without sacrificing constant locality.

Hash functions are commonly used to distribute items into bins. In its simplest form, N items $(u_1, \cdots, u_N)$ from a universe $\mathbb{U}$ can be placed into $L \cdot N$ bins $(1, \cdots, N)$ (for a suitable constant L) using a hash function $h : \mathbb{U} \to [L \cdot N]$, by placing each item u_i at position $h(i)$; when h is a random function, it is well known that with high probability, the maximum load across all bins will be $O(\log N / \log \log N)$ [RS98]. The question of finding alternative hashing strategies that result in a more balanced load has been an active and fruitful field of research [PSWW18; PRTY19; SGRP19]. Typically, these improved strategies rely on multiple hash functions $(h_1, h_2, \cdots)$, combined with an *allocation scheme* to determine, for each item u, which hash function should be used to allocate u to a bin.

For our application, we will need a hashing scheme that guarantees, with probability $1 - O(1)$ (over the random choice of the hash functions, for an arbitrary set of items to be placed) that each bin will contain a constant number of items. There are multiple options with different degrees of simplicity and parameter tradeoffs. For the sake of concreteness, we focus on one of the simplest possible solutions, called *2-choice hashing* [SEK03].

Definition 9.9 (Allocation)**.** Let $\mathbb{U}$ be a set, L be a constant, and N be an integer. Let h_0, h_1 be two hash functions from $\mathbb{U}$ to $[L \cdot N]$. For any N-tuple $\mathbf{u} = (u_1, \cdots, u_N) \in \mathbb{U}^N$, we define an *allocation* of $\mathbf{u}$ into the bins $1 \cdots N$ with respect to (h_0, h_1) to be a vector $\mathbf{b} \in \mathbb{F}_2^{L \cdot N}$ indicating which bin each item is mapped to: for any $i \in [N]$, the item u_i is mapped to the bin $h_{\mathbf{b}[i]}(u_i)$. Given (h_0, h_1), a tuple $\mathbf{u}$, and an allocation $\mathbf{b}$, we let $\mathsf{Load}_i(h_0, h_1, \mathbf{u}, \mathbf{b})$ denote the load of the bin i (*i.e.* the total number of indices j such that $h_{\mathbf{b}[j]}(u_j) = i$).

We will rely on the following lemma:

Lemma 9.19 (2-choice hashing [SEK03; PRTY19])**.** *Let $\mathbb{U}$ be a set, L be a constant, and N be an integer. Then, there exists a deterministic algorithm* Alloc *running in time $O(N \log N)$ which, on input two hash functions h_0, h_1 from $\mathbb{U}$ to $[L \cdot N]$ and an N-tuple $\mathbf{u} \in \mathbb{U}^N$, returns an allocation $\mathbf{b}$, and such that for every $\mathbf{u} \in \mathbb{U}^N$,*

$$\Pr_{h_0, h_1}\left[\mathbf{b} \leftarrow \mathsf{Alloc}(h_0, h_1, \mathbf{u}) \ : \ \max_{i \le L \cdot N} \mathsf{Load}_i(h_0, h_1, \mathbf{u}, \mathbf{b}) > L + 1\right] \le O\left(\frac{1}{N^L}\right).$$

For notational convenience, we will define the *quality* of a pair of hash function (h_0, h_1) as the fraction of vectors $\mathbf{u} \in \mathbb{U}^N$ that have a good allocation (*i.e.*, such that $\max_{i \le L \cdot N} \mathsf{Load}_i(h_0, h_1, \mathbf{u}, \mathbf{b}) \le L + 1$):

Definition 9.10 (Quality)**.** We call *quality* of a pair (h_0, h_1) of functions from $\mathbb{U}$ to $[L \cdot N]$, and denote $\mathsf{Quality}(h_0, h_1)$, the quantity

$$\mathsf{Quality}(h_0, h_1) := \Pr_{\mathbf{u} \xleftarrow{\$} \mathbb{U}^N}\left[\mathbf{b} \leftarrow \mathsf{Alloc}(h_0, h_1, \mathbf{u}) \ : \ \max_{i \le L \cdot N} \mathsf{Load}_i(h_0, h_1, \mathbf{u}, \mathbf{b}) > L + 1\right].$$

Then, Lemma 9.19 says that on average, a random choice of (h_0, h_1) has quality $1 - O(1/N^L)$.

9.5.4. Sampling the seed

As for the regular construction, we assume that all algorithms implicitly receive as input global parameters $\mathsf{gp} = (w, k, n, c, d, L)$ where $w = w(n)$ is the noise weight parameter, $k = k(n)$ is the LPN dimension, $n = n(k)$ denotes the stretch (or number of samples), and $c, d \geq 3$ and $L \geq 1$ denote three constants.

At a high level, our construction proceeds by using 2-choice hashing to allocate the nonzero entries of the noise vector into unit vectors, such that every entry of the noise vector can be reconstructed by looking at one position in $2(L+1)$ unit vectors, and compressing these unit vectors with the compression algorithm Comp. In more details, assume that the setup $\mathsf{Setup}(1^n)$ produces a matrix $A \xleftarrow{\$} \mathcal{M}^c_{k,n}$ together with two hash functions (h_0, h_1) (we ignore for now the issue of how "good" hash functions (h_0, h_1) are selected). Then, $\mathsf{SampleSeed}(\mathsf{pp})$ proceeds as follows:

Algorithm 2 $\mathsf{SampleSeed}(A, h_0, h_1)$:

---

1. Sample $\mathbf{x} \xleftarrow{\$} \mathbb{F}_2^k$.
2. Sample (the positions of) a noise vector $\mathbf{u} \xleftarrow{\$} [n]^w$.
3. Set $\mathbf{b} \leftarrow \mathsf{Alloc}(h_0, h_1, \mathbf{u})$. If $\max_{i \leq Lw} \mathsf{Load}_i(h_0, h_1, \mathbf{u}, \mathbf{b}) > L + 1$, **go to** Item 2.
4. For $i = 1$ to Lw, define $\mathbf{v}_i \in \mathbb{F}_2^n$ as follows:

$$\mathbf{v}_i \leftarrow \bigoplus_{j : h_{\mathbf{b}[j]}(\mathbf{u}[j]) = i} \mathsf{unit}_n(\mathbf{u}[j]).$$

   That is, for every entry j of $\mathbf{u}$ mapped to the bin i via the allocation (computed as $h_{\mathbf{b}[j]}(\mathbf{u}[j])$), sum 1 to the position $\mathbf{u}[j]$ in $\mathbf{v}_i$. Note that by construction, $\mathsf{HW}(\mathbf{v}_i) \leq L + 1$.
5. Write $\mathbf{v}_i$ as a sum of $L+1$ (unit or zero) vectors $\mathbf{v}_i = \mathbf{v}_i^0 + \cdots + \mathbf{v}_i^L$ with $\mathsf{HW}(\mathbf{v}_i^\ell) \leq 1$ for $\ell \in \{0, \cdots, L\}$.
6. Output $\mathsf{seed} \leftarrow (\mathbf{x}, (\mathsf{Comp}(\mathbf{v}_i^0, d), \cdots, \mathsf{Comp}(\mathbf{v}_i^L, d))_{i \leq Lw})$.

Note that $\mathsf{SampleSeed}$ does not have to work with an expanded representation of the vectors $\mathbf{v}_i$: the $\mathbf{v}_i$ can be manipulated in compact form (as the list of their nonzero entries, of size at most 2) throughout the entire execution (including the execution of Comp, since for any $\mathbf{v}$, $\mathsf{Comp}(\mathbf{v}, d)$ only needs the nonzero entry $i = \mathsf{nzi}(\mathbf{v})$, as noted in Section 9.5.1).

9.5.5. Expanding the seed

Let us denote $\mathbf{e} = \bigoplus_{j'=1}^{w} \mathsf{unit}_n(\mathbf{u}[j'])$ the noise vector in expanded form. Observe that in the $\mathsf{SampleSeed}$ process above, every vector $\mathsf{unit}_n(\mathbf{u}[j'])$ is XORed to exactly one $\mathbf{v}_i$: the one with index $i = h_{\mathbf{b}[j']}(\mathbf{u}[j'])$. Therefore, for every position $j \in [n]$ of $\mathbf{e}$, two cases can occur:

1. Either there exists j' such that $j = \mathbf{u}[j']$ (*i.e.*, j is a noise position of $\mathbf{e}$). Then the j-th entry of $\mathbf{v}_i$ with $i = h_{\mathbf{b}[j']}(\mathbf{u}[j'])$ is set to 1, and the j-th entry of the alternative option $\mathbf{v}_{i'}$, where $i' = h_{\bar{\mathbf{b}}[j']}(\mathbf{u}[j'])$ (i' is the index of the "other bin", not selected by the allocation) stays at 0.

2. Or there is no such j' (*i.e.*, j is not a noise position), in which case both $\mathbf{v}_{i_0}[j]$ and $\mathbf{v}_{i_1}[j]$ are equal to 0, where $(i_0, i_1) = (h_0(j), h_1(j))$.

In both cases, it holds that $\mathbf{e}[j] = \mathbf{v}_{i_0}[j] \oplus \mathbf{v}_{i_1}[j]$, with $(i_0, i_1) = (h_0(j), h_1(j))$. Hence, to "read" $\mathbf{e}[j]$, it suffices to read the j-th entry in $\mathbf{v}_{i_0}$ and $\mathbf{v}_{i_1}$ for $(i_0, i_1) = (h_0(j), h_1(j))$. This can be done by reading d entries in each of $\mathsf{Comp}(\mathbf{v}_{i_0}^0, d), \cdots, \mathsf{Comp}(\mathbf{v}_{i_0}^L, d), \mathsf{Comp}(\mathbf{v}_{i_1}^0, d), \cdots, \mathsf{Comp}(\mathbf{v}_{i_1}^L, d)$ (hence $2d(L+1)$ entries in total) by the d-locality of $(\mathsf{Comp}, \mathsf{Rec})$. Concretely, given parameters $\mathsf{pp} = (A, h_0, h_1)$ and a seed $\mathsf{seed} = (\mathbf{x}, (\mathsf{Comp}(\mathbf{v}_i^0, d), \cdots, \mathsf{Comp}(\mathbf{v}_i^L, d))_{i \le Lw})$, the j-th entry of the noise vector $\mathbf{e} \in \mathbb{F}_2^n$ (whose nonzero entries are given by $\mathbf{u}$) is reconstructed as follows: compute $(i_0, i_1) \leftarrow (h_0(j), h_1(j))$. Set

$$\mathbf{e}[j] \leftarrow \bigoplus_{\alpha \le L, \beta \in \{0,1\}} \mathsf{Rec}(j, \mathsf{Comp}(\mathbf{v}_{i_\beta}^\alpha, d)) = \bigoplus_{\alpha \le L, \beta \in \{0,1\}} \mathbf{v}_{i_\beta}^\alpha[j] = \mathbf{v}_{i_0}[j] \oplus \mathbf{v}_{i_1}[j].$$

The detailed procedure is represented below.

Algorithm 3 $\mathsf{PRG}_{\mathsf{pp}}(\mathbf{x}, (\mathsf{Comp}(\mathbf{v}_i^0, d), \mathsf{Comp}(\mathbf{v}_i^1, d))_{i \le w})$:

---

1. Parse pp as (A, h_0, h_1).

2. For $j = 1$ to n, set $(i_0^j, i_1^j) \leftarrow (h_0(j), h_1(j))$.

3. For $j = 1$ to n, set

$$y_j \leftarrow \langle \mathbf{a}_j, \mathbf{x} \rangle + \sum_{\substack{0 \le \alpha \le L \\ \beta \in \{0,1\}}} \mathsf{Rec}\left(j, \mathsf{Comp}\left(\mathbf{v}_{i_\beta^j}^\alpha, d\right)\right).$$

4. Output $(y_1, \cdots, y_n)$.

To complete the construction, it remains to explain how (h_0, h_1) are sampled by Setup. Concretely, the algorithm $\mathsf{SampleSeed}$ requires (h_0, h_1) to be "good" in the following sense: for a random choice of noise positions $\mathbf{u} \xleftarrow{\$} [n]^w$ in step 2, it should hold with sufficiently large probability p that $\max_{i \le Lw} \mathsf{Load}_i(h_0, h_1, \mathbf{u}, \mathbf{b}) \le L + 1$ using

the allocation $\mathbf{b} \leftarrow \mathsf{Alloc}(h_0, h_1, \mathbf{u})$, since the expected runtime of $\mathsf{SampleSeed}$ grows as $1/p$. In other words, we want $\mathsf{Quality}(h_0, h_1)$ to be sufficiently large (with overwhelming probability over the choice of (h_0, h_1)).

Lemma 9.19 guarantees that for a *fixed choice of* $\mathbf{u}$, a random choice of (h_0, h_1) has probability close to 1 of yielding a good allocation. Looking ahead, our Setup procedure builds upon Lemma 9.19 to produce with overwhelming probability a pair (h_0, h_1) that yield a good allocation for *a constant fraction* of all $\mathbf{u}$'s (*i.e.*, $\mathsf{Quality}(h_0, h_1)$ is bounded below by a constant). This bounds p by a constant, causing only a constant blowup to the expected runtime of $\mathsf{SampleSeed}$[1].

9.5.6. Testing the hash functions

We let $\mathcal{F}_{n,Lw}$ denote the set of all functions from $[n]$ to $[Lw]$. Our setup procedure builds upon a Test subroutine to test whether pairs of hash functions (h_0, h_1) are sufficiently good. The procedure is represented below:

Algorithm 4 $\mathsf{Test}(1^n, h_0, h_1)$:

1. Set $\mathsf{good} \leftarrow 0$ and $T \leftarrow 42 \cdot n$. For $j = 1$ to T,
   a) Sample $\mathbf{u}_j \xleftarrow{\$} [n]^w$.
   b) Set $\mathbf{b}_j \leftarrow \mathsf{Alloc}(h_0, h_1, \mathbf{u}_j)$.
   c) If $\max_{i \le Lw} \mathsf{Load}_i(h_0, h_1, \mathbf{u}_j, \mathbf{b}_j) \le L + 1$, set $\mathsf{good} \leftarrow \mathsf{good} + 1$.
2. If $\mathsf{good} \ge 0.3 \cdot T$, output 1. Else, output 0.

In other words, the procedure Test computes an empirical estimate of the quality of (h_0, h_1) on a random test set of $T = 42 \cdot n$ vectors $\mathbf{u}$, and outputs 1 iff the empirical quality is at least 30% (see Lemma 9.20 for a detailed explanation of the choice of number of vectors in the random test set and the empirical quality). Our Setup algorithm will rely on two properties of Test:

1. (few false positive) the probability (over the randomness of Test) that $\mathsf{Test}(1^n, h_0, h_1) = 1$ but $\mathsf{Quality}(h_0, h_1) < 0.2$ is negligible;
2. (high success rate) the probability (over the random choice of (h_0, h_1) and the randomness of Test) that $\mathsf{Test}(h_0, h_1) = 1$ is bounded below by a constant.

Both proofs are elementary applications of standard tail bounds; we prove them in Section 9.5.8.

[1] We note that one can make the runtime strictly polytime by adding a bound λ on the number of retries, where lambda is some security parameter, and aborting if the bound is reached. This gives a strict polytime algorithm with a n blowup in runtime and a negligible failure probability $(1-p)^n$.

9.5.7. Sampling the hash functions

The setup procedure is represented below.

Algorithm 5 $\mathsf{Setup}(1^n, k, n)$:

1. Sample $A \xleftarrow{\$} \mathcal{M}^c_{k,n}$. Let $(\mathbf{a}_1, \cdots, \mathbf{a}_n)$ denote the rows of A (of Hamming weight c).
2. Sample $(h_0, h_1) \xleftarrow{\$} \mathcal{F}_{n,Lw} \times \mathcal{F}_{n,Lw}$.
3. If $\mathsf{Test}(1^n, h_0, h_1) = 0$, **go to** Item 2.
4. Output $\mathsf{pp} \leftarrow (A, h_0, h_1)$.

In Section 9.5.8, we will prove two lemmas: Lemma 9.21 shows that Test succeeds with probability at least 1/2 ("Test succeeds often enough") and Lemma 9.20 shows that (h_0, h_1) have good quality $\mathsf{Quality}(h_0, h_1) \geq 0.2$ with overwhelming probability ("Test has few false positives"). We discuss consequences for the running time of Setup and SampleSeed below.

Since each Test succeeds with probability at least 1/2 by Lemma 9.21, $\mathsf{Setup}(1^n)$ executed a constant expected number of Test (alternatively, we can let Setup run up to n tests and abort if none succeeded to get a strict bound on the running time and negligible abortion probability). Furthermore, by Lemma 9.20, the probability that the functions (h_0, h_1) output by $\mathsf{Setup}(1^n)$ have quality $\mathsf{Quality}(h_0, h_1) < 0.2$ is at most $1/2^n$. In turn, this implies that the algorithm SampleSeed will succeed in sampling $\mathbf{u}$ in step 3 after at most 5 tries in expectation.

9.5.8. Properties of Test

9.5.8.1. Test has few false positives.

We prove the following:

Lemma 9.20. *Let (h_0, h_1) be two functions from $\mathcal{F}_{n,Lw}$ such that $\mathsf{Quality}(h_0, h_1) < 0.2$. Then,*

$$\Pr\left[\mathsf{Test}(1^n, h_0, h_1) = 1\right] \leq \frac{1}{2^n}.$$

Proof. Let (h_0, h_1) be two functions from $\mathcal{F}_{n,Lw}$ such that $\mathsf{Quality}(h_0, h_1) < 0.2$. Let X_j denote the random variable, for $j = 1$ to T, taking value 1 if $\max_{i \leq Lw} \mathsf{Load}_i(h_0, h_1, \mathbf{u}_j, \mathbf{b}_j) \leq L + 1$ and 0 else. Note that the random variables $X_1, \cdots, X_T$ are independent. Let $X \leftarrow \sum_{j=1}^{T} X_j$ and $\mu \leftarrow \mathbb{E}[X]$. Note that $\mu = T \cdot \mathbb{E}[X_1] = T \cdot \mathsf{Quality}(h_0, h_1) < 0.2T$. Let

us denote $x \leftarrow \mu/T < 0.2$. Then, by the Chernoff bound (Lemma 9.1),

$$\begin{aligned}\Pr[X \geq 0.3T] &\leq \exp\left(-\left(\frac{0.3T}{\mu} - 1\right)^2 \cdot \frac{\mu}{3}\right) \\ &= \exp\left(-\frac{0.03T^2}{\mu} + 0.2T - \frac{\mu}{3}\right) \\ &= \exp\left(-\left(\frac{0.03}{x} - 0.2 + \frac{x}{3}\right) \cdot T\right).\end{aligned}$$

Now, writing $f(x) = 0.03/x - 0.2 + x/3$, we have $f'(x) = (1 - (0.3/x)^2)/3 < 0$ since $x < 0.3$. Hence, $f(x)$ is decreasing and bounded above by $f(0.2) = 1/60$, which yields

$$\Pr[X \geq 0.3T] \leq \exp\left(-T/60\right) = 2^{-42 \cdot (\log_2 e/60) \cdot n} \leq 2^{-n},$$

which concludes the proof. □ □

9.5.8.2. Test succeeds often enough.

We prove the following:

Lemma 9.21. *There exists an integer N such that for all $w, n \geq N$*

$$\Pr_{h_0, h_1 \xleftarrow{\$} \mathcal{F}_{n,Lw}} \left[\mathsf{Test}(1^n, h_0, h_1) = 1\right] \geq \frac{1}{2}.$$

Proof. The proof follows immediately from the following (stronger) claim:

Claim 9.22. *There exists an integer N such that for all $w \geq N$,*

$$\Pr_{h_0, h_1 \xleftarrow{\$} \mathcal{F}_{n,Lw}} \left[\mathsf{Quality}(h_0, h_1) \geq 0.4\right] \geq \frac{1}{1.9}.$$

Furthermore, if (h_0, h_1) are two functions from $\mathcal{F}_{n,Lw}$ such that $\mathsf{Quality}(h_0, h_1) \geq 0.4$,

$$\Pr\left[\mathsf{Test}(1^n, h_0, h_1) = 0\right] \leq \frac{1}{2^{9n}}.$$

We prove each part of the stronger claim in turn. The first part of claim is an immediate application of the Markov inequality: by Lemma 9.19 and Markov inequality (Lemma 9.2),

$$\Pr_{h_0, h_1 \xleftarrow{\$} \mathcal{F}_{n,Lw}} \left[\mathsf{Quality}(h_0, h_1) \geq \frac{1 - O(1/w^L)}{1.9}\right] \geq \frac{1}{1.9},$$

and for a large enough w, $(1 - O(1/w^L))/1.9 \geq 0.9/1.9 > 0.4$. For the second claim, the proof is similar to that of (1), with a Chernoff bound in the other direction: let (h_0, h_1) be

two functions from $\mathcal{F}_{n,Lw}$ such that $\mathsf{Quality}(h_0, h_1) < 0.2$. Then by the Chernoff bound,

$$\begin{aligned}\Pr[X < 0.3T] &\leq \exp\left(-\left(1 - \frac{0.3T}{\mu}\right)^2 \cdot \frac{\mu}{2}\right) \\ &= \exp\left(-\frac{0.03T^2}{\mu} + 0.2T - \frac{\mu}{3}\right) \\ &= \exp\left(-\left(\frac{0.03}{x} - 0.2 + \frac{x}{3}\right) \cdot T\right).\end{aligned}$$

Since $\mathsf{Quality}(h_0, h_1) = \mu/T = x \in [0.4, 1)$, writing $f(x) = 0.03/x - 0.2 + x/3$, we have this time $f'(x) = (1 - (0.3/x)^2)/3 > 0$: $f(x)$ is increasing and bounded above by $f(1) = 49/300$. Then,

$$\Pr[X \geq 0.3T] \leq \exp\left(-49T/300\right) = 2^{-42 \cdot (49 \log_2 e/300) \cdot n} \leq 2^{-9n},$$

which concludes the proof. □ □

9.5.9. Efficiency and Security

Let $\mathcal{H}_L = \{\mathsf{P}_{h_0,h_1}\}$ denote the following family of predicates: given two hash functions $h_0, h_1 \in \mathcal{F}_{n,Lw}$ and an input $\mathbf{u} \in [n]^w$, $\mathsf{P}_{h_0,h_1}(\mathbf{u})$ samples $\mathbf{b} \leftarrow \mathsf{Alloc}(h_0, h_1, \mathbf{u})$. It returns 0 if $\max_{i \leq Lw} \mathsf{Load}_i(h_0, h_1, \mathbf{u}, \mathbf{b}) > L + 1$, and 1 otherwise. We now summarize the efficiency properties of our construction $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ described in the previous subsections.

Theorem 9.23. *Let $L \geq 1$ be a constant. Assume the (T, ε, δ)-hardness of the $\mathcal{H}_L$-conditioned $(\mathcal{M}^c_{n,k}, \mathcal{X}_{n,w})$-$\mathsf{LPN}$ problem, for some constant $c \geq 3$. Let $d \geq 2$ be a constant. Then there exists a $(T - O(n), \varepsilon, \delta - 1/2^n)$-secure structured-seed local pseudorandom generator $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ with the following characteristics:*

- Parameter size: $|\mathsf{pp}| = n \cdot (c \log k + 2 \log w)$;
- Seed length: $|\mathsf{seed}| = k + w \cdot L(L+1)d \cdot \lceil n^{1/d} \rceil$;
- Stretch: n;
- Locality: $c + 2(L+1)d$.

In addition, the algorithms have the following running time:

- $\mathsf{Setup}(1^n)$ *runs in time* $n \cdot \tilde{O}(n + nw)$,
- $\mathsf{SampleSeed}(\mathsf{pp})$ *runs in time* $O(k + w \cdot (n(\log w + \log n) + n^{1/d}))$.

Before we move on with the security analysis, we briefly overview each of the efficiency properties. We analyze runtime in a RAM model for simplicity, but note that all our algorithms can easily be implemented with similar runtime in a circuit model. The

size of pp and seed, and the stretch, can be read immediately from their description and that of Comp. As for locality, computing $\langle \mathbf{a}_j, \mathbf{x} \rangle$ requires reading c bits of $\mathbf{x}$ (since $\mathsf{HW}(\mathbf{a}_j) = c$), and computing $\mathsf{Rec}(j, \mathsf{Comp}(\mathbf{v}^{\alpha}_{i^{\alpha}_{j\beta}}, d))$ for $\alpha \leq L, \beta \in \{0,1\}$ requires reading d bits of $\mathsf{Comp}(\mathbf{v}^{\alpha}_{i_j}, d)$ each time (hence $2(L+1)d$ bits in total).

The running time of Setup is decomposed as follows: sampling A requires sampling $c \cdot n$ elements of $[k]$ in time $O(n \log k)$. Sampling (h_0, h_1) requires sampling $2n$ elements of $[w]$, in time $O(n \log w)$, and running $\mathsf{Test}(1^n, h_0, h_1)$ requires $O(n)$ samples over $[n]^w$ (each in time $w \cdot \log n$), computations of allocation and of the maximum load (in time $O(w \cdot (\log w + \log n))$. Furthermore, with overwhelming probability, Setup terminates after at most n executions of Test, hence the bound of $n \cdot \tilde{O}(n + nw)$ on the total runtime.

Eventually, the running time of $\mathsf{SampleSeed}$ is decomposed as follows: sampling $\mathbf{x}$ requires tossing k coins. Sampling $\mathbf{u} \xleftarrow{\$} [n]^w$, computing the allocation, and computing the maximum load runs in time $O(w \cdot (\log n + \log w))$. Then, computing the $(\mathbf{v}^0_i, \cdots, \mathbf{v}^L_i)$ (in implicit representation, as $\mathsf{nzi}(\mathbf{v}^0_i), \cdots, \mathsf{nzi}(\mathbf{v}^L_i)$) takes time $O(w \cdot (\log n + \log w))$, and compressing them takes time $O(w \cdot n^{1/d})$. Eventually, with overwhelming probability, $\mathsf{SampleSeed}$ produces $\mathbf{u}$ after at most n executions of the steps 2 and 3, hence the bound of $O(k + w \cdot (n(\log w + \log n) + n^{1/d}))$ on the total runtime.

Security analysis. We prove security of Theorem 9.5.9 under the (T, ε, δ)-hardness of the $\mathcal{H}_L$-conditioned $(\mathcal{M}^c_{n,k}, \mathcal{X}_{n,w})$-$\mathsf{LPN}$ problem. Let $\mathbf{e}$ denote the noise vector, which is sampled randomly from $\mathcal{X}_{n,w}$. We showed in Section 9.5.5 that $\bigoplus_{\alpha \leq L, \beta \in \{0,1\}} \mathsf{Rec}(j, \mathsf{Comp}(\mathbf{v}^{\alpha}_{i^{\alpha}_{j\beta}}, d)) = \mathbf{e}[j]$ for every $j \leq n$. Therefore, denoting $\mathbf{y} = (y_1, \cdots, y_n)$,

$$\mathbf{y} = (\langle \mathbf{a}_j, \mathbf{x} \rangle + \mathbf{e}[j])_{i \leq n} = A \cdot \mathbf{x} + \mathbf{e}.$$

It follows that distinguishing $\mathbf{y}$ from random is perfectly equivalent to breaking the the $\mathcal{H}_L$-conditioned $(\mathcal{M}^c_{n,k}, \mathcal{X}_{n,w})$-$\mathsf{LPN}$ problem.

Reduction to sparse LPN. Plugging the reduction from predicate-conditioned sparse-LPN to sparse-LPN from Lemma 9.16 yields a structured-seed local PRG under the sparse-LPN assumption. However, this comes at a loss $\mathsf{err}(\mathcal{H})$ in the advantage bound ε. We have

$$\mathsf{err}(\mathcal{H}) = \max_{\mathbf{e}} \Pr_{h_0, h_1} [h_0, h_1 \text{ are bad for } \mathbf{e}] \leq O\left(\frac{1}{w^L}\right).$$

Therefore, using our concrete formulation of the sparse-LPN assumption (Assumption 1), we obtain a (T, ε, δ)-secure $(c + 2(L+1)d)$-local PRG with $\varepsilon = O(1/w^L)$ and $\delta = \Omega(k^{\eta})^{c-2}$, for a suitable constant $\eta > 0$.

9.5.10. Structured-seed local PRGs beyond quadratic stretch

If we plug our Parameter Set 1 in Section 9.5.9, we obtain a PRG with seed length $|\mathsf{seed}| = O(nn^{1/d}) \cdot k$ and stretch $n = k^{1+\eta}$, where η is a constant arbitrarily close to 1 (using the constant $c(\eta) = 1/(1-\eta)$). As k grows (polynomially), this means that the stretch

can be made as large as $|\mathsf{seed}|^{2-\gamma}$ for an arbitrarily small constant $\gamma = \gamma(\eta, d, \log_n k)$. However, the construction cannot achieve a super-quadratic stretch directly.

In general, the stretch of a polynomial-stretch local PRG can always be extended to an arbitrary polynomial stretch (keeping the locality constant) by composing the PRG with itself [IKOS08]. However, this does not hold anymore for a structured-seed PRG, since the output distribution of the PRG does not match the required seed distribution. Nevertheless, we show in this section that the stretch of our construction can be extended to an arbitrary polynomial via self-composition. At a high level, we leverage the observation that the seed seed of our construction has two components:

- A random bitstring $\mathbf{x}$ of length k, and
- The items $\mathsf{Comp}(\mathbf{v}_i^j)$, of total length $O(w \cdot n^{1/d})$.

Above, the *structured part* of the seed grows only with w, while the *random part* of the seed grows with k. We leverage this observation by making w as small as we possibly can (while keeping the stretch n to be polynomial, $n \geq k^{1+\gamma/2}$ for some constant $\gamma > 0$), and recursively invoke the structured-seed local PRG to generate the length-k random part of the seed. The relevant set of parameters of the sparse-LPN assumption is given in Parameter Set 2; details follow.

Parameters. For simplicity and concreteness, we use the following parameters throughout:

- $\gamma > 0$ denotes an arbitrarily small constant. Fix $L = 1$.
- n is set to $k^{1+\gamma/2}$ and w to $n \cdot k^{\gamma}$, according to Parameter Set 2 (where $c = c(\gamma) \geq 1 \log \gamma/(\log \gamma - 1)$)
- We choose a large dimension $k \geq n^{1/\gamma^2}$ and set $d \geq 1/\gamma + 1/2$. Note that this guarantees that $n \leq k^{\gamma}$ and $n^{1/d} \leq k^{\gamma}$. Therefore, we have $w \cdot n^{1/d} \leq k^{3\gamma}$.

With the set of parameters above, our structured-seed local PRG has the following characteristics:

- Seed length $|\mathsf{seed}| = k + O(k^{3\gamma})$ (where the $O(\cdot)$ hides a factor proportional to $1/\gamma$ and the size-k part is the random part of the seed),
- Stretch $n = k^{1+\gamma/2}$.

We view the construction as *shrinking* n (pseudorandom) bits into a seed of size (dominated by) $k = n^{1/(1+\gamma/2)}$. Let us denote $\theta(\gamma) = 1/(1+\gamma/2) < 1$ this shrinkage parameter.

Construction. Equipped with the parameters above, we are ready to describe our construction. Let s be the target expansion; that is, we want to map $|\mathsf{seed}|$ bits to $|\mathsf{seed}|^s$ bits. Set γ such that $3\gamma\theta(\gamma) = 3\gamma/(1+\gamma/2) = 1/2s$, and let $t = t(s)$ be a constant such that $\theta^t \leq 1/2s$. For readability, we describe the construction as a sequence of t seed-shrinking steps:

- Step 1: fix the target output length n: we aim to generate a pseudorandom vector $\mathbf{x}_0 \in \mathbb{F}_2^n$. Set $k \leftarrow n^\theta$. Define seed_1 to be the seed of a structured-seed local PRG with stretch n, with parameters $(n, k, w, L, d, \gamma, c)$ as defined above. We have

$$\mathsf{seed}_1 = (\mathbf{x}_1, S_1 = (\mathsf{Comp}(\mathbf{v}_i^0, d), \cdots, \mathsf{Comp}(\mathbf{v}_i^L, d))_{i \le w}),$$

  with $|\mathbf{x}_1| = n^\theta$ and $|S_1| = O(n^{3\gamma\theta}) = O(n^{1/2s})$.

- Step 2: in this step, we shrink $\mathbf{x}_1 \in \mathbb{F}_2^{n^\theta}$ using again our structured-seed local PRG. Writing $n_1 = k = n^\theta$, we generate a pseudorandom $\mathbf{x}_1$ using the seed:

$$\mathsf{seed}_2 = (\mathbf{x}_2, S_2),$$

  where $|\mathbf{x}_2| = n_1^\theta = n^{\theta^2}$ and $|S_2| = O(n_1^{1/2s}) \le O(n^{1/2s})$.

- Step i: we maintain the invariant that we generate a pseudorandom vector $\mathbf{x}_{i-1} \in \mathbb{F}_2^{n^{\theta^{i-1}}}$ using a seed $\mathsf{seed}_i = (\mathbf{x}_i, S_i)$ with $|\mathbf{x}_i| = n^{\theta^i}$ and $|S_i| \le O(n^{1/2s})$.

After t steps of the seed-shrinking step, we end up with a final seed

$$\mathsf{seed} = (\mathbf{x}_t, S_t, S_{t-1}, \cdots, S_1),$$

where $|\mathbf{x}_t| \le n^{\theta^t} \le n^{1/2s}$, and $|S_i| \le O(n^{1/2s})$. The total seed length is therefore $O(n^{1/2s})$ (where the $O(\cdot)$ hides a factor t), which is below $n^{1/s}$ for a large enough n. Security follows immediately from a sequence of t hybrids that "undo" the seed-shrinking steps, replacing everytime $\mathbf{x}_{i-1}$ with a random string given a random seed $(\mathbf{x}_i, S_i)$ for the structured-seed local PRG, under the hardness of the $\mathcal{H}_1$-conditioned sparse-LPN problem. Due to the t hybrids, the reduction loses a factor t in the parameters (ε, δ) of the LPN problem. Plugging in the reduction from predicate-conditioned sparse-LPN to sparse-LPN from Lemma 9.16 yields:

Theorem 9.24. *For every $s > 1$, there exists constants γ such that $3\gamma/(1+\gamma/2) = 1/2s$, $\theta = 1/(1+\gamma/2)$, $c \ge 2\log\gamma/(\log\gamma - 1)$, $d \ge 1/\gamma + 1/2$, and t such that $\theta^t \le 1/2s$, for all large enough n, if the $(\mathcal{S}^c, \mathcal{E})$-$\mathsf{LPN}$ problem with dimension n^{θ^i} and number of samples $n^{\theta^{i-1}}$ is $(2^{o(n)}, \varepsilon_i = \Omega(1/n^{2\gamma\theta^i}, \acute{s}\delta_i = \Omega(1/n^{\theta^i(1+\gamma/2)})$-secure for $i = 1$ to t, then there exists a $(T, \sum_{i=1}^t \varepsilon_i, \sum_{i=1}^t \delta_i)$-secure structured-seed constant-locality pseudorandom generator $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ with seed size $|\mathsf{seed}| \le n^{1/s}$.*

9.5.11. Structured-seed local PRGs beyond quadratic stretch

In this section, we show how to achieve a structured-seed local PRG with a polynomial stretch by recursive our structured-seed local PRG.

Let $(w, k) = (w(n), k(n))$ denote respectively the noise weight and dimension of an instance $(A, \mathbf{x}, \mathbf{e}) \in (\mathcal{M}_{n,k}^c, \mathcal{S}_{n,w})$-$\mathsf{LPN}$, and let $n = n(k)$ be the (polynomial) number of samples. For the security of spare-LPN we set the number of samples $n = k^{1+\eta}$, and noise $w = n \cdot k$ for $\eta \in (0, 1)$ (Lemma 9.4.4). Let $c \ge 3$ and d be two constants.

Intuitively, given a local structured-seed PRG $\mathsf{PRG} = (\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG}_{\mathsf{pp}})$ with the security relying on $(\mathcal{M}^c_{n,k}, \mathcal{S}_{n,w})$-LPN problem, we construct a new local PRG with a larger stretch by: taking the output of $\mathsf{PRG}_{\mathsf{pp}}$ and defining it as a part of seed of the new local PRG that have security based on $(\mathcal{M}^c_{n',n}, \mathcal{S}_{n',w'})$-LPN problem ($n', w'$ are chosen such that the spare-LPN problem is hard). Taking advantage of noise compression, we obtain a new local structured seed PRG with the following parameters:

- The size of seed : $k + w \cdot (1 + 2d \cdot \lceil n^{1/d} \rceil) + w' \cdot (1 + 2d \cdot \lceil n' 1/d \rceil) = k + (1 + 2d \cdot \lceil n^{1/d} \rceil)(w + w') = k(1 + \lambda \cdot (1 + k^{\eta})(1 + 2d \cdot \lceil k^{(1+\eta)/d} \rceil)$.
- The stretch $n' = n^{1+\eta} = k^{(1+\eta)^2}$.

We generalize below a structured-seed local PRG whose security reduces to the hardness of the $(\mathcal{M}^c_{n_1,k_1}, \mathcal{S}_{n_1,w_1})$ sparse-LPN assumption in τ-recursive times:

- Global parameters: a repetition $\tau \in \mathbb{N}$, two constants (c, d), the noise weight $w_1 = w_1(k, \lambda)$ of, the dimension $k_1 = k_1(n)$, and the $n_1 = n_1(k_1)$ satisfy Lemma 9.4.4. All global parameters are implicitly passed as inputs to all algorithms.
- $\mathsf{Setup}(1^n)$: for all $i \in [\tau]$,
  - Set $k_i = n_i$ and $n_i = (n_{i-1})^{1+\eta}$. // to make sure spare-LPN assumption hold
  - Sample $(A_i, h^i_0, h^i_i) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, k_i, n_i)$ (Algorithm 5). // to remove the regularity of noise

  Let $(\mathbf{a}^i_1, \cdots, \mathbf{a}^i_n)$ denote the rows of A_i (of Hamming weight c). Output $\mathsf{pp} \leftarrow (A_i, h^i_0, h^i_1)_{i \le [\tau]}$.
- $\mathsf{SampleSeed}(\mathsf{pp})$: Sample $\mathbf{x} \xleftarrow{\$} \mathbb{F}_2^{k_1}$.

  For all $i \in [\tau]$,
  - Sample $\mathbf{e}_i \xleftarrow{\$} \mathcal{S}_{n_i,w_i}$. // sample the noise vector
  - Define $((\mathsf{Comp}(\mathbf{v}^0_{i,j}, d), \mathsf{Comp}(\mathbf{v}^1_{i,j}, d))_{j \le w_i}) \xleftarrow{\$} \mathsf{SampleSeed}(\mathbf{e}_i, h^i_0, h^i_1)$ (Algorithm 2). // compress noise using Comp and 2-choice hash

  Output $\mathsf{seed} \leftarrow (\mathbf{x}, (\mathsf{Comp}(\mathbf{v}^0_{i,j}, d), \mathsf{Comp}(\mathbf{v}^1_{i,j}, d))_{i \le \tau, j \le w_i})$.
- $\mathsf{PRG}_{\mathsf{pp}}(\mathsf{seed})$: parse seed as $(\mathbf{x}, (\mathsf{Comp}(\mathbf{v}^0_{i,j}, d), \mathsf{Comp}(\mathbf{v}^1_{i,j}, d))_{j \le w_i})$ for all $i \le \tau$.

  For $i = 1$ to τ:
  - Compute $\mathbf{y}_i = \mathsf{PRG}_{\mathsf{pp}}(\mathbf{x}, \mathsf{Comp}(\mathbf{v}^0_{i,j}, d), \mathsf{Comp}(\mathbf{v}^1_{i,j}, d))_{j \le w_i})$ (Algorithm 3). // Output of each PRG
  - Set $\mathbf{x} := \mathbf{y}_i$. // define the output of PRG at i-th iteration as the randomness input of $i + 1$-th iteration

  Output $\mathbf{y}^\tau := (y_1, \cdots, y_{n_\tau})$.

Theorem 9.25. *Assume the (T, ε, δ)-hardness of the $(\mathcal{M}^c_{n,k}, \mathcal{S}_{n,w})$-LPN problem, for some constant $c \geq 3$. Let $d, \tau \geq 2$ be a constant. Then there exists a $(?, \varepsilon, \delta - 1/2^n)$-secure structured-seed local pseudorandom generator $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ with the following characteristics:*

- Parameter size: $|\mathsf{pp}| = \sum_{i=1}^{\tau} n_i \cdot (c \log k_i + 2 \log w_i)$;
- Seed length: $|\mathsf{seed}| = k_i + \sum_{i=1}^{\tau} w_i \cdot (1 + 2d \cdot \lceil n_i^{1/d} \rceil)$;
- Stretch: $n = k^{(1+\eta)^\tau}$;
- Locality: $\tau(c + 4d)$.

In addition, the algorithms have the following running time:

- $\mathsf{Setup}(1^n)$ *runs in time* $n \cdot \tilde{O}(\sum_{i=1}^{\tau}(n_i + nw_i))$,
- $\mathsf{SampleSeed}(\mathsf{pp})$ *runs in time* $O(\sum_{i=1}^{\tau}(k_i + w_i \cdot (n(\log w_i + \log n_i) + n_i^{1/d}))$.

9.6. A Structured-Seed Local PRG from Expand-Accumulate Codes

This section shows how to achieve concrete security for our structured-seed local PRG based on another variant of LPN assumption, *i.e.*, Expanded-Accumulate LPN ($\mathsf{EA-LPN}$).

Definition 9.11 (Binary Accumulator Matrix [BCGIKRS22])**.** For a positive integer k, the accumulator matrix $H \in \mathbb{F}_2^{k \times k}$ is the matrix with 1's on and below the main diagonal, and 0's elsewhere. In particular, if $H \cdot \mathbf{x} = \mathbf{y}$ with $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^k$, we have the following relations:

$$y_i := \bigoplus_{j=1}^{i} x_j \quad \forall i \in [k] \qquad\qquad y_i := x_i \oplus y_{i-1} \quad \forall 2 \leq i \leq k.$$

Definition 9.12 (Expand-Accumulate (EA) codes [BCGIKRS22])**.** For any integer $k \in \mathbb{N}$, let $n = n(k)$ be a polynomial. For a desired density $p \in (0, 1)$, a generator matrix for an expand-accumulate (EA) code is sampled as follows:

- Sample row vectors $\mathbf{b}_1^\intercal, \ldots, \mathbf{b}_n^\intercal \xleftarrow{\$} \mathcal{B}_{k,k \cdot p}$ independent and define

$$B = \begin{bmatrix} \text{———} & \mathbf{b}_1^\intercal & \text{———} \\ \text{———} & \mathbf{b}_2^\intercal & \text{———} \\ \vdots & \vdots & \vdots \\ \text{———} & \mathbf{b}_n^\intercal & \text{———} \end{bmatrix}$$

.

- Output the matrix product BH, where $H \in \mathbb{F}_2^{k \times k}$ is the accumulator matrix.

We use $\mathcal{A}_{k,n}^p$ to denote a code sampled from this distribution and the sampling of the corresponding generator matrix is denoted $A \xleftarrow{\$} \mathsf{EA.Setup}(k, n, p)$.

The $\mathsf{EA-LPN}$ assumption is LPN assumption where the matrix $A \in \mathbb{F}_2^{n \times k}$ is sampled from $\mathcal{A}^p = \mathcal{A}_{k,n}^p$. For a suitable parameter set [BCGIKRS22], the assumption $(\mathcal{A}^p, \mathcal{S}_w, \mathbb{F}_2)$-$\mathsf{LPN}(k, n)$ holds.

9.7. Applications

In this section, we identify works that make use of a local PRG in their main theorems and explore the possibility of substituting their PRG with our own construction. These works span a variety of applications: indistinguishability obfuscation (iO) [JLS21], constant-overhead secure computation [IKOS08], sublinear secure computation [BCM23], and hardness of learning [DV21]. At a high level, we can substitute local PRGs with structured-seed local PRGs in these works because they do not rely on the seed being uniform, but only require the seed to be *short*, and the sampling of the seed to be *efficient*.

We note that in contrast, in other papers such as [BCGIKRS23], it is not obvious how to use our PRG out of the box, because the output is being used as input to another invocation of the PRG, and in this case our idea breaks down as the seed is not structured anymore.

9.7.1. Indistinguishability obfuscation

Indistinguishability obfuscation (iO) is a cryptographic primitive that allows to obfuscate the code of a program such that no polynomial-time adversary can distinguish which of two (equal size) functionally equivalent programs has been obfuscated. Code obfuscation has been formalized already in the early 2000s as a cryptographic building block, by Hada [Had00] and Barak *et al.* [BGIRSVY01], along with a number of early positive [Can97; LPS04; Wee05; HRsV07; HMS07] and negative [BGIRSVY01; GK05; Wee05] results. In a recent sequence of breakthrough results culminating with [JLS21], Jain, Lin, and Sahai have shown how to base indistinguishability obfuscation on the subexponential hardness of four assumptions:

- the LWE assumption,
- the learning parity with noise over a general prime field $\mathbb{F}_p$,
- a boolean local PRG in NC^0,
- the Decision Linear assumption on symmetric bilinear groups of prime order.

At the heart of their construction is a sequence of transformations starting from weak forms of functional encryption which are progressively boosted to full-fledged indistinguishability obfuscation. Above, the local PRG is used for constructing a *structured-seed*

PRG. We note that the notion of structured-seed PRGs in [JLS21] differs from the notion of structured-seed local PRG considered in our work. However, it follows by inspection that the construction of structured-seed PRG of [JLS21] goes through identically if the boolean local PRG is replaced by a structured-seed local PRG.

We mention one technicality, though: the local PRG used in [JLS21] needs to have subexponential security. For our construction from regular sparse-LPN, one can reasonably conjecture security against subexponential time adversaries while keeping ε inverse-subexponential: this follows directly from our concrete version of the sparse-LPN assumption (see Assumption 1) by setting T to subexponential in n. However, the value of δ remains always noticeable due to the non-negligible probability of sampling a matrix with a small dual distance. One can make δ negligible using the alternative matrix distribution introduced in [AK19], but this only makes δ *slightly* negligible, while the construction requires δ to be subexponentially small.

We note that a very similar issue happens with (standard) local PRGs, which require an explicit hypergraph with good expansions property, while random hypergraphs will only satisfy the required property with probability $1-1/\mathsf{poly}$. There are two workarounds to this issue. The first one is identical to the solution that was used in [JLS21]: using an explicit choice of the sparse matrix A and assuming the subexponential hardness of sparse-LPN *with respect to this matrix*; then, Assumption 1 implies that *most* choices of A will yield plausible candidates. This works directly, with the caveat that it only provides a non-uniform construction of iO (which would become uniform if an efficiently sampleable distribution over sparse matrices with subexponentially small probability of having a small dual distance is found in the future).

The second solution is to observe that except with subexponentially small probability, if we sample *multiple* parameters pp for a structured-seed local PRG, at least one of them will be secure against subexponential adversaries. Then, as observed in [JLS22], this implies in turn that the construction of functional encryption will yield multiple candidates functional encryption schemes such that except with subexponentially small probability, one of them is subexponentially secure. Then, one can obtain a full-fledged functional encryption scheme out of these schemes using FE combiners (see Remark 3.1 in [JLS22]). We get the following:

Theorem 9.26 (informal)**.** *Assume sub-exponential security of the following assumptions:*

- *the LWE assumption,*
- *the learning parity with noise over a general prime field $\mathbb{F}_p$,*
- *the sparse-LPN assumption with regular noise,*
- *the Decision Linear assumption on symmetric bilinear groups of prime order,*

there exists a (subexponentially secure) indistinguishability obfuscation for all polynomial-size circuits. Further, assuming only polynomial security of the aforementioned assumptions, there exists collusion-resistant public-key functional encryption for all polynomial-size circuits.

We note that the follow-up work of [JLS22] gets rid of the LWE assumption by making a more involved use of the local PRG. It is not immediately obvious how to replace the local PRG by a structured-seed local PRG in their more involved construction, because it requires in particular an affine randomized encoding construction that relies on self-composing the PRG (using its pseudorandom output as a seed), which does not work with structured-seed local PRGs. We had preliminary results in this direction, but in light of the recent concurrent and independent work of [RVV24] that focuses precisely on this application, and which provides a full-fledged solution to overcoming this obstacle (and others), we refrain in this work from pursuing this route further.

9.7.2. Constant-overhead secure computation

The seminal work of Ishai, Kushilevitz, Ostrovsky, and Sahai [IKOS08] showed that assuming polynomial-stretch local pseudorandom generators (and oblivious transfers), any two-party functionality can be securely computed with *constant computational overhead* over the cost of evaluating the functionality in the clear. In this section, we observe that the local PRG in [IKOS08] can be replaced by a structured-seed local PRG. We summarize the result in Theorem 9.27 below.

Theorem 9.27. *Assume the existence of a polynomial-stretch structured-seed local PRG in* NC^0*, denoted as* $G : \{0,1\}^n \to \{0,1\}^m$*, and of a standard OT protocol. Given a family of circuit* $C = \{C_n\}$ *of polynomial size* $s(n)$ *that defines a two-party computation functionality* f*, there exists a two-party protocol* π_f *that realizes* f *in the semi-honest setting where each party in* π_f *can be implemented by a circuit of size* $O(s(n))$*.*

As a direct corollary, we obtain that secure two-party computation with constant computational overhead can be based on (oblivious transfer and) the hardness of regular sparse-LPN, or that of $\mathcal{H}$-conditioned sparse-LPN, diversifying the set of assumptions under which extreme efficiency can be achieved in secure computation.

We sketch the proof of Theorem 9.27 below. The construction of [IKOS08] uses the following sequence of steps:

1. Using the GMW protocol [GMW87b], given black-box access to $O(s)$ oblivious transfers, any two-party computation for a circuit of size s can be securely evaluated using $O(s)$ bits of communication and $O(s)$ bit operations. Hence, building constant-overhead secure computation reduces to the problem of constructing $O(s)$ OTs with a constant computational overhead. In particular, denoting $p = O(s)$, [IKOS08] shows how to construct p bit-OTs using a local PRG and $\sqrt{p}$ OT instances (on strings of length $\sqrt{p}$) where each party can be implemented by a circuit of size $O(p)$.

2. Let g be a functionality parametrized with a local PRG G which, on input a seed seed from the receiver and p pairs of bits $(\sigma_0^i, \sigma_1^i)_{i \le p}$ from the sender, outputs $(\sigma^i_{G(\mathsf{seed})_i})_{i \le p}$ to the receiver. Then, given black-box access to g, there is a constant-overhead construction of a secure protocol to generate p (chosen) bit-OTs. The protocol follows from the standard information-theoretic derandomization of OT

with random selection bits [Bea92], using $G(\mathsf{seed})$ instead of a random mask to hide the selection bits. Since $G(\mathsf{seed})$ can be computed in linear time and the derandomization involves only $O(p)$ XORs and ANDs, the claim follows.

3. The core component of the reduction is the constant-overhead reduction from securely implementing g to a black-box access to $\sqrt{p}$ OTs on strings of total length $O(p)$. This reduction uses decomposable randomized encodings and builds upon the fact that $g \in \mathsf{NC}^0$.

Above, step 1 and 3 remain perfectly identical if G is replaced by a structured-seed local PRG: in particular, g has the description of the stretching algorithm PRG hardcoded, which is in NC^0, and is oblivious to how the seed seed was sampled. The only difference is in step 2, where the receiver must be instructed to sample $\mathsf{seed} \xleftarrow{\$} \mathsf{SampleSeed}(\mathsf{pp})$ instead of picking a random seed. But since the cost of running $\mathsf{SampleSeed}$ is sublinear in p, this has no effect on the computational complexity of the protocol.

We mention two additional minor technicalities:

- The construction of [IKOS08] is described using a quadratic-stretch local PRG, which is without loss of generality since a local PRG with arbitrary polynomial stretch can be extended to quadratic stretch via self-composition. However, our structured-seed local PRG achieves only *near* quadratic stretch, and structured-seed local PRGs cannot be self-composed. Nevertheless, the assumption of (strict) quadratic stretch in [IKOS08] was made only for notational convenience, and can be generalized in a straightforward way to work with a PRG with a smaller (polynomial) stretch. The reduction then invokes $O(p^{1/2+\varepsilon})$ string-OTs on strings of total length $O(p)$, where ε is such that the local PRG stretches $O(p^{1/2+\varepsilon})$ bits into p bits.

- After completing the reduction, it remains to implement the $O(p^{1/2+\varepsilon})$ string-OTs on strings of total length $O(p)$ with constant computational overhead. This relies again on a constant-overhead PRG: given any pair of strings (α_0, α_1) of length $O(p^{1/2-\varepsilon})$, the sender samples two ℓ-bit seeds $(\mathsf{seed}_0, \mathsf{seed}_1)$ for a local PRG. The sender and the receiver (with bit b) use an ℓ-bit string-OT to let the receiver learn seed_b, using $\mathsf{poly}(n) \cdot \ell$ computation. Then, the sender sends $G(\mathsf{seed}_0) \oplus \alpha_0, G(\mathsf{seed}_1) \oplus \alpha_1$, and the receiver unmasks α_b. The total computation per OT scales as $\mathsf{poly}(n) \cdot \ell + O(|\alpha_0|+|\alpha_1|)$, hence an overall cost of $\mathsf{poly}(n) \cdot \ell \cdot p^{1/2+\varepsilon} + O(p)$. Now, using a structured-seed local PRG with any polynomial stretch yields $\ell = O(p^{(1/2-\varepsilon)\cdot\gamma})$ for some $\gamma < 1$, hence $\mathsf{poly}(n) \cdot \ell \cdot p^{1/2+\varepsilon} = o(p)$ for a large enough p. Here again, an arbitrary structured-seed local PRG with polynomial stretch suffices.

9.7.3. Sublinear secure computation and compact HSS

Homomorphic secret sharing (HSS) was introduced in the work of [BGI16a] as an alternative to fully homomorphic encryption to achieve secure computation with sublinear communication. At a high level, an N-party HSS for a class of functions $\mathcal{F}$ allows to

share an input x such that for every function $f \in \mathcal{F}$, each party with input share x_i can locally compute y_i such that $(y_1, \cdots, y_N)$ form additive shares of $y = f(x)$. A *compact* HSS is an HSS where the share size and sharing algorithm runtime are $O(|x|) + \mathsf{poly}(n)$. Combined with a generic MPC protocol with linear communication overhead to securely run the sharing algorithm, a compact HSS scheme immediately gives rise to a secure N-party protocol with essentially optimal communication $O(N \cdot (|x|+|y|)) + \mathsf{poly}(n)$ for every function $f \in \mathcal{F}$.

A standard approach to build compact HSS from HSS is to use a "hybrid encapsulation" trick: to share a long input x, share a short seed seed with HSS among the parties, and reveal $u = x \oplus G(\mathsf{seed})$ to everyone, where G is a PRG. If the PRG runs in linear time, the share size and runtime of sharing are clearly $O(|x|) + \mathsf{poly}(n)$. Then, to get shares of $f(x)$, the parties homomorphically evaluate $g_u(\mathsf{seed}) := f(u \oplus G(\mathsf{seed})) = f(x)$. This approach works as long as $g_u \in \mathcal{F}$. In particular, this means that if the HSS scheme supports a very low function class $\mathcal{F}$, the PRG G needs to belong to a very low complexity class.

The recent work of [BCM23] showed how to achieve sublinear secure computation and compact HSS from assumptions that were not previously known to imply it. In particular, they show:

Theorem 9.28 (Theorem 32 in [BCM23])**.** *Assuming the superpolynomial hardness of DCR and the existence of PRGs with constant locality, there exists a four-party HSS scheme for the class of loglog-depth circuits with n inputs; the HSS scheme has share size $n \cdot (1 + o(1))$. Furthermore, there exists a protocol with communication complexity $n \cdot (4 + o(1))$ (for large enough n) for securely realizing the four-party functionality that generates HSS shares of the concatenation of the parties' inputs.*

It is immediate from the description of the construction that Theorem 32 in [BCM23] extends directly to the setting where a *structured-seed* constant-locality PRG (with arbitrarily small polynomial stretch) is used instead: in their construction, each party locally samples a short seed seed_i and a generic secure computation protocol is ran on their concatenation $(\mathsf{seed}_1||\mathsf{seed}_2||\mathsf{seed}_3||\mathsf{seed}_4)$. The only impact of using a structured-seed local PRG is that the parties will locally run $\mathsf{SampleSeed}$ instead of sampling their seed uniformly, which has no influence on the correctness, security, of communication efficiency of the protocol. As a consequence, we immediately obtain the following corollary:

Corollary 9.29. *Assuming the superpolynomial hardness of DCR and the hardness of the regular sparse-LPN assumption (or, alternatively, of the $\mathcal{H}$-conditioned sparse-LPN assumption), there exists a four-party HSS scheme for the class of loglog-depth circuits with n inputs; the HSS scheme has share size $n \cdot (1 + o(1))$. Furthermore, there exists a protocol with communication complexity $n \cdot (4 + o(1))$ (for large enough n) for securely realizing the four-party functionality that generates HSS shares of the concatenation of the parties' inputs.*

Combining this corollary with the compiler of [BCM23] from N-party compact HSS to $(N+1)$-party secure computation with sublinear communication yields a 5-party protocol

with sublinear communication $O(s/\log\log s)$ for all layered circuits of size s under the same assumptions as above.[2]

9.7.4. Hardness of learning

PAC learning [Val84] is the algorithmic problem of finding a hypothesis that predicts the output of an unknown class of functions with high probability. The hardness of learning focuses on showing a learning algorithm's ability to return a hypothesis. Daniely and Vardi in [DV21] recently proved several hardness of learning results based on the assumption that local PRGs with polynomial stretch and constant distinguishing advantage exist. We show that our structured seed local PRG can be used in place of their PRG, obtaining hardness-of-learning results from the sparse-LPN assumption.

Definition 9.13 (Predicate)**.** Given a structured-seed ℓ-local PRG $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ with input size k, and stretch n, we let P denote the predicate such that for all $i \in [n]$, there exists a subset $S_i \subset [k]$ of size $|S_i| \leq \ell$ such that for all $x \in \mathsf{Supp}(\mathsf{SampleSeed}(\mathsf{pp}))$, defining $y = \mathsf{PRG}(x)$, we have $y_i = P(x[S_i])$.

Remark 8. In general, an ℓ-local PRG only guarantees that for each output bit y_i, there is a predicate P_i and a size-ℓ subset S_i of the bits of the seed x such that $y_i = P_i(x[S_i])$. However, assuming a single predicate P is without of generality when the PRG has polynomial stretch: since there are at most 2^{2^ℓ} possible predicates P_i on ℓ-bit inputs, and ℓ is a constant, setting P to be the most frequent P_i and keeping only the output bits computed using P yields an ℓ-local PRG with polynomial stretch (reduced by a constant factor at most 2^{2^ℓ}) and a single global predicate P. Furthermore, we note that our constructions from sparse-LPN directly have a single global predicate.

9.7.4.1. DNFs.

We prove that formulas in disjunctive normal form with $\omega(1)$ terms cannot be efficiently PAC-learned assuming the sparse-LPN assumption:

Theorem 9.30 (Theorem 3.1 in [DV21])**.** *Under the assumptions of Theorem 9.5.11, for every $q(n) = \omega(1)$ there is no efficient algorithm that PAC-learns DNF formulas with n variables and $q(n)$ terms.*

We note that, contrary with the other applications discussed in this section, we only need to assume a structured-seed local PRG with constant (ε, δ). Therefore, we can rely directly on sparse-LPN rather than regular sparse-LPN via our reduction from Lemma 9.16. However, the result also crucially requires a local PRG with *arbitrary polynomial stretch*. Fortunately, one can achieve an arbitrary polynomial stretch under sparse-LPN via the construction of Section 9.5.11.

[2] The compiler of [BCM23] requires assuming the hardness of DCR and LPN as it relies on the (DCR+LPN)-based construction of correlated symmetric PIR from [BCM22]. However, their construction can be instantiated with any suitable variant of LPN, including sparse-LPN, hence it requires only assumptions that are redundant with the one we already assume.

At the heart of the proof of Theorem 3.1 in [DV21] is the following clever idea: let ℓ be a constant, and let $P : \{0,1\}^\ell \to \{0,1\}$ be an ℓ-local predicate. Given a size-ℓ subset $S = \{s_1, \cdots, s_\ell\} \subset [n]$, write $\mathbf{v}_i := \mathsf{unit}_n(s_i) \in \{0,1\}^n$ for $i = 1$ to ℓ. Then, consider the following formula ψ:

$$\psi(\mathbf{v}_1, \cdots, \mathbf{v}_\ell) = \bigvee_{\mathbf{x}: P(\mathbf{x})=1} \bigwedge_{i \le \ell} \bigwedge_{j:\mathsf{seed}_j \neq x_i} \bar{v}_{i,j}.$$

We have

$$\begin{aligned}
\psi(\mathbf{v}_1, \cdots, \mathbf{v}_\ell) = 1 &\iff \exists \mathbf{x} \in P^{-1}(1), \forall i \le \ell, \forall \mathsf{seed}_j \neq x_i, \bar{v}_{i,j} = 1 \\
&\iff \exists \mathbf{x} \in P^{-1}(1), \forall i \le \ell, \forall \mathsf{seed}_j \neq x_i, \mathsf{unit}_n(s_i)_j = 0 \\
&\iff \exists \mathbf{x} \in P^{-1}(1), \forall i \le \ell, \forall \mathsf{seed}_j \neq x_i, s_i \neq j \\
&\iff \exists \mathbf{x} \in P^{-1}(1), \forall i \le \ell, \mathsf{seed}_{s_i} = x_i \\
&\iff \exists \mathbf{x} \in P^{-1}(1), \mathsf{seed}_S = \mathbf{x} \\
&\iff P(\mathsf{seed}_S) = 1.
\end{aligned}$$

This means that given a predicate P and a seed $\mathbf{x}$, it is possible to hardcode $(P, \mathbf{x})$ in a DNF ϕ such that for every size-ℓ subset S, there is an encoding $\mathsf{Encode}(S) = (\mathbf{v}_1, \cdots, \mathbf{v}_\ell)$ such that $\psi(\mathsf{Encode}(S)) = P(\mathbf{x}[S])$.

Now, we explain how to adapt the proof of Theorem 3.1 in [DV21] to structured-seed local PRGs. We need the following assumption:

Assumption 2. For every constant $s > 1$, there exists a constant ℓ such that there exists a $(T, 1/6, 1/6)$-secure structured-seed ℓ-local PRG $(\mathsf{Setup}, \mathsf{SampleSeed}, \mathsf{PRG})$ with predicate P, mapping k bits to k^s bits, for every $T = \mathsf{poly}(n)$.

By Theorem 9.5.11, the assumption above is implied by the sparse-LPN assumption. Then, let $\mathcal{A}$ be a PPT adversary that PAC-learns DNF formulas with k variables and $q = \omega(1)$ terms. Let Q denote the number of queries to the DNF oracle made by $\mathcal{A}$, and set s such that $k^s > 100Q^2$. Define the following distribution $\mathcal{D}$:

- Sample $\mathsf{pp} \leftarrow \mathsf{Setup}(1^n)$.
- For any index $i \le k^s$, let S_i denote the size-ℓ subset of the bits of seed used by $\mathsf{PRG}_{\mathsf{pp}}(\mathsf{seed})$ (note that S_i is independent of the particular choice of seed, but might depend on pp).
- Define $\mathcal{D} = \mathcal{D}_{\mathsf{pp}}$ to be the distribution that samples $i \xleftarrow{\$} [k^s]$ and outputs $z = \mathsf{Encode}(S_i)$.

Now, sample $\mathsf{seed} \leftarrow \mathsf{SampleSeed}(\mathsf{pp})$, and let ψ be the DNF (with seed, P hardcoded) encoding the computation of the mapping $\mathsf{PRG}_{\mathsf{pp}}(\mathsf{seed})_i = P(\mathsf{seed}_{S_i}) = \psi(\mathsf{Encode}(S_i))$. Note that ψ is a DNF formula with at most 2^ℓ terms. Given Q samples $(z_i, \psi(z_i))_{i \le Q}$ (the training set), the adversary $\mathcal{A}$ returns a hypothesis h with, with small probability, has a small error on the training set. Note that except with probability at most $1/100$, there are no collisions among the queries. Now, given h, it is straightforward to distinguish the next sample $(z_{Q+1}, \psi(z_{Q+1}))$ from random with high probability.

9.7.4.2. Other classes.

Because any function represented by a DNF formula with $q(n)$ terms can also be represented by a polynomial threshold function over $\{0,1\}^n$ with $q(n)$ monomials, assuming sparse-LPN, the following corollary follows from Theorem 9.30.

Corollary 9.31 (Corollary 3.2 in [DV21])**.** *For all $q(n) = \omega(1)$ there is no efficient algorithm that learns $q(n)$-sparse polynomial threshold functions over $\{0,1\}^n$.*

One can also consider $\omega(1)$-sparse $GF(2)$ polynomials over $\{0,1\}^n$ which are simply a sum of $\omega(1)$ monomials modulo 2.

Theorem 9.32. *For all $q(n) = \omega(1)$ there is no efficient algorithm that learns $q(n)$-sparse GF(2) polynomials over $\{0,1\}^n$.*

Bibliography

[ABGKR14] Adi Akavia, Andrej Bogdanov, Siyao Guo, Akshay Kamath, and Alon Rosen. *Candidate weak pseudorandom functions in* $\mathsf{AC}^0 \circ MOD_2$. In: *ITCS 2014: 5th Conference on Innovations in Theoretical Computer Science*. Jan. 2014. DOI: 10.1145/2554797.2554821.

[ABR16] Benny Applebaum, Andrej Bogdanov, and Alon Rosen. *A Dichotomy for Local Small-Bias Generators*. In: *Journal of Cryptology* 3 (July 2016). DOI: 10.1007/s00145-015-9202-8.

[ADEL22] Thomas Attema, Vincent Dunning, Maarten H. Everts, and Peter Langenkamp. *Efficient Compiler to Covert Security with Public Verifiability for Honest Majority MPC*. In: *ACNS 22: 20th International Conference on Applied Cryptography and Network Security*. June 2022. DOI: 10.1007/978-3-031-09234-3_33.

[ADINZ17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. *Secure Arithmetic Computation with Constant Computational Overhead*. In: *Advances in Cryptology – CRYPTO 2017, Part I*. Aug. 2017. DOI: 10.1007/978-3-319-63688-7_8.

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. *An Algebraic Framework for Silent Preprocessing with Trustless Setup and Active Security*. In: *Advances in Cryptology – CRYPTO 2022, Part IV*. Aug. 2022. DOI: 10.1007/978-3-031-15985-5_15.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. *Cryptography in* NC^0. In: *45th Annual Symposium on Foundations of Computer Science*. Oct. 2004. DOI: 10.1109/FOCS.2004.20.

[AIK08] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. *On pseudorandom generators with linear stretch in NC 0*. In: *Computational Complexity* 1 (2008).

[AJLTVW12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. *Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE*. In: *Advances in Cryptology – EUROCRYPT 2012*. Apr. 2012. DOI: 10.1007/978-3-642-29011-4_29.

[AK19] Benny Applebaum and Eliran Kachlon. *Sampling Graphs without Forbidden Subgraphs and Unbalanced Expanders with Negligible Error*. In: *60th Annual Symposium on Foundations of Computer Science*. Nov. 2019. DOI: 10.1109/FOCS.2019.00020.

[AL07] Yonatan Aumann and Yehuda Lindell. *Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries*. In: *TCC 2007: 4th Theory of Cryptography Conference*. Feb. 2007. DOI: 10.1007/978-3-540-70936-7_8.

[AL16] Benny Applebaum and Shachar Lovett. *Algebraic attacks against random local functions and their countermeasures*. In: *48th Annual ACM Symposium on Theory of Computing*. June 2016. DOI: 10.1145/2897518.2897554.

[Ale03] Michael Alekhnovich. *More on Average Case vs Approximation Complexity*. In: *44th Annual Symposium on Foundations of Computer Science*. Oct. 2003. DOI: 10.1109/SFCS.2003.1238204.

[AOP20] Bar Alon, Eran Omri, and Anat Paskin-Cherniavsky. *MPC with Friends and Foes*. In: *Advances in Cryptology – CRYPTO 2020, Part II*. Aug. 2020. DOI: 10.1007/978-3-030-56880-1_24.

[App12] Benny Applebaum. *Pseudorandom generators with long stretch and low locality from random local one-way functions*. In: *44th Annual ACM Symposium on Theory of Computing*. May 2012. DOI: 10.1145/2213977.2214050.

[App15] Benny Applebaum. *The Cryptographic Hardness of Random Local Functions – Survey*. Cryptology ePrint Archive, Report 2015/165. 2015. URL: https://eprint.iacr.org/2015/165.

[APY20] Ittai Abraham, Benny Pinkas, and Avishay Yanai. *Blinder - Scalable, Robust Anonymous Committed Broadcast*. In: *ACM CCS 2020: 27th Conference on Computer and Communications Security*. Nov. 2020. DOI: 10.1145/3372297.3417261.

[ASY22] Damiano Abram, Peter Scholl, and Sophia Yakoubov. *Distributed (Correlation) Samplers: How to Remove a Trusted Dealer in One Round*. In: *Advances in Cryptology – EUROCRYPT 2022, Part I*. May 2022. DOI: 10.1007/978-3-031-06944-4_27.

[BBCCDS24a] 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 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024*. https://eprint.iacr.org/2024/429. 2024.

[BBCCDS24b] Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, Clément Ducros, and Sacha Servan-Schreiber. *FOLEAGE: F4 OLE-Based Multi-Party Computation for Boolean Circuits*. In: *Cryptology ePrint Archive* (2024).

[BBHP22] Michael Backes, Pascal Berrang, Lucjan Hanzlik, and Ivan Pryvalov. *A framework for constructing Single Secret Leader Election from MPC*. Cryptology ePrint Archive, Report 2022/1040. 2022. URL: https://eprint.iacr.org/2022/1040.

[BCCD23] Maxime Bombar, Geoffroy Couteau, Alain Couvreur, and Clément Ducros. *Correlated Pseudorandomness from the Hardness of Quasi-Abelian Decoding*. In: *Advances in Cryptology – CRYPTO 2023, Part IV*. Aug. 2023. DOI: 10.1007/978-3-031-38551-3_18.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. *Compressing Vector OLE*. In: *ACM CCS 2018: 25th Conference on Computer and Communications Security*. Oct. 2018. DOI: 10.1145/3243734.3243868.

[BCGIKRS19] 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: 26th Conference on Computer and Communications Security*. Nov. 2019. DOI: 10.1145/3319535.3354255.

[BCGIKRS22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. *Correlated Pseudorandomness from Expand-Accumulate Codes*. In: *Advances in Cryptology – CRYPTO 2022, Part II*. Aug. 2022. DOI: 10.1007/978-3-031-15979-4_21.

[BCGIKRS23] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. *Oblivious Transfer with Constant Computational Overhead*. In: *Advances in Cryptology – EUROCRYPT 2023, Part I*. Apr. 2023. DOI: 10.1007/978-3-031-30545-0_10.

[BCGIKS19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. *Efficient Pseudorandom Correlation Generators: Silent OT Extension and More*. In: *Advances in Cryptology – CRYPTO 2019, Part III*. Aug. 2019. DOI: 10.1007/978-3-030-26954-8_16.

[BCGIKS20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. *Correlated Pseudorandom Functions from Variable-Density LPN*. In: *61st Annual Symposium on Foundations of Computer Science*. Nov. 2020. DOI: 10.1109/FOCS46700.2020.00103.

[BCGIKS20b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. *Efficient Pseudorandom Correlation Generators from Ring-LPN*. In: *Advances in Cryptology – CRYPTO 2020, Part II*. Aug. 2020. DOI: 10.1007/978-3-030-56880-1_14.

[BCGIO17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. *Homomorphic Secret Sharing: Optimizations and Applications*. In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Oct. 2017. DOI: 10.1145/3133956.3134107.

[BCM22] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. *Sublinear Secure Computation from New Assumptions*. In: *TCC 2022: 20th Theory of Cryptography Conference, Part II*. Nov. 2022. DOI: 10.1007/978-3-031-22365-5_5.

[BCM23] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. *Sublinear-Communication Secure Multiparty Computation Does Not Require FHE*. In: *Advances in Cryptology – EUROCRYPT 2023, Part II*. Apr. 2023. DOI: 10.1007/978-3-031-30617-4_6.

[BCM24] Dung Bui, Geoffroy Couteau, and Nikolas Melissaris. *Structured-Seed Local Pseudorandom Generators and their Applications*. Cryptology ePrint Archive, Report 2024/1027. 2024. URL: https://eprint.iacr.org/2024/1027.

[BCMPR24] 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, Part VI*. May 2024. DOI: 10.1007/978-3-031-58751-1_4.

[BDD20] Carsten Baum, Bernardo David, and Rafael Dowsley. *A Framework for Universally Composable Publicly Verifiable Cryptographic Protocols*. Cryptology ePrint Archive, Report 2020/207. 2020. URL: https://eprint.iacr.org/2020/207.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. *Semi-homomorphic Encryption and Multiparty Computation*. In: *Advances in Cryptology – EUROCRYPT 2011*. May 2011. DOI: 10.1007/978-3-642-20465-4_11.

[BDSW23] Carsten Baum, Samuel Dittmer, Peter Scholl, and Xiao Wang. *Sok: vector OLE-based zero-knowledge protocols*. In: *Designs, Codes and Cryptography* 11 (2023). DOI: 10.1007/s10623-023-01292-8.

[Bea92] Donald Beaver. *Efficient Multiparty Protocols Using Circuit Randomization*. In: *Advances in Cryptology – CRYPTO'91*. Aug. 1992. DOI: 10.1007/3-540-46766-1_34.

[Bea96] Donald Beaver. *Correlated Pseudorandomness and the Complexity of Private Computations*. In: *28th Annual ACM Symposium on Theory of Computing*. May 1996. DOI: 10.1145/237814.237996.

[BEHG20] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. *Single secret leader election*. In: *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 2020.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. *Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)*. In: *20th Annual ACM Symposium on Theory of Computing*. May 1988. DOI: 10.1145/62212.62222.

[BGGJKRS18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. *Threshold Cryptosystems from Threshold Fully Homomorphic Encryption*. In: *Advances in Cryptology – CRYPTO 2018, Part I*. Aug. 2018. DOI: 10.1007/978-3-319-96884-1_19.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. *Functional Signatures and Pseudorandom Functions*. In: *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*. Mar. 2014. DOI: 10.1007/978-3-642-54631-0_29.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. *Function Secret Sharing*. In: *Advances in Cryptology – EUROCRYPT 2015, Part II*. Apr. 2015. DOI: 10.1007/978-3-662-46803-6_12.

[BGI16a] Elette Boyle, Niv Gilboa, and Yuval Ishai. *Breaking the Circuit Size Barrier for Secure Computation Under DDH*. In: *Advances in Cryptology – CRYPTO 2016, Part I*. Aug. 2016. DOI: 10.1007/978-3-662-53018-4_19.

[BGI16b] Elette Boyle, Niv Gilboa, and Yuval Ishai. *Function Secret Sharing: Improvements and Extensions*. In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. Oct. 2016. DOI: 10.1145/2976749.2978429.

[BGIK22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. *Programmable Distributed Point Functions*. In: *Advances in Cryptology – CRYPTO 2022, Part IV*. Aug. 2022. DOI: 10.1007/978-3-031-15985-5_5.

[BGIRSVY01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. *On the (Im)possibility of Obfuscating Programs*. In: *Advances in Cryptology – CRYPTO 2001*. Aug. 2001. DOI: 10.1007/3-540-44647-8_1.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. *Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)*. In: *20th Annual ACM Symposium on Theory of Computing*. May 1988. DOI: 10.1145/62212.62213.

[BKR23] Andrej Bogdanov, Pravesh K. Kothari, and Alon Rosen. *Public-Key Encryption, Local Pseudorandom Generators, and the Low-Degree Method*. In: *TCC 2023: 21st Theory of Cryptography Conference, Part I*. Nov. 2023. DOI: 10.1007/978-3-031-48615-9_10.

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. *Homomorphic Secret Sharing from Lattices Without FHE*. In: *Advances in Cryptology – EUROCRYPT 2019, Part II*. May 2019. DOI: 10.1007/978-3-030-17656-3_1.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. *Noise-tolerant learning, the parity problem, and the statistical query model*. In: *32nd Annual ACM Symposium on Theory of Computing*. May 2000. DOI: 10.1145/335305.335355.

[BM97] Mihir Bellare and Daniele Micciancio. *A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost*. In: *Advances in Cryptology – EUROCRYPT'97*. May 1997. DOI: 10.1007/3-540-69053-0_13.

[BMMM20] Nicholas-Philip Brandt, Sven Maier, Tobias Müller, and Jörn Müller-Quade. *Constructing Secure Multi-Party Computation with Identifiable Abort*. Cryptology ePrint Archive, Report 2020/153. 2020. URL: https://eprint.iacr.org/2020/153.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. *The Round Complexity of Secure Protocols (Extended Abstract)*. In: *22nd Annual ACM Symposium on Theory of Computing*. May 1990. DOI: 10.1145/100216.100287.

[BMRS23] Carsten Baum, Nikolas Melissaris, Rahul Rachuri, and Peter Scholl. *Cheater Identification on a Budget: MPC with Identifiable Abort from Pairwise MACs*. Cryptology ePrint Archive, Report 2023/1548. 2023. URL: https://eprint.iacr.org/2023/1548.

[BMRS24] Carsten Baum, Nikolas Melissaris, Rahul Rachuri, and Peter Scholl. *Cheater Identification on a Budget: MPC with Identifiable Abort from Pairwise MACs*. In: *Advances in Cryptology – CRYPTO 2024, Part VIII*. Aug. 2024. DOI: 10.1007/978-3-031-68397-8_14.

[Bog+08] Peter Bogetoft, Dan Lund Christensen, Ivan Damgard, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. *Multiparty Computation Goes Live*. Cryptology ePrint Archive, Report 2008/068. 2008. URL: https://eprint.iacr.org/2008/068.

[BOS16] Carsten Baum, Emmanuela Orsini, and Peter Scholl. *Efficient Secure Multiparty Computation with Identifiable Abort*. In: *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*. Oct. 2016. DOI: 10.1007/978-3-662-53641-4_18.

[BOSS20] Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. *Efficient Constant-Round MPC with Identifiable Abort and Public Verifiability*. In: *Advances in Cryptology – CRYPTO 2020, Part II*. Aug. 2020. DOI: 10.1007/978-3-030-56880-1_20.

[BQ09] Andrej Bogdanov and Youming Qiao. *On the security of goldreich's one-way function*. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. 2009.

[BR17] Andrej Bogdanov and Alon Rosen. *Pseudorandom Functions: Three Decades Later*. Cryptology ePrint Archive, Report 2017/652. 2017. URL: https://eprint.iacr.org/2017/652.

[BW13] Dan Boneh and Brent Waters. *Constrained Pseudorandom Functions and Their Applications*. In: *Advances in Cryptology – ASIACRYPT 2013, Part II*. Dec. 2013. DOI: 10.1007/978-3-642-42045-0_15.

[Can01] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. In: *42nd Annual Symposium on Foundations of Computer Science*. Oct. 2001. DOI: 10.1109/SFCS.2001.959888.

[Can97] Ran Canetti. *Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information*. In: *Advances in Cryptology – CRYPTO'97*. Aug. 1997. DOI: 10.1007/BFb0052255.

[CBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. *Riposte: An Anonymous Messaging System Handling Millions of Users*. In: *2015 IEEE Symposium on Security and Privacy*. May 2015. DOI: 10.1109/SP.2015.27.

[CCMBM24] Miranda Christ, Kevin Choi, Walter McKelvie, Joseph Bonneau, and Tal Malkin. *Accountable Secret Leader Election*. In: *6th Conference on Advances in Financial Technologies (AFT 2024)*. 2024.

[CDKs23] Ran Cohen, Jack Doerner, Yashvanth Kondi, and abhi shelat. *Secure Multiparty Computation with Identifiable Abort from Vindicating Release*. Cryptology ePrint Archive, Report 2023/1136. 2023. URL: https://eprint.iacr.org/2023/1136.

[CDKs24] Ran Cohen, Jack Doerner, Yashvanth Kondi, and abhi shelat. *Secure Multiparty Computation with Identifiable Abort via Vindicating Release*. In: *Advances in Cryptology – CRYPTO 2024, Part VIII*. Aug. 2024. DOI: 10.1007/978-3-031-68397-8_2.

[CDMRR18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. *On the Concrete Security of Goldreich's Pseudorandom Generator*. In: *Advances in Cryptology – ASIACRYPT 2018, Part II*. Dec. 2018. DOI: 10.1007/978-3-030-03329-3_4.

[CEMT14] James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. *On the one-way function candidate proposed by Goldreich*. In: *ACM Transactions on Computation Theory (TOCT)* 3 (2014).

[CFG22] Dario Catalano, Dario Fiore, and Emanuele Giunta. *Adaptively Secure Single Secret Leader Election from DDH*. In: *41st ACM Symposium Annual on Principles of Distributed Computing*. July 2022. DOI: 10.1145/3519270.3538424.

[CFG23] Dario Catalano, Dario Fiore, and Emanuele Giunta. *Efficient and Universally Composable Single Secret Leader Election from Pairings*. In: *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*. May 2023. DOI: 10.1007/978-3-031-31368-4_17.

[CFY17] Robert K. Cunningham, Benjamin Fuller, and Sophia Yakoubov. *Catching MPC Cheaters: Identification and Openability*. In: *ICITS 17: 10th International Conference on Information Theoretic Security*. Nov. 2017. DOI: 10.1007/978-3-319-72089-0_7.

[CGZ20] Ran Cohen, Juan A. Garay, and Vassilis Zikas. *Broadcast-Optimal Two-Round MPC*. In: *Advances in Cryptology – EUROCRYPT 2020, Part II*. May 2020. DOI: 10.1007/978-3-030-45724-2_28.

[Cha88] David Chaum. *The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability*. In: *Journal of Cryptology* 1 (Jan. 1988). DOI: 10.1007/BF00206326.

[Che+21] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, abhi shelat, Muthu Venkitasubramaniam, and Ruihan Wang. *Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority*. In: *2021 IEEE Symposium on Security and Privacy*. May 2021. DOI: 10.1109/SP40001.2021.00025.

[CKMSS24] Geoffroy Couteau, Alexander Koch, Nikolas Melissaris, Sacha Servan-Schreiber, and Peter Scholl. *Compressing Pseudorandom Permutation Correlations*. In Submission. 2024.

[Cle86] Richard Cleve. *Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract)*. In: *18th Annual ACM Symposium on Theory of Computing*. May 1986. DOI: 10.1145/12130.12168.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. *Universally composable two-party and multi-party secure computation*. In: *34th Annual ACM Symposium on Theory of Computing*. May 2002. DOI: 10.1145/509907.509980.

[CM01] Mary Cryan and Peter Bro Miltersen. *On pseudorandom generators in NC 0*. In: *International Symposium on Mathematical Foundations of Computer Science*. 2001.

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. *Constrained Pseudorandom Functions from Homomorphic Secret Sharing*. In: *Advances in Cryptology – EUROCRYPT 2023, Part III*. Apr. 2023. 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: *Advances in Cryptology – CRYPTO 2021, Part III*. Aug. 2021. DOI: 10.1007/978-3-030-84252-9_17.

[CRS03] Artur Czumaj, Chris Riley, and Christian Scheideler. *Perfectly balanced allocation*. In: *International Workshop on Randomization and Approximation Techniques in Computer Science*. 2003.

[CRSW22] Michele Ciampi, Divya Ravi, Luisa Siniscalchi, and Hendrik Waldner. *Round-Optimal Multi-party Computation with Identifiable Abort*. In: *Advances in Cryptology – EUROCRYPT 2022, Part I*. May 2022. DOI: 10.1007/978-3-031-06944-4_12.

[CS10] Octavian Catrina and Amitabh Saxena. *Secure Computation with Fixed-Point Numbers*. In: *FC 2010: 14th International Conference on Financial Cryptography and Data Security*. Jan. 2010. DOI: 10.1007/978-3-642-14577-3_6.

[DEK21] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. *Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security*. In: *USENIX Security 2021: 30th USENIX Security Symposium*. Aug. 2021.

[DFKNT06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. *Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation*. In: *TCC 2006: 3rd Theory of Cryptography Conference*. Mar. 2006. DOI: 10.1007/11681878_15.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. *Spooky Encryption and Its Applications*. In: *Advances in Cryptology – CRYPTO 2016, Part III*. Aug. 2016. DOI: 10.1007/978-3-662-53015-3_4.

[DIJL23a] Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. *Multi-party Homomorphic Secret Sharing and Sublinear MPC from Sparse LPN*. In: *Annual International Cryptology Conference*. 2023.

[DIJL23b] Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. *Multi-party Homomorphic Secret Sharing and Sublinear MPC from Sparse LPN*. In: *Advances in Cryptology – CRYPTO 2023, Part II*. Aug. 2023. DOI: 10.1007/978-3-031-38545-2_11.

[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).

[DMRSY21] Ivan Damgård, Bernardo Magri, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. *Broadcast-Optimal Two Round MPC with an Honest Majority*. In: *Advances in Cryptology – CRYPTO 2021, Part II*. Aug. 2021. DOI: 10.1007/978-3-030-84245-1_6.

[DN14] Ivan Damgård and Jesper Buus Nielsen. *Adaptive versus Static Security in the UC Model*. In: *ProvSec 2014: 8th International Conference on Provable Security*. Oct. 2014. DOI: 10.1007/978-3-319-12475-9_2.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. *Multiparty Computation from Somewhat Homomorphic Encryption*. In: *Advances in Cryptology – CRYPTO 2012*. Aug. 2012. DOI: 10.1007/978-3-642-32009-5_38.

[DRSY23] Ivan Damgård, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. *Minimizing Setup in Broadcast-Optimal Two Round MPC*. In: *Advances in Cryptology – EUROCRYPT 2023, Part II*. Apr. 2023. DOI: 10.1007/978-3-031-30617-4_5.

[Ds17] Jack Doerner and abhi shelat. *Scaling ORAM for Secure Computation*. In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Oct. 2017. DOI: 10.1145/3133956.3133967.

[DV21] Amit Daniely and Gal Vardi. *From Local Pseudorandom Generators to Hardness of Learning*. In: *Proceedings of Thirty Fourth Conference on Learning Theory*. Aug. 2021. URL: https://proceedings.mlr.press/v134/daniely21a.html.

[ECZB21] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. *Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy*. In: *USENIX Security 2021: 30th USENIX Security Symposium*. Aug. 2021.

[EKM17] Andre Esser, Robert Kübler, and Alexander May. *LPN Decoded*. In: *Advances in Cryptology – CRYPTO 2017, Part II*. Aug. 2017. DOI: 10.1007/978-3-319-63715-0_17.

[FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E Skeith. *Homomorphic secret sharing from paillier encryption*. In: *Provable Security: 11th International Conference, ProvSec 2017, Xi'an, China, October 23-25, 2017, Proceedings 11*. 2017.

[FHKS21] Sebastian Faust, Carmit Hazay, David Kretzler, and Benjamin Schlosser. *Generic Compiler for Publicly Verifiable Covert Multi-Party Computation*. In: *Advances in Cryptology – EUROCRYPT 2021, Part II*. Oct. 2021. DOI: 10.1007/978-3-030-77886-6_27.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. *How to Construct Random Functions*. In: *Journal of the ACM* 4 (Oct. 1986). DOI: 10.1145/6490.6503.

[GI14] Niv Gilboa and Yuval Ishai. *Distributed Point Functions and Their Applications*. In: *Advances in Cryptology – EUROCRYPT 2014*. May 2014. DOI: 10.1007/978-3-642-55220-5_35.

[GIKR02] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. *On 2-Round Secure Multiparty Computation*. In: *Advances in Cryptology – CRYPTO 2002*. Aug. 2002. DOI: 10.1007/3-540-45708-9_12.

[GJ04] Philippe Golle and Ari Juels. *Dining Cryptographers Revisited*. In: *Advances in Cryptology – EUROCRYPT 2004*. May 2004. DOI: 10.1007/978-3-540-24676-3_27.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. *On the Impossibility of Obfuscation with Auxiliary Input*. In: *46th Annual Symposium on Foundations of Computer Science*. Oct. 2005. DOI: 10.1109/SFCS.2005.60.

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. *Constant-Round MPC with Fairness and Guarantee of Output Delivery*. In: *Advances in Cryptology – CRYPTO 2015, Part II*. Aug. 2015. DOI: 10.1007/978-3-662-48000-7_4.

[GMW87a] Oded Goldreich, Silvio Micali, and Avi Wigderson. *How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority*. In: *19th Annual ACM Symposium on Theory of Computing*. May 1987. DOI: 10.1145/28395.28420.

[GMW87b] Oded Goldreich, Silvio Micali, and Avi Wigderson. *How to Prove all NP-Statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design*. In: *Advances in Cryptology – CRYPTO'86*. Aug. 1987. DOI: 10.1007/3-540-47721-7_11.

[Gol00] Oded Goldreich. *Candidate One-Way Functions Based on Expander Graphs*. Cryptology ePrint Archive, Report 2000/063. 2000. URL: https://eprint.iacr.org/2000/063.

[Had00] Satoshi Hada. *Zero-Knowledge and Code Obfuscation*. In: *Advances in Cryptology – ASIACRYPT 2000*. Dec. 2000. DOI: 10.1007/3-540-44448-3_34.

[HKKPPP22] Aditya Hegde, Nishat Koti, Varsha Bhat Kukkala, Shravani Patil, Arpita Patra, and Protik Paul. *Attaining GOD Beyond Honest Majority with Friends and Foes*. In: *Advances in Cryptology – ASIACRYPT 2022, Part I*. Dec. 2022. DOI: 10.1007/978-3-031-22963-3_19.

[HMS07] Dennis Hofheinz, John Malone-Lee, and Martijn Stam. *Obfuscation for Cryptographic Purposes*. In: *TCC 2007: 4th Theory of Cryptography Conference*. Feb. 2007. DOI: 10.1007/978-3-540-70936-7_12.

[HN06] Martin Hirt and Jesper Buus Nielsen. *Robust Multiparty Computation with Linear Communication Complexity*. In: *Advances in Cryptology – CRYPTO 2006*. Aug. 2006. DOI: 10.1007/11818175_28.

[HRsV07] Susan Hohenberger, Guy N. Rothblum, abhi shelat, and Vinod Vaikuntanathan. *Securely Obfuscating Re-encryption*. In: *TCC 2007: 4th Theory of Cryptography Conference*. Feb. 2007. DOI: 10.1007/978-3-540-70936-7_13.

[HVW22] Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. *Protecting Distributed Primitives Against Leakage: Equivocal Secret Sharing and More*. In: *3rd Conference on Information-Theoretic Cryptography (ITC 2022)*. 2022.

[IKKP15] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. *Secure Computation with Minimal Interaction, Revisited*. In: *Advances in Cryptology – CRYPTO 2015, Part II*. Aug. 2015. DOI: 10.1007/978-3-662-48000-7_18.

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. *On Combining Privacy with Guaranteed Output Delivery in Secure Multiparty Computation*. In: *Advances in Cryptology – CRYPTO 2006*. Aug. 2006. DOI: 10.1007/11818175_29.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. *Extending Oblivious Transfers Efficiently*. In: *Advances in Cryptology – CRYPTO 2003*. Aug. 2003. DOI: 10.1007/978-3-540-45146-4_9.

[IKNZ23] Yuval Ishai, Mahimna Kelkar, Varun Narayanan, and Liav Zafar. *One-Message Secure Reductions: On the Cost of Converting Correlations*. In: *Advances in Cryptology – CRYPTO 2023, Part I*. Aug. 2023. DOI: 10.1007/978-3-031-38557-5_17.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. *Batch codes and their applications*. In: *36th Annual ACM Symposium on Theory of Computing*. June 2004. DOI: 10.1145/1007352.1007396.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. *Cryptography with constant computational overhead*. In: *40th Annual ACM Symposium on Theory of Computing*. May 2008. DOI: 10.1145/1374376.1374438.

[IKP10] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. *Secure Multiparty Computation with Minimal Interaction*. In: *Advances in Cryptology – CRYPTO 2010*. Aug. 2010. DOI: 10.1007/978-3-642-14623-7_31.

[IOS12] Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. *Identifying Cheaters without an Honest Majority*. In: *TCC 2012: 9th Theory of Cryptography Conference*. Mar. 2012. DOI: 10.1007/978-3-642-28914-9_2.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. *Secure Multi-Party Computation with Identifiable Abort*. In: *Advances in Cryptology – CRYPTO 2014, Part II*. Aug. 2014. DOI: 10.1007/978-3-662-44381-1_21.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. *Founding Cryptography on Oblivious Transfer - Efficiently*. In: *Advances in Cryptology – CRYPTO 2008*. Aug. 2008. DOI: 10.1007/978-3-540-85174-5_32.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. *Indistinguishability obfuscation from well-founded assumptions*. In: *53rd Annual ACM Symposium on Theory of Computing*. June 2021. DOI: 10.1145/3406325.3451093.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. *Indistinguishability Obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0*. In: *Advances in Cryptology – EUROCRYPT 2022, Part I*. May 2022. DOI: 10.1007/978-3-031-06944-4_23.

[Kat07] Jonathan Katz. *On achieving the "best of both worlds" in secure multiparty computation*. In: *39th Annual ACM Symposium on Theory of Computing*. June 2007. DOI: 10.1145/1250790.1250793.

[Kil88] Joe Kilian. *Founding Cryptography on Oblivious Transfer*. In: *20th Annual ACM Symposium on Theory of Computing*. May 1988. DOI: 10.1145/62212.62215.

[Kir11] Paul Kirchner. *Improved Generalized Birthday Attack*. Cryptology ePrint Archive, Report 2011/377. 2011. URL: https://eprint.iacr.org/2011/377.

[KKPG22] Nishat Koti, Varsha Bhat Kukkala, Arpita Patra, and Bhavish Raj Gopal. *PentaGOD: Stepping beyond Traditional GOD with Five Parties*. In: *ACM CCS 2022: 29th Conference on Computer and Communications Security*. Nov. 2022. DOI: 10.1145/3548606.3559369.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. *Universally Composable Synchronous Computation*. In: *TCC 2013: 10th Theory of Cryptography Conference*. Mar. 2013. DOI: 10.1007/978-3-642-36594-2_27.

[KPPS21] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. *SWIFT: Superfast and Robust Privacy-Preserving Machine Learning*. In: *USENIX Security 2021: 30th USENIX Security Symposium*. Aug. 2021.

[KPRS21] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. *Tetrad: Actively Secure 4PC for Secure Training and Inference*. Cryptology ePrint Archive, Report 2021/755. 2021. URL: https://eprint.iacr.org/2021/755.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. *Delegatable pseudorandom functions and applications*. In: *ACM CCS 2013: 20th Conference on Computer and Communications Security*. Nov. 2013. DOI: 10.1145/2508859.2516668.

[LF06] Éric Levieil and Pierre-Alain Fouque. *An Improved LPN Algorithm*. In: *SCN 06: 5th International Conference on Security in Communication Networks*. Sept. 2006. DOI: 10.1007/11832072_24.

[LN17] Yehuda Lindell and Ariel Nof. *A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority*. In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Oct. 2017. DOI: 10.1145/3133956.3133999.

[LPS04] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. *Positive Results and Techniques for Obfuscation*. In: *Advances in Cryptology – EUROCRYPT 2004*. May 2004. DOI: 10.1007/978-3-540-24676-3_2.

[LV17] Alex Lombardi and Vinod Vaikuntanathan. *Limits on the Locality of Pseudorandom Generators and Applications to Indistinguishability Obfuscation*. In: *TCC 2017: 15th Theory of Cryptography Conference, Part I*. Nov. 2017. DOI: 10.1007/978-3-319-70500-2_5.

[LWYY24] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. *The hardness of LPN over any integer ring and field for PCG applications*. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2024.

[Lyu05] Vadim Lyubashevsky. *The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem*. In: 2005.

[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* (2022).

[MRY22] Nikolas Melissaris, Divya Ravi, and Sophia Yakoubov. *Threshold-Optimal MPC With Friends and Foes*. Cryptology ePrint Archive, Report 2022/1526. 2022. URL: https://eprint.iacr.org/2022/1526.

[MRY23] Nikolas Melissaris, Divya Ravi, and Sophia Yakoubov. *Threshold-Optimal MPC with Friends and Foes*. In: *Progress in Cryptology - INDOCRYPT 2023: 24th International Conference in Cryptology in India, Part II*. Dec. 2023. DOI: 10.1007/978-3-031-56235-8_1.

[MST03] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. *On e-Biased Generators in NC0*. In: *44th Annual Symposium on Foundations of Computer Science*. Oct. 2003. DOI: 10.1109/SFCS.2003.1238188.

[NN90] Joseph Naor and Moni Naor. *Small-bias Probability Spaces: Efficient Constructions and Applications*. In: *22nd Annual ACM Symposium on Theory of Computing*. May 1990. DOI: 10.1145/100216.100244.

[NSD22] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. *Spectrum: High-bandwidth anonymous broadcast*. In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 2022.

[OST19] Igor Carboni Oliveira, Rahul Santhanam, and Roei Tell. *Expander-Based Cryptography Meets Natural Proofs*. In: *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*. Jan. 2019. DOI: 10.4230/LIPIcs.ITCS.2019.18.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. *The Rise of Paillier: Homomorphic Secret Sharing and Public-Key Silent OT*. In: *Advances in Cryptology – EUROCRYPT 2021, Part I*. Oct. 2021. DOI: 10.1007/978-3-030-77870-5_24.

[Ove06] Raphael Overbeck. *Statistical Decoding Revisited*. In: *ACISP 06: 11th Australasian Conference on Information Security and Privacy*. July 2006. DOI: 10.1007/11780656_24.

[OW14] Ryan ODonnell and David Witmer. *Goldreich's PRG: evidence for near-optimal polynomial stretch*. In: *Computational Complexity (CCC), 2014 IEEE 29th Conference on*. 2014.

[Pie12] Krzysztof Pietrzak. *Cryptography from learning parity with noise*. In: *International Conference on Current Trends in Theory and Practice of Computer Science*. 2012.

[PR18] Arpita Patra and Divya Ravi. *On the Exact Round Complexity of Secure Three-Party Computation*. In: *Advances in Cryptology – CRYPTO 2018, Part II*. Aug. 2018. DOI: 10.1007/978-3-319-96881-0_15.

[PR19] Arpita Patra and Divya Ravi. *Beyond Honest Majority: The Round Complexity of Fair and Robust Multi-party Computation*. In: *Advances in Cryptology – ASIACRYPT 2019, Part I*. Dec. 2019. DOI: 10.1007/978-3-030-34578-5_17.

[Pra62] Eugene Prange. *The use of information sets in decoding cyclic codes*. In: (1962).

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. *SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension*. In: *Advances in Cryptology – CRYPTO 2019, Part III*. Aug. 2019. DOI: 10.1007/978-3-030-26954-8_13.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. *Efficient Circuit-Based PSI via Cuckoo Hashing*. In: *Advances in Cryptology – EUROCRYPT 2018, Part III*. Apr. 2018. DOI: 10.1007/978-3-319-78372-7_5.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. *A Framework for Efficient and Composable Oblivious Transfer*. In: *Advances in Cryptology – CRYPTO 2008*. Aug. 2008. DOI: 10.1007/978-3-540-85174-5_31.

[Roy22] Lawrence Roy. *SoftSpokenOT: Quieter OT Extension from Small-Field Silent VOLE in the Minicrypt Model*. In: *Advances in Cryptology – CRYPTO 2022, Part I*. Aug. 2022. DOI: 10.1007/978-3-031-15802-5_23.

[RS21] Lawrence Roy and Jaspal Singh. *Large Message Homomorphic Secret Sharing from DCR and Applications*. In: *Advances in Cryptology – CRYPTO 2021, Part III*. Aug. 2021. DOI: 10.1007/978-3-030-84252-9_23.

[RS22] Rahul Rachuri and Peter Scholl. *Le Mans: Dynamic and Fluid MPC for Dishonest Majority*. In: *Advances in Cryptology – CRYPTO 2022, Part I*. Aug. 2022. DOI: 10.1007/978-3-031-15802-5_25.

[RS98] Martin Raab and Angelika Steger. *"Balls into Bins" - A Simple and Tight Analysis*. In: *Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*. 1998. ISBN: 354065142X.

[RVV24] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. *Indistinguishability Obfuscation from Bilinear Maps and LPN Variants*. In: *Cryptology ePrint Archive* (2024).

[Saa07] Markku-Juhani Olavi Saarinen. *Linearization Attacks Against Syndrome Based Hashes*. In: *Progress in Cryptology - INDOCRYPT 2007: 8th International Conference in Cryptology in India*. Dec. 2007. DOI: 10.1007/978-3-540-77026-8_1.

[SB07] Chris Studholme and Ian Blake. *Multiparty Computation to Generate Secret Permutations*. Cryptology ePrint Archive, Report 2007/353. 2007. URL: https://eprint.iacr.org/2007/353.

[SEK03] Sanders, Egner, and Korst. *Fast concurrent access to parallel disks*. In: *Algorithmica* (2003).

[SF16] Gabriele Spini and Serge Fehr. *Cheater Detection in SPDZ Multiparty Computation*. In: *ICITS 16: 9th International Conference on Information Theoretic Security*. Aug. 2016. DOI: 10.1007/978-3-319-49175-2_8.

[SGRP19] Phillipp Schoppmann, Adrià Gascón, Mariana Raykova, and Benny Pinkas. *Make Some ROOM for the Zeros: Data Sparsity in Secure Distributed Machine Learning*. In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Nov. 2019. DOI: 10.1145/3319535.3339816.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. *Distributed Vector-OLE: Improved Constructions and Implementation*. In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Nov. 2019. DOI: 10.1145/3319535.3363228.

[SSS22] Peter Scholl, Mark Simkin, and Luisa Siniscalchi. *Multiparty Computation with Covert Security and Public Verifiability*. In: *3rd Conference on Information-Theoretic Cryptography*. 2022.

[SSY22] Mark Simkin, Luisa Siniscalchi, and Sophia Yakoubov. *On Sufficient Oracles for Secure Computation with Identifiable Abort*. In: *SCN 22: 13th International Conference on Security in Communication Networks*. Sept. 2022. DOI: 10.1007/978-3-031-14791-3_22.

[The24] The OpenSSL Project. *OpenSSL Cryptography and SSL/TLS Toolkit*. https://www.openssl.org/. Accessed: 2024-02-12. 2024.

[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: *Advances in Cryptology – EUROCRYPT 2023, Part I*. Apr. 2023. DOI: 10.1007/978-3-031-30545-0_2.

[UR24] Antoine Urban and Matthieu Rambaud. *Robust Multiparty Computation from Threshold Encryption Based on RLWE*. Cryptology ePrint Archive, Report 2024/1285. 2024. URL: https://eprint.iacr.org/2024/1285.

[Val84] L. G. Valiant. *A theory of the learnable*. In: *Commun. ACM* 11 (Nov. 1984). ISSN: 0001-0782. DOI: 10.1145/1968.1972. URL: https://doi.org/10.1145/1968.1972.

[Von+63] John Von Neumann et al. *Various techniques used in connection with random digits*. In: *John von Neumann, Collected Works* (1963).

[Wag02] David Wagner. *A Generalized Birthday Problem*. In: *Advances in Cryptology – CRYPTO 2002*. Aug. 2002. DOI: 10.1007/3-540-45708-9_19.

[Wee05] Hoeteck Wee. *On obfuscating point functions*. In: *37th Annual ACM Symposium on Theory of Computing*. May 2005. DOI: 10.1145/1060590.1060669.

[WYKW21] 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*. May 2021. DOI: 10.1109/SP40001.2021.00056.

[Yao86] Andrew Chi-Chih Yao. *How to Generate and Exchange Secrets (Extended Abstract)*. In: *27th Annual Symposium on Foundations of Computer Science*. Oct. 1986. DOI: 10.1109/SFCS.1986.25.

[YGJL21] Jing Yang, Qian Guo, Thomas Johansson, and Michael Lentmaier. *Revisiting the concrete security of goldreich's pseudorandom generator*. In: *IEEE Transactions on Information Theory* 2 (2021).