Université Paris Cité

cnrs

Centre National de la Recherche Scientifique
Institut de Recherche en Informatique Fondamentale (UMR 8243)

and

Université Paris Cité

Correlated Pseudorandomness in Secure Computation

Geoffroy Couteau

Habilitation à Diriger des Recherches

Defended on March 20, 2024 in front of a jury composed of

Michel Abdalla	Directeur de recherche,	CNRS, ENS Paris	*Reviewer*
Benny Applebaum	Professor,	Tel-Aviv University	*Reviewer*
Ivan Damgård	Professor,	Aarhus University	*Examiner*
Carmit Hazay	Professor,	Bar-Ilan University	*Reviewer*
Sophie Laplante	Professor,	Université Paris-Cité	*Examiner*

Correlated Pseudorandomness in Secure Computation

Geoffroy COUTEAU

Acknowledgements

First, I would like to thank Michel Abdalla, Benny Applebaum, and Carmit Hazay for agreeing to review this HDR manuscript. Succinctness has never been my strong suit —the manuscript spans almost 130 pages when I was initially aiming for "70, 80 pages at most"— and I often struggle with striking the right balance between an appropriate level of formalism and readability. I am grateful that they took the time to read and digest it despite its inaccuracies and its ponderousness, and for all their comments on the write-up. Many thanks also to Michel Abdalla, Benny Applebaum, Ivan Damgård, Carmit Hazay, and Sophie Laplante for accepting to be on the jury of my defense.

My journey through cryptography has been made incredibly more enjoyable with the help and support of many people. Special thanks go to David Pointcheval, who patiently guided my first steps through research and kept providing support and advice over the years, to Yuval Ishai, who became my second mentor (and a friend), and who made me learn so much more than I could have imagined (it is amazing how fast you can learn when the deadline is in less than a week, not a single line of the paper has been written, and *someone* is coming up with brand new ideas and a new section to write at 5 am), and to Chris Brzuska for showing me that close friendship and research projects could be an awesome mix.

I also want to thank my colleagues at IRIF, and in particular those of the Algorithm and Complexity group, for making coming to work so enjoyable: Simon Apers, Michel De Rougemont, Pierre Fraigniaud, Iordanis Kerenidis, Amos Korman, Sophie Laplante, Frédéric Magniez, Claire Mathieu, Michele Orrù, Sylvain Perifel, Adi Rosén, Miklos Santha, David Saulpic, and Adrian Vladu. Special thanks to Sophie Laplante, Frédéric Magniez, and Adi Rosén, who have always been so nice and friendly, and who helped me and guided me on countless occasions. Many thanks also to Adrian Vladu, my office mate, for bearing my loud meetings throughout the years without ever protesting. Huge thanks also to the administrative and technical staff of IRIF for their everyday support: Omur Avci, Juliette Calvi, Natalia Hacquart, Houy Kuoy, Maximilien Leselliers, Jemuel Samtchar, and Marie-Laure Susairaj.

The work presented in this thesis would not have been possible without the help of my collaborators. In particular, I am extremely grateful for the chance I was given to work with amazing students (either supervised or hosted for long visits) and postdocs: Balthazar Bauer, Dung Bui, Eliana Carozza, Clément Ducros, Christoph Egger, Dominik Hartmann, Alexander Koch, Ulysse Léchine, Sven Maier, Pierre Meyer, Michael Reichle, Elahe Sadeghi, Sacha Servan-Schreiber, and Maryam Zarezadeh. I also want to acknowledge some of my closest collaborators: Thomas Agrikola, Chris Brzuska, Elette Boyle, Alain Couvreur, Pooya Farshim, Niv Gilboa, Dahmun Goudarzi (notice how I've put your name instead of mine, Dahmun), Dennis Hofheinz, Yuval Ishai, Antoine Joux, Shuichi Katsumata, Michael Klooß, Lisa Kohl, Mohammad Mahmoody, Alain Passelègue, David Pointcheval, Thomas Peters, Nicolas Resch, Peter Rindal, Yann Rotella, and Peter Scholl.

Finally, I would like to thank my wife, Camille, who makes me the happiest and luckiest human being and who has always been incredibly supportive even when I was spending night after night trying to complete papers by the deadline, and our son Émile, who agrees to let me go to work because the "motos" at the daycare are much more fun than I am anyway.

Abstract

The focus of this habilitation thesis is on secure computation, an area of cryptography that lets multiple parties distributively compute a function on their private data. After providing a high-level introduction to my work in cryptography, the manuscript provides a gentle introduction to secret-sharing-based secure computation, which is aimed at a general audience. Then, the last chapter covers some of my contributions to secure computation through the introduction and construction of pseudorandom correlation generators (PCG), a cryptographic primitive that enables considerable efficiency improvements for a wide variety of secure computation protocols. I provide a step-by-step introduction to the notion of PCG and its security properties, outline the challenges in building them, and present a general framework for constructing PCGs. The chapter also contains extensive efficiency considerations and covers various optimizations, as well as extensions and generalizations of the notion of PCGs. Altogether, this provides a unified introduction to the work on pseudorandom correlation generators developed in my work over the past five years, aimed at a broad cryptography audience.

Contents

Chapter 1

Introduction

The focus of this manuscript is on secure computation, an area of cryptography that is concerned with the protection of private data when they are used within distributed computations. The intent of the manuscript is not to provide an exhaustive introduction to cryptography (indeed, it does not even provide an exhaustive introduction to secure computation). In particular, I will not cover here the historical background that led cryptography to be the vibrant and flourishing area that it is now. There are countless amazing monographs on the subject, and I would not have much to add to them. I will also take the liberty to assume some familiarity of the reader with basic concepts and notations from cryptography. However, I will do my best to introduce all notations that could be unclear at first.

This was for the early disclaimers. At this stage, dear reader, you might be wondering what you should expect to get from this manuscript. So let's try to clarify this upfront, shall we?

1.1 What you Might Get from this Manuscript

The answer, of course, really depends on who you *are.* This manuscript is written as part of my application to the *Habilitation à Diriger des Recherches* (HDR), a French academic procedure whose successful completion grants the right to supervise PhD students. I hope, however, to also later use this writeup as a basis for a survey providing an introduction to pseudorandom correlations and their use in secure computation and zero-knowledge proofs, since this has been an active area in the past half-decade, and this could facilitate navigating through the field. In its current shape, this manuscript is somewhat of an introduction to secure computation (with a focus on secret-sharing-based secure computation) followed by a tutorial on silent secure computation, with a focus on their design and use within my work (so, for example, while there has been a lot of exciting work on zero-knowledge from pseudorandom correlation generators and on silent secure computation over the ring $\mathbb{Z}_{2^\kappa}$, you won't read anything about that here). It is therefore somewhat of a hybrid between an overview of my research activity, and a survey of the subfield of secure computation to which I contributed the most: it only covers the portion of my research that I deemed useful (hopefully) to synthesize in a unified and organized survey, and does not go much beyond what appeared in my work (except when it comes to covering general background on secure computation).

Chapter 2 provides an accessible introduction to secure computation, with a focus on the

GMW protocol, oblivious transfers, and secure computation in the preprocessing model. It is by no means comprehensive, but I tried to make it self-sufficient: it does not assume any extensive background in cryptography. The chapter aims to convey an intuition about where secure computation is today, and what challenges it faces – with a focus, of course, on the challenges *I* try to tackle. To keep the flow of the discourse fluid, I chose most of the time to introduce notions as I need them (with an informal definition when the discussion is informal, and a formal definition when I'm making precise theorem statements) rather than gathering them all in a preliminary section. I hope that this makes the task of reading the chapter slightly more like reading a story, and slightly less like deciphering a dry textbook. To assist the reader in finding where a notion was first introduced in the writeup, and assuming it will be mostly read on an electronic device, I used the knowledge package – see Section 1.4 "How to Read this Manuscript" for details. The writeup is also filled extensively with rough numbers, orders of magnitudes, and back-of-the-envelope calculations: the purpose is to convey a better intuition regarding what it means to say that a protocol is *slow* or *fast*, or that it requires *a lot* of communication or computation.

Chapter 3 covers the young field of silent secure computation, a branch of secure computation that emerged around 2018 in my work (and that of my coauthors). As is often the case in research, isolated algorithms, theorems, and protocols appeared first, and their unifying features became apparent over time. Hence, irrespective of how the results appeared chronologically in my publications, the chapter provides a step-by-step overview of a general recipe to construct pseudorandom correlation generators (from coding theoretic assumptions) before delving into the specific challenges of more concrete flavors of the notion and of its applications to secure computation.

So, to answer the question asked in the title of the Section: if you are a reviewer of this manuscript or an expert on secure computation, Chapter 2 will cover stuff which you are already very familiar with – though if you feel a bit rusty on the edges you might enjoy skimming through some of the explanations or back-of-the-envelope calculations. If you are a student or a young cryptographer discovering the field of secure computation, the chapter might help you get a better intuitive grasp of the field and its challenges, though beware that it only scratches at a restricted area of its surface. As for Chapter 3, if you did not work on (or with) pseudorandom correlation generators in the past, or if you lack a bird-eye view of how they work, the chapter should hopefully provide a good introduction to the subject. Here again, the focus is on conveying an intuition about how they emerged, how efficient they can get, what are the rules of thumb to estimate their security guarantees, and what are (some of) the remaining challenges. It only covers a small fraction of the subject: the field is evolving rapidly, and its applications have branched out into multiple directions, far more than I could cover in a manuscript of this length. But as a *starting point* for getting into the area, I hope that this chapter can be a reasonable introduction!

1.2 What you Won't Get from this Manuscript

This manuscript is intended to showcase some of my research contributions. However, it does not cover all of my research work by any means. It is restricted to my work on silent secure computation – and even there, does not cover all of my publications in the area. So, for the sake of completeness, in Section 1.3, I will briefly overview the *other* things I've been doing which did not make it to this manuscript. A warning is in order: in the context of the overview in the next Section, and for the sake of keeping the discussion short, I do not provide definitions of most of the cryptographic primitives I mention – the section is intended to be mostly accessible to the knowledgeable reader (and since its purpose is only providing a broad overview of what I've worked on, it can probably be skipped by most readers). This is the section of the manuscript that does not attempt to provide accessible coverage for the non-expert reader, and this reader might want to skip it.

Another thing the reader should not expect from this manuscript is fully detailed formal proofs. I focus on conveying the intuition behind the constructions, and usually only provide a brief overview of how the security analysis goes on. I tried to consistently provide pointers to where the interested reader can find the detailed proof or sometimes left them as an exercise (mostly for students or readers new to the area who would like to try their hands at how security analysis goes for these constructions).

1.3 What I Also Worked On

Below, I cover the other directions which I have been investigating. For each direction, I start with a brief introduction of the subject, outlining what was known and what challenges I focused on, and I give a list of my contributions with one or two sentences describing their content. The full list of my publications can be found at the end of this chapter.

1.3.1 Sublinear secure computation

Securely computing a function is the main subject of this manuscript. However, my focus is on improving *practical* secure computation protocols – that's why you will see many concrete numbers in the writeup. It is however also of fundamental interest to understand some more theoretical aspects of secure computations. One specific aspect I am particularly interested in is understanding the communication complexity of secure computation.

Concretely, if we want to compute a function represented by a boolean circuit of size s, we know how to do it generically with an amount of communication that scales linearly with s. Can we do better? If the parties only want to invest a polynomial amount of resources, and if the function does not belong to some restricted function class, the answer is far from obvious. It was even a long-standing open problem until the breakthrough work of Gentry, which introduced the first fully homomorphic encryption scheme (FHE) [Gen09]. This led to secure computation protocols with communication independent of the size of the function (proportional only to its input size and its output size), under (a circular security variant of) the LWE assumption. Almost three decades after the problem was first raised, this work finally broke through what is commonly referred to as the "circuit-size barrier".

However, this is not the end of the story: as of today, we only know of constructions of FHE from variants of the LWE assumption, or assuming the existence of indistinguishability obfuscation [CLTV15]. This leaves open an important question: under which cryptographic assumptions is it possible to securely compute a circuit with a sublinear amount of communication? Another good motivation for studying alternative roads to sublinear secure computation is the inefficiency of FHE-based constructions. Modern secure computation protocols, as we will see in this manuscript, are typically described in a *preprocessing model* where the bulk of the computation is pushed to an online phase, and the actual secure computation requires much fewer resources. FHE-based protocols do not fit in this paradigm, which begs the question: is it possible to securely compute a function in the preprocessing model using a sublinear amount of computation?

This question has motivated several exciting results recently, including but not limited to [BGI16a; OSY21; RS21; DIJL23]. Part of my work in the past few years has been devoted to this question, with a focus on secure computation protocols that go *slightly* beyond the circuit-size barrier (typically, communicating $O(s/\log\log s)$ bits for circuits of size s) for a large but restricted family of layered circuits (*i.e.* divided into layers such that every edge of the circuit connects adjacent layers), obtaining a variety of results:

- In [Cou19], I gave an information-theoretic secure computation protocol for layered circuits in the correlated-randomness model with $O(s/\log\log s)$ communication and a polynomial amount of correlated randomness (the protocol also extends to other settings, such as arithmetic circuits)

- In [CM21], together with my PhD student Pierre Meyer, we gave a secure computation protocol for layered circuits in the preprocessing model with $O(s/\log\log s)$ communication, assuming the superpolynomial hardness of the learning parity with noise assumption.

- In [BCM22], with Elette Boyle (co-supervisor of Pierre Meyer) and Pierre Meyer, we showed a new approach to sublinear secure computation that uses a strong form of private information retrieval (PIR), which does not imply the primitives used in previous works on the area (such as FHE and homomorphic secret sharing) and which implied, in particular, a construction of sublinear secure computation for layered circuit with $O(s/\log\log s)$ communication under the quadratic residuosity and learning parity with noise assumption (the paper also describes a new polylogarithmic private information retrieval protocol from the computational Diffie-Hellman assumption).

- In [BCM23], with Elette Boyle and Pierre Meyer, we showed how combining our strong form of PIR with homomorphic secret sharing yields sublinear secure computation protocols for more than two parties, breaking the 2-party barrier in sublinear secure computation without FHE. We provide instantiations for $N \in \{3, 4, 5\}$ parties.

- Eventually, in [CMPR23], with Pierre Meyer, Alain Passelègue, and Mahshid Riahinia, we gave the first sublinear one-sided statistically secure two-party computation protocol without FHE. Our work also obtains new constructions of constrained pseudorandom functions.

This line of work is an exciting and ongoing research area, and several follow-ups are currently in preparation. With the exception of [Cou19], all the works above have involved Pierre Meyer and were conducted in the context of his Ph.D. The techniques involved are closely related to, and build upon, the work on pseudorandom correlation generators which will be the focus of this manuscript. However, since these techniques and the results above are already the focus of the Ph.D. manuscript of Pierre Meyer, who will defend his Ph.D. in September 2023, I decided against discussing them any further in this manuscript. The interested reader is referred to Pierre's Ph.D. manuscript for an excellent introduction to the subject and a detailed coverage of this active research area.

1.3.2 Practical secure computation

Most of my work on secure computation deals with either (1) developing techniques to speed up generic secure computation in the silent preprocessing model (which is the focus of this manuscript), and (2) investigating the existence of secure computation protocols with sublinear communication (which I covered in Section 1.3.1). However, I also enjoy the challenge of designing improved (practical) secure computation protocols for functionalities that are useful in real-world applications. In particular, I have investigated the following functionalities:

- In [Cou18], I designed improved two-party protocol for computing the equality test (given inputs x, y, outputs shares of 1 if $x = y$, and shares of 0 otherwise) and greater-than (outputs shares of 1 iff $x \geq y$) functionalities.
- In [CZ22], we designed improved post-quantum non-interactive secure two-party protocols to compute the inner product functionality (on inputs vectors $\vec{x}, \vec{y}$, outputs $\vec{x}^\intercal \cdot \vec{y}$).
- In [BC23], together with my PhD student Dung Bui, we designed improved protocols for private set intersection (on inputs sets X, Y, outputs $X \cap Y$).

The last two protocols above borrow ideas and techniques from my work on pseudorandom correlation generators. I also worked on secure computation protocols for switching between encryption schemes (transforming an encryption $E(m)$ into an encryption $E'(m)$ under a new scheme, without leaking m) in [CPP16], and on protocols for partial fair exchange (exchanging two inputs while achieving a weaker flavor of fairness) in [CRR21b].

1.3.3 Foundations of non-interactive zero-knowledge

Zero-knowledge proofs [GMR89] allow a prover to demonstrate the truth of a statement to a verifier while concealing all information beyond this truth. Non-interactive zero-knowledge proofs [BFM88] (NIZKs) are a specific type of zero-knowledge proof where the interaction consists of a single message from the prover to the verifier. NIZKs are an extremely useful cryptographic primitive, with countless applications. One question I'm especially interested in concerns the existence of NIZKs, and more precisely, under which assumptions can NIZKs be shown to exist, and using which setup (since NIZKs for all of NP are known not to exist if no setup is assumed).

Concretely, in the common reference string model (which is, in a sense, the weakest possible), NIZKs are known to exist from assumptions in pairing groups [GOS06; GS08], factorization assumptions [BFM88; FLS90], indistinguishability obfuscation [SW14], and LWE [PS19]. Two big question marks are however (1) do NIZKs exist from discrete-logarithm-style assumptions in groups *without* pairings, and (2) do NIZKs require public-key cryptography? Another interesting thread of research, which I view as a more "bottom-up" approach to NIZKs, investigates relaxed notions of NIZKs, for example with a stronger form of setup. This includes the study of preprocessing NIZKs (where both the prover and the verifier receive a key in the setup) or of designated-verifier NIZKs (DVNIZKs), where the setup hands a special proof verification key to the verifier.

My work so far has pursued both the top-down approach (trying to base full-fledged NIZKs in the common reference string model on the weakest possible assumptions) and the bottom-up approach. In particular:

- In my very first paper [BCPW15], we introduced implicit zero-knowledge, an intermediate notion between interactive and non-interactive zero-knowledge which can be realized from plain DDH and has application to round-efficient secure computation. This work is what motivated me to investigate the area in the first place.
- In [CC18], we introduced the first designated-verifier NIZKs with strong soundness notion (reusable and statistical soundness) from the DCR assumption. This was the first efficient construction of designated-verifier NIZK with reusable soundness (which states that soundness is not compromised when the verifier checks multiple proofs) in the standard model without pairings.
- In [CH19], we showed that designated-verifier NIZKs with reusable soundness exist from the plain CDH assumption (without pairings), separating for the first time the assumptions known to imply (reusable) DVNIZKs from those known to imply NIZKs.
- In [BCGI18; BCG+19b; BCG+20a; BCG+22], we obtained various constructions of (reusable) preprocessing NIZKs from variants of the syndrome decoding assumption. These works construct pseudorandom correlation generators, which are the focus of this manuscript, and the construction of preprocessing NIZKs follows from a close connection between pseudorandom correlation generators and preprocessing NIZKs (which I will not cover in this manuscript).
- In [CKU20], we obtained full-fledged NIZKs in pairing-free groups under a new, strong but falsifiable variant of the discrete logarithm assumption, as well as the first candidate NIZK from assumptions not known to imply public key encryption.
- In [CH20], we described a new framework to build NIZKs in pairing groups, obtaining in many settings shorter proofs, but also yielding statistical witness indistinguishable proofs in the plain model (for a family of witness-samplable languages), a variant of NIZKs which has proven notably hard to achieve.
- In [CLPØ21], we extended the framework of [CH20], showing in particular short pairing-based NIZKs for a large class of algebraic languages from a new type of gap assumptions.
- In [CKSU21], we described a new compiler to build statistical non-interactive witness indistinguishable proofs for NP in the plain model, obtaining constructions either from standard pairing-based assumption (by plugging [CH20] in our compiler) or from a strong but falsifiable variant of the discrete log assumption in pairing-free groups (by plugging [CKU20] in the compiler).
- Eventually, in [CJJQ23], we showed that infinitely-often NIZKs for NP could be based on the subexponential hardness of the CDH assumption in pairing-free groups, or from the CDH+LPN assumptions. Our techniques also enable us to mitigate strongly the limitations of the infinitely-often flavor of security.

Zero-knowledge proofs were the focus of my PhD (though only the first two works of the list above are covered there). I decided against including an extended coverage of the subject in this manuscript, having already included an accessible introduction of the topic in the form of a 50-page chapter in my Ph.D. thesis. The reader interested in the area can check it out: it provides relatively wide coverage of the foundations but does not cover so much of more recent results (and it has a focus on the areas that I investigated during my PhD).

1.3.4 Efficient zero-knowledge proofs and signatures

Even though my original motivation lies mostly within foundational aspects of zero-knowledge proofs, my work has also led to the development of new concretely efficient zero-knowledge proofs. The previous section already covers [BCPW15; CC18; CH20; CLPØ21] and the works based on pseudorandom correlation generators [BCGI18; BCG+19b; BCG+20a; BCG+22], which all had an eye (or two) in the direction of concretely efficient proofs. But I've also investigated extensively the design of interactive zero-knowledge proofs (which can be made non-interactive using the Fiat-Shamir heuristic) for statements of interest in the real world. Concretely, I have had two research threads in this direction, one older and one more recent:

- I study the construction of range proofs, which are zero-knowledge proofs tailored to proving statements of the form "this committed integer belongs to the public range $[a, b]$". Such proofs enjoy numerous applications, from anonymous credentials [Cha90] to e-voting [Gro05], e-cash and anonymous transactions [CHL05].

- More recently, I developed a strong interest in the design of zero-knowledge proofs of knowledge of a preimage for the syndrome decoding problem. The motivation behind this goal is that such zero-knowledge proofs yield new (plausibly) post-quantum signature schemes whose security reduces to code-based cryptographic assumption, which is of particular interest in the context of recent NIST competitions for standardizing post-quantum cryptography.

My work in these areas resulted in the following publications:

- In [CPP17], we showed how to replace the strong RSA assumption (underlying zero-knowledge proofs for integer commitment schemes) with the standard RSA assumption, and described a new range proof based on integer commitments.

- In [CKLR21], we developed an approach to building bounded integer commitment schemes without relying on hidden order groups. This enabled the construction of new range proofs from integer commitment over standard pairing-free groups, with extremely short proof and high concrete efficiency.

- In [CGKR22], we refined our previous approach and developed a heavily optimized family of new range-proof schemes with various security-efficiency-functionality tradeoffs, with proof of concept implementations, and investigated their use in several applications.

- In [CCJ23], with my PhD students Eliana Carozza and Antoine Joux, we introduced a new highly competitive signature scheme from the regular syndrome decoding assumption, a variant of the syndrome decoding assumption that will also show up on several occasions in this manuscript.

- On a slightly different topic, I also studied the use of my DVNIZK framework from [CC18] in the construction of more efficient anonymous credential systems, which resulted in another publication [CR19].

1.3.5 Fine-grained cryptography

So far, I have exclusively covered works related to either secure computation or zero-knowledge proofs, which constitute my main topic. However, I also have a deep interest in the foundations of cryptography. In particular, I am interested in the limitations of black-box techniques, connections between cryptography and learning theory, and (this will be the focus of this section) fine-grained cryptography.

Traditional cryptography requires the hardness of cryptographic primitives to hold against arbitrary polynomial-time adversaries. In contrast, *fine-grained* cryptography aims to study the feasibility of cryptographic primitives when the adversarial power is restricted, for example, to some fixed polynomial bound. A core motivation underlying the research on fine-grained cryptography is the hope that by relaxing the security to hold against less powerful adversaries, it might be possible to base the existence of fine-grained primitives on assumptions that are weaker than those known to imply their full-fledged counterparts.[1]

While the study of fine-grained cryptography can be traced back to the seminal paper of Merkle [Mer74; Mer78], who constructed a key exchange from idealized hash functions with security against subquadratic-time adversaries, this primitive has recently spurred a renewed interest, leading to a collection of constructions [BGI08; BHK+11; DVV16; BRSV17; BRSV18; CG18; LLW19; EWT21; DH21; WP22] and lower bounds [BM09; BC22] for fine-grained cryptographic primitives. While some results deal with adversaries of restricted (e.g. logarithmic) depth, I am more specifically interested in fine-grained protocols that only bound the *runtime* of the adversary, since as computational power grows, polynomial runtime gaps between honest parties and adversaries become more and more meaningful. In this area, I obtained the following results:

- In [BC22], we investigated the classical question of whether one-way functions can be based on (strong forms of) average-case hardness, but this time by relaxing the goal to building fine-grained one-way functions. We obtained both negative results (proving black-box separations ruling out constructions of fine-grained OWFs even from non-amortizable, exponentially average-case hard languages in $\mathsf{NP} \cap \mathsf{coNP}$) and (limited) positive results, showing that very strong forms of average-case hardness imply fine-grained OWFs with a quadratic gap. This work was invited to the Journal of Cryptology.

- In [ACM22], we studied the intriguing possibility of anonymously transmitting information while communicating solely over authenticated channels, without relying on trusted or non-colluding parties, probing the necessity of trust assumptions for the important goal of anonymous whistleblowing. We proved a general impossibility result in the standard setting, but (perhaps more interestingly) also achieved a fine-grained protocol for anonymous communication over public channels, where deanonymizing the sender requires quadratic work (over that of the honest participants), assuming idealized obfuscation (which yields a plausible heuristic construction when relying on indistinguishability obfuscation).

- In [ACMS23], we proved that fine-grained 3-party non-interactive key exchange (NIKE) with $n^{1.5}$ hardness gap exists in the random oracle model, and that fine-grained 4-party

[1] In fact, since fine-grained cryptography only deals with hardness within P, one could theoretically hope to achieve unconditional constructions. But we are not there yet, except when targeting security against extremely weak adversaries within the class AC^0 [DVV16].

NIKE (with quadratic hardness gap) exists in pairing-free groups (in the generic group model). The result also extends to a 6-party NIKE with quadratic hardness over bilinear groups. We also proved the impossibility of achieving 3-party NIKE in the generic group model with superquadratic hardness, essentially matching our upper bound. In contrast, NIKE with more than 3 parties is only known from indistinguishability obfuscation or multilinear map in the standard (non-fine-grained) setting.

1.3.6 Other topics: learning theory, black-box separations, and more

For the sake of completeness, let me briefly overview my works that did not fit in the previous sections, in no particular order:

- I have also worked on connections between cryptography and learning theory, by studying the existence of (weak) pseudorandom functions in low-complexity. This line of work was initially motivated by connections to pseudorandom correlation generators and led to a result [BCG+20a] which is covered later on in this manuscript. We investigated further this area in a follow-up work [BCG+21b].

- I also enjoy studying black-box separations (I covered some in the previous section), and worked in particular on black-box limitations for constructing strong one-way functions from weak one-way functions in [BCKR21], and introduced the intriguing concept of black-box uselessness in [CFM21] (informally, a primitive A is black-box useless for constructing a primitive B if, for every black-box construction of B from A, C, where C is any other primitive, there must exist a black-box construction of B from C in the first place) together with a preliminary investigation of the black-box uselessness of one-way functions for building key agreement (among other primitives).

- I have a couple of other works on foundational questions in secure computation. In [ACI+20], we studied the notion of pseudorandom encoding (a primitive that allows embedding any distribution into the uniform distribution), deriving a particular an interesting connection between covert secure computation and fully adaptive secure computation, as well as to various other notions related to steganography and separations between Yao and HILL entropy. In [CR22], we asked how many parties need to be able to toss coins in information-theoretic secure computation, proving, in particular, a surprising result: it is necessary and sufficient that t parties can toss coins to t-privately compute any deterministic functionality (hence, in particular, the adversary could corrupt all parties that can toss coins).

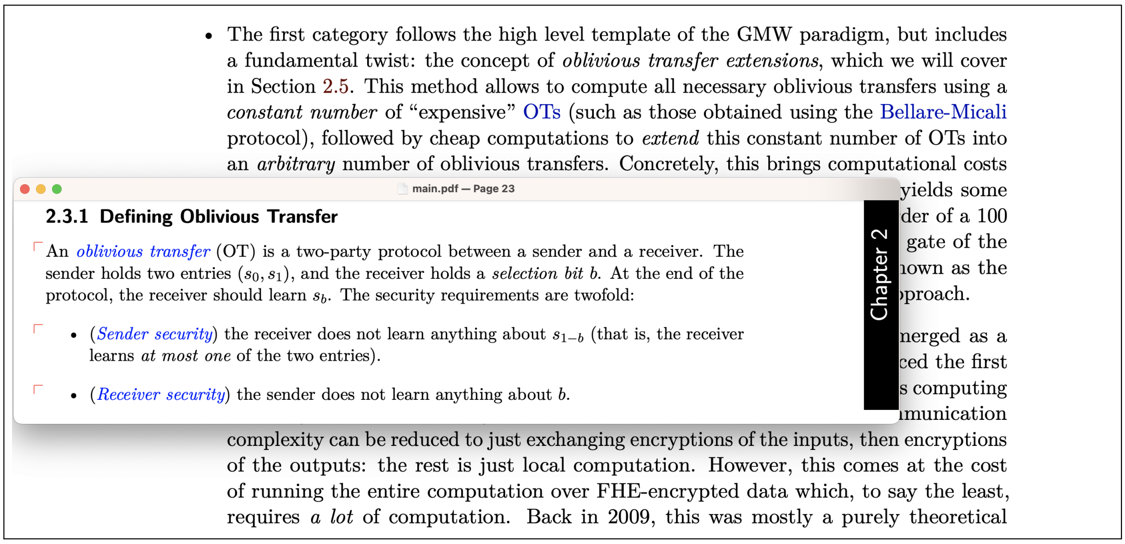
- Eventually, I investigated a couple of other cryptographic primitives, such as indistinguishability obfuscation in [ACH20] (showing how to base e.g. FHE on polynomially secure iO together with extremely lossy functions), and constrained pseudorandom functions in [CMPR23].

At the end of this introduction, I include a full list of my publications for reference.

1.4 How to Read this Manuscript

This manuscript introduces, covers, and manipulates a very large number of notions: secure computation is a vast field that comes in many shapes, with multiple variants and flavors for each of the security notions, and builds upon a large pool of cryptographic primitives. On the negative side, this can make the task of reading the manuscript quite cumbersome for anyone who lacks familiarity with the field: if read linearly by someone who never saw these notions, and even though most of them are relatively simple and intuitive in isolation, it might overflow the short-term memory buffer of the reader.

On the positive side, however, it gave me the necessary motivation to finally try my hands at the wonderful knowledge package created by our very own Thomas Colcombet. The *knowledge package* helps the reader keep track of notions and definitions, by providing a clean way to link a notion to its definition in the manuscript. If you are reading this on an electronic device, you can try it out: simply hover your cursor on any occurrence of the terms knowledge package to get a glimpse of the paragraph where they are introduced, or click on it to access the corresponding position in the document. I tried to use this consistently for all field-specific notions throughout this document, so in case of doubt about what a notion means, just click on the obscure term! In case you want to try it in your writeups, I followed the nice tutorial of Rémi Morvan (hover your cursor on tutorial to see the link and click it to open it in your browser). If you are afraid that clicking on a notion will send you to a random page in the document, and that the back-and-forth navigation between where the notion was defined and where you were will be annoying, you might consider using a PDF reader that enables opening small windows from pointers to parts of the document. For example, I use Skim, and using MAJ-ctrl-click (MAJ-cmd-click on a Mac) on a link opens a small window to the relevant part of the document (see the image below, where I put my cursor on OT and MAJ-ctrl-click it).



- The first category follows the high level template of the GMW paradigm, but includes a fundamental twist: the concept of *oblivious transfer extensions*, which we will cover in Section 2.5. This method allows to compute all necessary oblivious transfers using a *constant number* of “expensive” OTs (such as those obtained using the Bellare-Micali protocol), followed by cheap computations to *extend* this constant number of OTs into an *arbitrary* number of oblivious transfers. Concretely, this brings computational costs

2.3.1 Defining Oblivious Transfer

An *oblivious transfer* (OT) is a two-party protocol between a sender and a receiver. The sender holds two entries (s_0, s_1), and the receiver holds a *selection bit* b. At the end of the protocol, the receiver should learn s_b. The security requirements are twofold:

- (*Sender security*) the receiver does not learn anything about s_{1-b} (that is, the receiver learns *at most one* of the two entries).
- (*Receiver security*) the sender does not learn anything about b.

yields some
der of a 100
gate of the
nown as the
pproach.

nerged as a
ced the first
s computing
nmunication
complexity can be reduced to just exchanging encryptions of the inputs, then encryptions of the outputs: the rest is just local computation. However, this comes at the cost of running the entire computation over FHE-encrypted data which, to say the least, requires *a lot* of computation. Back in 2009, this was mostly a purely theoretical

1.5 Notations

In this writeup, λ denotes a security parameter, and write $n(\lambda) \in \text{poly}(\lambda)$ to indicate that $n(\lambda)$ is polynomial in λ (we also sometimes write this as $n(\lambda) = \text{poly}(\lambda)$). We write $\mu(\lambda) \leq \mathsf{negl}(\lambda)$ to indicate that μ goes to zero faster than $1/\lambda^c$ for any constant c, and say that μ is a *negligible* function. Given a distribution $\mathcal{D}$, we write $x \overset{\$}{\leftarrow} \mathcal{D}$ to indicate that x is sampled from $\mathcal{D}$. Given a set S, we also write $x \overset{\$}{\leftarrow} S$ as a shorthand to indicate that x is sampled from the uniform distribution over S. For an integer $n \in \mathbb{N}$, we write $[n]$ to denote the set of integers

$\{1, \cdots, n\}$. We write vectors as $\vec{x}$, but often identify vectors over $\mathbb{F}_2$ as bitstrings, and write them x, without the arrow. Given two vectors $\vec{x}$, $\vec{y}$, $\vec{x} \oplus \vec{y}$ denotes their component-wise xor (when the vectors are over $\mathbb{F}_2$), and $\vec{x} \odot \vec{y}$ denotes their component-wise product. By default, we view vectors as columns.

Adversaries (modeled as polynomial-size boolean circuits) are generally denoted $\mathcal{A}$. We use the following template to write probabilities: Pr[event | experiment]. For example, $\Pr[\mathcal{A}(x) = 0 \;\; x \xleftarrow{\$} \{0,1\}^\lambda]$ denotes the probability that the adversary $\mathcal{A}$ outputs 0 when given an input x sampled uniformly from $\{0,1\}^\lambda$. When the experiment can be concisely described, we sometimes use the more compact notation $\Pr_{\text{experiment}}[\text{event}]$ (e.g. $\Pr_{x \xleftarrow{\$} \{0,1\}^\lambda}[\mathcal{A}(x) = 0]$), or even Pr[event] when the experiment is clear from the context. For distributions, we use the template $\mathcal{D} = \{\text{sample} \mid \text{experiment}\}$. For example, $\{(a, b, a \cdot b) \mid (a, b) \xleftarrow{\$} \mathbb{F}^2\}$ denotes the distribution which samples two random elements (a, b) from a finite field $\mathbb{F}$ and outputs $(a, b, a \cdot b)$.

1.6 Personal Publications

The bibliography below contains all my publications so far, ranked alphabetically. All papers listed have been published at international conferences, and all have an open-access full version (except two, whose full versions are a work in progress and will appear soon).

[ACH20] Thomas Agrikola, Geoffroy Couteau, and Dennis Hofheinz. "The Usefulness of Sparsifiable Inputs: How to Avoid Subexponential iO". In: *PKC 2020, Part I.* Ed. by Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas. Vol. 12110. LNCS. Springer, Heidelberg, May 2020, pp. 187–219. DOI: 10.1007/978-3-030-45374-9_7 (cit. on p. 9).

[ACI+20] Thomas Agrikola, Geoffroy Couteau, Yuval Ishai, Stanislaw Jarecki, and Amit Sahai. "On Pseudorandom Encodings". In: *TCC 2020, Part III.* Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12552. LNCS. Springer, Heidelberg, Nov. 2020, pp. 639–669. DOI: 10.1007/978-3-030-64381-2_23 (cit. on p. 9).

[ACM22] Thomas Agrikola, Geoffroy Couteau, and Sven Maier. "Anonymous Whistleblowing over Authenticated Channels". In: *TCC 2022, Part II.* Ed. by Eike Kiltz and Vinod Vaikuntanathan. Vol. 13748. LNCS. Springer, Heidelberg, Nov. 2022, pp. 685–714. DOI: 10.1007/978-3-031-22365-5_24 (cit. on p. 8).

[ACMS23] Abtin Afshar, Geoffroy Couteau, Mohammad Mahmoody, and Elahe Sadeghi. "Fine-Grained Non-interactive Key-Exchange: Constructions and Lower Bounds". In: *EUROCRYPT 2023, Part I.* LNCS. Springer, Heidelberg, June 2023, pp. 55–85. DOI: 10.1007/978-3-031-30545-0_3 (cit. on p. 8).

[BC22] Chris Brzuska and Geoffroy Couteau. "On Building Fine-Grained One-Way Functions from Strong Average-Case Hardness". In: *EUROCRYPT 2022, Part II.* Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. LNCS. Springer, Heidelberg, May 2022, pp. 584–613. DOI: 10.1007/978-3-031-07085-3_20 (cit. on p. 8).

[BC23] Dung Bui and Geoffroy Couteau. “Improved Private Set Intersection for Sets with Small Entries”. In: *PKC 2023, Part II*. LNCS. Springer, Heidelberg, May 2023, pp. 190–220. DOI: 10.1007/978-3-031-31371-4_7 (cit. on p. 5).

[BCCD23] Maxime Bombar, Geoffroy Couteau, Alain Couvreur, and Clément Ducros. “Correlated Pseudorandomness from the Hardness of Quasi-Abelian Decoding”. In: *Annual International Cryptology Conference*. Springer. 2023 (cit. on pp. 53, 61, 77, 80, 91).

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. “Homomorphic Secret Sharing: Optimizations and Applications”. In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 2105–2122. DOI: 10.1145/3133956.3134107 (cit. on pp. 53, 61).

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. “Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation”. In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 291–308. DOI: 10.1145/3319535.3354255 (cit. on pp. 53, 61, 80, 83, 87).

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. “Efficient Pseudorandom Correlation Generators: Silent OT Extension and More”. In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 489–518. DOI: 10.1007/978-3-030-26954-8_16 (cit. on pp. 6, 7, 53, 58–61, 80, 105).

[BCG+20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. “Correlated Pseudorandom Functions from Variable-Density LPN”. In: *61st FOCS*. IEEE Computer Society Press, Nov. 2020, pp. 1069–1080. DOI: 10.1109/FOCS46700.2020.00103 (cit. on pp. 6, 7, 9, 53, 61, 77, 93, 96, 98, 99).

[BCG+20b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. “Efficient Pseudorandom Correlation Generators from Ring-LPN”. In: *CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. LNCS. Springer, Heidelberg, Aug. 2020, pp. 387–416. DOI: 10.1007/978-3-030-56880-1_14 (cit. on pp. 53, 61, 80, 90, 91, 105).

[BCG+21b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. “Low-Complexity Weak Pseudorandom Functions in AC0[MOD2]”. In: *CRYPTO 2021, Part IV*. Ed. by Tal Malkin and Chris Peikert. Vol. 12828. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 487–516. DOI: 10.1007/978-3-030-84259-8_17 (cit. on pp. 9, 53).

[BCG+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. “Correlated Pseudorandomness from Expand-Accumulate Codes”. In: *CRYPTO 2022, Part II*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13508. LNCS. Springer, Heidelberg, Aug. 2022, pp. 603–633. DOI: 10.1007/978-3-031-15979-4_21 (cit. on pp. 6, 7, 53, 61, 77, 80, 88, 99–102).

[BCG+23] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. “Oblivious Transfer with Constant Computational Overhead”. In: *EUROCRYPT 2023, Part I.* LNCS. Springer, Heidelberg, June 2023, pp. 271–302. DOI: 10.1007/978-3-031-30545-0_10.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. “Compressing Vector OLE”. In: *ACM CCS 2018.* Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM Press, Oct. 2018, pp. 896–912. DOI: 10.1145/3243734.3243868 (cit. on pp. 6, 7, 53, 59–61, 79–81, 83).

[BCKR21] Chris Brzuska, Geoffroy Couteau, Pihla Karanko, and Felix Rohrbach. “On Derandomizing Yao’s Weak-to-Strong OWF Construction”. In: *TCC 2021, Part II.* Ed. by Kobbi Nissim and Brent Waters. Vol. 13043. LNCS. Springer, Heidelberg, Nov. 2021, pp. 429–456. DOI: 10.1007/978-3-030-90453-1_15 (cit. on p. 9).

[BCM22] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. “Sublinear Secure Computation from New Assumptions”. In: *TCC 2022, Part II.* Ed. by Eike Kiltz and Vinod Vaikuntanathan. Vol. 13748. LNCS. Springer, Heidelberg, Nov. 2022, pp. 121–150. DOI: 10.1007/978-3-031-22365-5_5 (cit. on p. 4).

[BCM23] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. “Sublinear-Communication Secure Multiparty Computation Does Not Require FHE”. In: *EUROCRYPT 2023, Part II.* LNCS. Springer, Heidelberg, June 2023, pp. 159–189. DOI: 10.1007/978-3-031-30617-4_6 (cit. on pp. 4, 105).

[BCPW15] Fabrice Benhamouda, Geoffroy Couteau, David Pointcheval, and Hoeteck Wee. “Implicit Zero-Knowledge Arguments and Applications to the Malicious Setting”. In: *CRYPTO 2015, Part II.* Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. LNCS. Springer, Heidelberg, Aug. 2015, pp. 107–129. DOI: 10.1007/978-3-662-48000-7_6 (cit. on pp. 6, 7).

[CC18] Pyrros Chaidos and Geoffroy Couteau. “Efficient Designated-Verifier Non-interactive Zero-Knowledge Proofs of Knowledge”. In: *EUROCRYPT 2018, Part III.* Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, Apr. 2018, pp. 193–221. DOI: 10.1007/978-3-319-78372-7_7 (cit. on pp. 6, 7).

[CCJ23] Eliana Carozza, Geoffroy Couteau, and Antoine Joux. “Short Signatures from Regular Syndrome Decoding in the Head”. In: LNCS. Springer, Heidelberg, June 2023, pp. 532–563. DOI: 10.1007/978-3-031-30589-4_19 (cit. on p. 7).

[CD23] Geoffroy Couteau and Clément Ducros. “Pseudorandom Correlation Functions from Variable-Density LPN, Revisited”. In: *PKC 2023, Part II.* LNCS. Springer, Heidelberg, May 2023, pp. 221–250. DOI: 10.1007/978-3-031-31371-4_8 (cit. on pp. 53, 77, 98, 99).

[CDM+18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. "On the Concrete Security of Goldreich's Pseudorandom Generator". In: *ASIACRYPT 2018, Part II*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. LNCS. Springer, Heidelberg, Dec. 2018, pp. 96–124. DOI: 10.1007/978-3-030-03329-3_4.

[CFM21] Geoffroy Couteau, Pooya Farshim, and Mohammad Mahmoody. "Black-Box Uselessness: Composing Separations in Cryptography". In: *ITCS 2021*. Ed. by James R. Lee. Vol. 185. LIPIcs, Jan. 2021, 47:1–47:20. DOI: 10.4230/LIPIcs.ITCS.2021.47 (cit. on p. 9).

[CGKR22] Geoffroy Couteau, Dahmun Goudarzi, Michael Klooß, and Michael Reichle. "Sharp: Short Relaxed Range Proofs". In: *ACM CCS 2022*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. ACM Press, Nov. 2022, pp. 609–622. DOI: 10.1145/3548606.3560628 (cit. on p. 7).

[CH19] Geoffroy Couteau and Dennis Hofheinz. "Designated-Verifier Pseudorandom Generators, and Their Applications". In: *EUROCRYPT 2019, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 562–592. DOI: 10.1007/978-3-030-17656-3_20 (cit. on p. 6).

[CH20] Geoffroy Couteau and Dominik Hartmann. "Shorter Non-interactive Zero-Knowledge Arguments and ZAPs for Algebraic Languages". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 768–798. DOI: 10.1007/978-3-030-56877-1_27 (cit. on pp. 6, 7).

[CJJQ23] Geoffroy Couteau, Abhishek Jain, Zhengzhong Jin, and Willy Quach. "A Note on Non-Interactive Zero-Knowledge from CDH". In: *Annual International Cryptology Conference*. Springer. 2023 (cit. on p. 6).

[CKLR21] Geoffroy Couteau, Michael Klooß, Huang Lin, and Michael Reichle. "Efficient Range Proofs with Transparent Setup from Bounded Integer Commitments". In: *EUROCRYPT 2021, Part III*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12698. LNCS. Springer, Heidelberg, Oct. 2021, pp. 247–277. DOI: 10.1007/978-3-030-77883-5_9 (cit. on p. 7).

[CKSU21] Geoffroy Couteau, Shuichi Katsumata, Elahe Sadeghi, and Bogdan Ursu. "Statistical ZAPs from Group-Based Assumptions". In: *TCC 2021, Part I*. Ed. by Kobbi Nissim and Brent Waters. Vol. 13042. LNCS. Springer, Heidelberg, Nov. 2021, pp. 466–498. DOI: 10.1007/978-3-030-90459-3_16 (cit. on p. 6).

[CKU20] Geoffroy Couteau, Shuichi Katsumata, and Bogdan Ursu. "Non-interactive Zero-Knowledge in Pairing-Free Groups from Weaker Assumptions". In: *EUROCRYPT 2020, Part III*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12107. LNCS. Springer, Heidelberg, May 2020, pp. 442–471. DOI: 10.1007/978-3-030-45727-3_15 (cit. on p. 6).

[CLPØ21] Geoffroy Couteau, Helger Lipmaa, Roberto Parisella, and Arne Tobias Ødegaard. "Efficient NIZKs for Algebraic Sets". In: *ASIACRYPT 2021, Part III*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13092. LNCS. Springer, Heidelberg, Dec. 2021, pp. 128–158. DOI: 10.1007/978-3-030-92078-4_5 (cit. on pp. 6, 7).

[CM21] Geoffroy Couteau and Pierre Meyer. “Breaking the Circuit Size Barrier for Secure Computation Under Quasi-Polynomial LPN”. In: *EUROCRYPT 2021, Part II*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. LNCS. Springer, Heidelberg, Oct. 2021, pp. 842–870. DOI: 10.1007/978-3-030-77886-6_29 (cit. on pp. 4, 53).

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. “Constrained Pseudorandom Functions from Homomorphic Secret Sharing”. In: *EUROCRYPT 2023, Part III*. LNCS. Springer, Heidelberg, June 2023, pp. 194–224. DOI: 10.1007/978-3-031-30620-4_7 (cit. on pp. 4, 9, 53).

[Cou18] Geoffroy Couteau. “New Protocols for Secure Equality Test and Comparison”. In: *ACNS 18*. Ed. by Bart Preneel and Frederik Vercauteren. Vol. 10892. LNCS. Springer, Heidelberg, July 2018, pp. 303–320. DOI: 10.1007/978-3-319-93387-0_16 (cit. on p. 5).

[Cou19] Geoffroy Couteau. “A Note on the Communication Complexity of Multiparty Computation in the Correlated Randomness Model”. In: *EUROCRYPT 2019, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 473–503. DOI: 10.1007/978-3-030-17656-3_17 (cit. on p. 4).

[CPP16] Geoffroy Couteau, Thomas Peters, and David Pointcheval. “Encryption Switching Protocols”. In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 308–338. DOI: 10.1007/978-3-662-53018-4_12 (cit. on p. 5).

[CPP17] Geoffroy Couteau, Thomas Peters, and David Pointcheval. “Removing the Strong RSA Assumption from Arguments over the Integers”. In: *EUROCRYPT 2017, Part II*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. LNCS. Springer, Heidelberg, Apr. 2017, pp. 321–350. DOI: 10.1007/978-3-319-56614-6_11 (cit. on p. 7).

[CR19] Geoffroy Couteau and Michael Reichle. “Non-interactive Keyed-Verification Anonymous Credentials”. In: *PKC 2019, Part I*. Ed. by Dongdai Lin and Kazue Sako. Vol. 11442. LNCS. Springer, Heidelberg, Apr. 2019, pp. 66–96. DOI: 10.1007/978-3-030-17253-4_3 (cit. on p. 7).

[CR22] Geoffroy Couteau and Adi Rosén. “Random Sources in Private Computation”. In: *ASIACRYPT 2022, Part I*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13791. LNCS. Springer, Heidelberg, Dec. 2022, pp. 443–473. DOI: 10.1007/978-3-031-22963-3_15 (cit. on p. 9).

[CRR21a] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. “Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes”. In: *CRYPTO 2021, Part III*. Ed. by Tal Malkin and Chris Peikert. Vol. 12827. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 502–534. DOI: 10.1007/978-3-030-84252-9_17 (cit. on pp. 53, 61, 77, 80).

[CRR21b] Geoffroy Couteau, A. W. Roscoe, and Peter Y. A. Ryan. “Partially-Fair Computation from Timed-Release Encryption and Oblivious Transfer”. In: *ACISP 21*. Ed. by Joonsang Baek and Sushmita Ruj. Vol. 13083. LNCS. Springer, Heidelberg, Dec. 2021, pp. 330–349. DOI: 10.1007/978-3-030-90567-5_17 (cit. on p. 5).

[CZ22] Geoffroy Couteau and Maryam Zarezadeh. “Non-interactive Secure Computation of Inner-Product from LPN and LWE”. In: *ASIACRYPT 2022, Part I*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13791. LNCS. Springer, Heidelberg, Dec. 2022, pp. 474–503. DOI: 10.1007/978-3-031-22963-3_16 (cit. on p. 5).

Chapter 2

Secure Computation: a Primer

Traditionally, the goal of cryptography has been to secure *communications*: Alice wants to send a message to Bob, but outside observers should not be able to read the message. In a certain way, this notion lacks granularity: either a participant learns the full message of Alice or none of it. Yet, in many realistic scenarios, it is desirable to have a more fine-grained control over which information is revealed. For example, imagine that Alice and Bob are interacting via a dating app, and wish to discover whether they are romantically interested in each other: it might be that Alice does not want to simply reveal to Bob that she is romantically interested in him. Rather, she could want to let Bob learn her interest *if and only if* Bob is also romantically interested in her. Secure communication does not provide a solution to this problem. One way to formalize this simple task is as follows: Alice has an input bit a and Bob has an input bit b, and both parties want to let the other learn $f(a, b) = a \wedge b$ (the bit equal to 1 in case of mutual romantic interest), *and nothing more.* In other words, our (potential) lovebirds want to exchange private information defined by the computation of a function on their joint private input. The goal of *secure multiparty computation*, often abbreviated MPC (for MultiParty Computation), is to let them do exactly that – and much more.

Dall-E's best attempt at drawing Alice and Bob securely computing a function

Contents

2.1 Secure Computation

Secure computation (also called multiparty computation, or MPC) is the branch of cryptography that studies the design of methods to execute computations on sensitive data held by multiple parties, without compromising the data privacy. It was introduced in the seminal works of Yao [Yao82] and of Goldreich, Micali, and Wigderson [GMW87b]. In a secure computation protocol, each participant has an input x_i. It allows all participants to jointly reveal to a subset of them the value $y = f(x_1, \cdots, x_N)$, where f is a public function, *while concealing all information about* $(x_1, \cdots, x_N)$ beyond y. The target function f is commonly represented as a *boolean circuit*, *i.e.* a directed acyclic graph with indegree two where the internal nodes are called *gates* and compute basic boolean operations, such as XORs or ANDs.

In the following, we overview some important considerations for secure computation: how do the parties communicate? How many of the parties can collude in an attempt to break security? How do we formalize security? As will become clear to the reader, the answers to these questions come in multiple flavors. Hence, along the way, we will also clarify which of these flavors will be our focus in this manuscript.

2.1.1 Network hypothesis

Protocols for secure computation commonly assume the availability of two network models: (1) the point-to-point network model, where each pair of parties is connected with a secure and authenticated channel, and (2) the broadcast model, where a party can broadcast an authenticated message to everyone (and everyone is guaranteed to receive the same message). We will typically assume that the communication proceeds synchronously, and ignore the issues of network delays, packet loss, and other non-trivial difficulties of distributed computing. Some rationale behind these choices is that they separate the challenges of protecting the privacy of the parties' inputs from the challenges of implementing reliable communication: authentication (which is typically achieved through digital signatures and certificates) and security of the channels (typically achieved through encryption) are themselves the subject of "traditional" cryptography, and considerations on network delays, the difficulty of broadcast, and synchronicity, are active research subjects in distributed computing. Nevertheless, we note that there are many examples, in the secure computation community, of efforts in the direction of tackling directly the challenges of securely computing functions over unstable networks.

2.1.2 Corruption threshold

We informally defined secure computation as a way to compute $y = f(x_1, \cdots, x_N)$ while concealing all remaining information about $(x_1, \cdots, x_N)$. However, this informal definition does not say anything about what happens when multiple parties join their forces (and combine their information) in an attempt to learn new information about the remaining inputs. A slightly less informal restatement addressing this setting could go as follows: even if a subset S up to $|S| = t$ parties combine the information they gathered throughout the protocol (we usually call this information their view of the protocol), they should not learn anything that cannot already be deduced from $(x_i)_{i \in S}$ (their combined inputs) and y (the output).

The value t loosely introduced above is called the *corruption threshold* – the maximum number of parties that can join their forces in an attempt to cheat. The choice of t plays a fundamental role in secure computation, as different choices for t yield to a completely different landscape of secure computation protocols. A particularly important consideration is the distinction between the *honest majority* setting (where $t < N/2$) and the *dishonest majority* setting ($t \geq N/2$):

- When a (strict) majority of the participants are guaranteed to remain honest throughout the protocol, there exists *unconditionally* secure computation protocol for essentially all functions of interest. In particular, security against the corrupted parties is guaranteed even if they are allowed unbounded running time.

- In contrast, if a majority of the participants can be corrupted, secure computation is known to be impossible to achieve with unconditional security for all but a few restricted type of functions. Hence, protocols in this setting will typically assume that the participants' computational power is bounded, and security is proven by reduction to a cryptographic hardness assumption.

A few comments are in order. First, the statements above hold in a specific security model where the participants do not deviate from the specifications of the protocol (and, in particular, do not quit the protocol before its completion). Considering other types of adversarial behaviors (that might deviate from the specifications, or abort early) complicates the landscape, but related distinctions between low values of t (typically $t < N/2$ or $t < N/3$) and high values of t can be made in other settings. Second, the statement about unconditional secure computation holds under the point-to-point network model, where each pair of parties can interact through a secure and authenticated channel.

In the rest of this manuscript, we will always be interested in the *maximal corruption threshold* setting, i.e., $t = N - 1$. If fact, a significant part of our discussions will focus on the case $N = 2$, where $t = N - 1$ is the only meaningful setting. The protocols we will discuss will therefore ultimately rely on computational hardness assumptions. Before moving on, I am including below a few references on the distinction between honest and dishonest majority (warning: the references prove results about securely computing functions, for a notion of *securely computing* which I have not yet introduced).

2.1.2.1 Historical notes

The existence of unconditionally secure protocols (for all N-variate functions over a finite domain) for every $t < N/2$ was first established by Ben-Or, Goldwasser, and Wigderson [BGW88]. The first observation that some functions (e.g., the N-wise OR function) cannot be computed with unconditional security was made in [CCD88] (they also observed that other functions, such as the N-wise XOR, admit unconditionally secure protocols for any $t \leq N - 1$). A stronger *zero-one law* was later established in [CK89], which showed that the only functions computable with unconditional security for $t \geq N/2$ are those of the form $f(x_1, \cdots, x_N) = f_1(x_1) \oplus \cdots \oplus f_N(x_N)$ (where $\oplus$ denotes the XOR). Since all such functions can be securely computed even when $t = n - 1$ (by direct reduction to the protocol of [CCD88] for securely computing an N-wise XOR), this yields an interesting phenomenon, where functions can either be computed with unconditional security only when $t < N/2$, or for any $t \leq N - 1$.

2.1.3 Notions of security

In secure computation, the security of a protocol for computing a function f between N parties $(P_1, \cdots, P_N)$, with respective inputs $(x_1, \cdots, x_N)$, is defined by considering an *adversary* $\mathcal{A}$, which can *corrupt* a subset of the parties. When a party is corrupted, $\mathcal{A}$ sees everything they see: their input, their private coins, and all messages they receive throughout the protocol. The protocol is secure if everything $\mathcal{A}$ can learn about the private inputs $(x_1, \cdots, c_N)$ can be efficiently computed from the inputs $(x_i)_{i \in \mathsf{C}}$ of the corrupted parties C, and the output $y = f(x_1, \cdots, x_N)$ (this is formalized by requiring that the distribution of everything $\mathcal{A}$ sees can be efficiently *simulated* given y and $(x_i)_{i \in \mathsf{C}}$).

2.1.3.1 Semi-honest versus malicious security

The two standard settings are the *honest-but-curious* setting, where the adversary observes the view of all corrupted parties, but the parties follow the specifications of the protocol, and the *malicious* setting, where the adversary fully controls the corrupted parties, and can make them deviate arbitrarily from the specifications of the protocol. The honest-but-curious setting often serves as an important first step in the design of a secure computation protocol, as several techniques and compilers can be used to enhance honest-but-curious protocols to the malicious setting.

In the rest of this manuscript, we will focus exclusively on the semi-honest setting: all definitions and results outlined afterwards, unless explicitely stated otherwise, refer to this setting. This is not to say that the setting of malicious security is not of fundamental importance; however, a full coverage of malicious security would require a second manuscript in itself, and considerations for semi-honest security are a necessary first step. Hence, we expect our coverage to be useful even for readers who would be mostly interested in this stronger setting.

2.1.3.2 Functionalities

In the short exposition given above, we focussed on the goal of letting all parties obtain the output $y = f(x_1, \cdots, x_N)$ of a function f evaluated on their joint inputs $(x_1, \cdots, x_N)$. More generally, we will often consider the setting where each party can receive a different output. We will say that N parties compute the *functionality* $f : (x_1, \cdots, x_N) \mapsto (y_1, \cdots, y_N)$ on their joint input $(x_1, \cdots, x_N)$ if each party P_i obtains $f_i(x_1, \cdots, x_N) = y_i$ at the end of the protocol. In some settings, the participants will want to compute a *randomized functionality*: on their joint input $\vec{x}$, the joint output $(y_1, \cdots, y_N)$ is randomly sampled by the functionality from a distribution parametrized by the joint inputs $\vec{x} = (x_1, \cdots, x_N)$, and each party P_i receives y_i. Equivalently, a randomized functionality $f = (f_1, \cdots, f_N)$ is a process mapping an input $\vec{x} = (x_1, \cdots, x_N)$ to a sequence $(f_1(\vec{x}), \cdots, f_N(\vec{x}))$ of random variables.

We write $f : (\{0,1\}^*)^N \to (\{0,1\}^*)^*$ to denote the randomized process mapping $(x_1, \cdots, x_N)$ to the random variables $(f_1(\vec{x}), \cdots, f_N(\vec{x}))$ over $\{0,1\}^*$. While we allow functionalities to take inputs of arbitrary length, we restrict our attention to protocols where all parties hold inputs of the same length $|x_1| = \cdots = |x_N|$.

For future reference, we outline below a formal definition of security in the semi-honest setting via the notion of t-privacy, which we define as an asymptotic notion with respect to the input length $\lambda = |x_1| = \cdots = |x_N|$ of the parties. For most of the following sections, and until explicitly mentioned otherwise, sticking with the intuitive notion of "secure against parties that follow the specifications" suffices. The reader can therefore safely skip what follows for now, and returns to it afterwards if they want to grasp more of the formalism.

2.1.3.3 Indistinguishability

The notion of t-privacy aims to capture the fact that the joint view of t corrupted participants does not allow them to learn anything about the other parties' inputs, beyond what they can deduce from their outputs. To formalize this notion, we want to say that the distribution of their joint views "looks like" a distribution that would be *independent* of the honest parties' inputs (more precisely, that depends solely on the corrupted parties' outputs). In cryptography, we define "look like" via the notion of indistinguishability between distributions. Because the distribution of the parties' views depend on an asymptotic parameter $\lambda = |x_1| = \cdots = |x_N|$, they are actually *families* of distributions parametrized by λ. Then:

Definition 2.1.1. *Two distribution families $(\mathcal{D}^0_\lambda, \mathcal{D}^1_\lambda)_{\lambda \in \mathbb{N}}$ are (perfectly, statistically, computationally) indistinguishable if:*

- *$\mathcal{D}^0_\lambda = \mathcal{D}^1_\lambda$ for every $\lambda \in \mathbb{N}$ (perfect indistinguishability).*
- *The statistical distance between $\mathcal{D}^0_\lambda$ and $\mathcal{D}^0_\lambda$ is bounded by a negligible function of λ*[1] *(statistical indistinguishability).*
- *For every probabilistic polynomial-time adversary $\mathscr{A}$, for every large enough λ, it holds that*

$$\left| \Pr_{x \overset{\$}{\leftarrow} \mathcal{D}^0_\lambda} [\mathscr{A}(x) = 0] - \Pr_{x \overset{\$}{\leftarrow} \mathcal{D}^1_\lambda} [\mathscr{A}(x) = 0] \right| \leq \mathsf{negl}(\lambda),$$

  where negl denotes a negligible function of λ (computational indistinguishability).

In our definitions below, we will use $\equiv$ as a shorthand to denote either perfect, statistical, or computational indistinguishability, each choice corresponding to a possible flavor of t-privacy (respectively perfect, statistical, and computational t-privacy). Later in this manuscript, if not specified otherwise, we assume $\equiv$ to refer to computational indistinguishability.

2.1.3.4 Formally defining privacy

In the following, we implicitly parametrize all sets, functions, and random variables by a protocol Π. Informally, t-privacy means that the joint view of a coalition of up to t corrupted players contains no information about the inputs of the remaining parties, beyond what can be deduced from the output of the protocol. To formalize this notion, we first define the notion of a view of a player.

Definition 2.1.2 (View)**.** *The view of party P_i (on a joint input $\vec{x} = (x_1, \cdots, x_N)$ from all parties), denoted $\mathsf{View}_i(\vec{x})$, is the (joint) distribution of the sequence of messages received by P_i during the execution of the protocol, and the sequence of the results of all coin tosses performed by P_i.*

[1] A function $g : \mathbb{N} \to [0, 1)$ is *negligible* if for every constant $c > 0$, there exists a λ' such that for all $\lambda \geq \lambda'$, $g(\lambda) < 1/\lambda^c$.

Since some of our definitions will also cover the case of randomized functionalities, whose outputs are sampled from a distribution, we also introduce a notation for the output distribution of a protocol Π:

Definition 2.1.3 (Output Distribution)**.** *We let $O_i(\vec{x})$ denote the distribution of the output of P_i after an execution of the protocol Π with a joint input $\vec{x}$.*

Given a subset C of $[N]$, we write $\mathsf{View}_C(\vec{x})$ to denote $(\mathsf{View}_i(\vec{x}))_{i\in C}$ and $O_C(\vec{x})$ to denote $(O_i(\vec{x}))_{i\in C}$; we use $O(\vec{x})$ as a shorthand for $O_{[N]}(\vec{x})$. Below, we define the notion of t-privacy for deterministic and for randomized functionalities; we borrow most of the formalism from [AL17].

Definition 2.1.4 (t-Privacy for deterministic functionalities [AL17])**.** *Let $f : (\{0,1\}^*)^N \mapsto (\{0,1\}^*)^N$ be an N-party deterministic functionality and let Π be a protocol. We say that Π is t-private if (1) Π computes f with perfect correctness, and (2) there exists a probabilistic polynomial-time algorithm Sim such that for every $C \subset [N]$ of cardinality at most t and every $\vec{x} \in (\{0,1\}^*)^N$ where $|x_1| = \cdots = |x_N|$, it holds that $\mathsf{Sim}(C, \vec{x}_C, f_C(\vec{x})) \equiv \mathsf{View}_C(\vec{x})$.*

While the above definition considers separately (with (1) and (2)) the issues of correctness and privacy, in the general case of randomized functionalities, the two notions are intertwined:

Definition 2.1.5 (t-Privacy for randomized functionalities ([AL17], def. 2.2))**.** *Let $f : (\{0,1\}^*)^N \mapsto (\{0,1\}^*)^N$ be an N-party randomized functionality and let Π be a protocol. We say that Π is* t-private *if (1) Π computes f with perfect correctness, and (2) there exists a probabilistic polynomial-time algorithm Sim such that for every $C \subset [N]$ of cardinality at most t and every $\vec{x} \in (\{0,1\}^*)^N$ where $|x_1| = \cdots = |x_N|$, it holds that*

$$\{(v, y) \;:\; y \leftarrow f(\vec{x}), v \leftarrow \mathsf{Sim}(C, \vec{x}_C, y_C)\} \equiv (\mathsf{View}_C(\vec{x}), O(\vec{x})),$$

where $(\mathsf{View}_C(\vec{x}), O(\vec{x}))$ denotes the joint *distribution of the corrupted parties' (final) views and the outputs of all parties in a run of the protocol on common input $\vec{x}$.*

A simulator according to the above definitions is simply a machine that produces emulated views for all corrupted parties. However, it is often convenient to view the simulator Sim as an *interactive* machine, which pretends to play the role of the honest parties during an execution of the protocol, and interacts with the corrupted parties. Under this viewpoint, Sim receives as input $(C, \vec{x}_C, f_C(\vec{x}))$, but also the random tapes of all corrupted parties; this is w.l.o.g. since in the above definition, Sim will sample these random coins itself when emulating the views.

Remark 2.1.6. *All definitions above treat the parties' input length $|x_1| = \cdots = |x_N|$ as an asymptotic security parameter. In this writeup, we will often abuse this formalism by considering functionalities f acting on* fixed length *inputs (or even sometimes randomized functionalities that take no inputs), and introduce an auxiliary* security parameter λ *for bounding the running time of the participants and of the simulator. Formally, this means that we will implicitly apply the above definitions of t-privacy to the following modified functionality:*

$$f' : ((x_1, 1^{i_1}), \cdots, (x_N, 1^{i_N})) \mapsto \begin{cases} f(x_1, \cdots, x_N) \text{ if } i_1 = \cdots = i_N \\ \perp \text{ otherwise,} \end{cases}$$

where 1^i *referes to a dummy string* $11\cdots1$ *of length* i. *This requirement guarantees that an algorithm that runs in time polynomial in its input length will run in time polynomial in* i *when given* 1^i *as input. We will call* $\lambda = i_1 = \cdots = i_N$ *the* security parameter *of the protocol.*

We end here our overview of the formal definition of t-privacy. We stress that our coverage is relatively superficial, and only intended to facilitate a more precise statement of the results discussed later on. For a much more extensive coverage, discussing technical subtleties which I overlooked on purpose in the above definitions, but also extensions to other settings (such as malicious security), I recommend the interested reader to consult Goldreich's book *Foundations of Cryptography, Volume 2* [Gol09].

2.1.3.5 Composition

In this manuscript, I will often discuss the construction of secure computation protocols using smaller protocols as a building block. In discussing their security, I will implicitely assume that the security of the building blocks is *preserved under composition*: that is, assuming that a higher-level protocol is proven secure when the parties are given access to an other functionality (the "building-block"), we want to deduce that when each invocation of the building block is replaced by a t-private protocol computing this functionality, the higher level protocol remains secure. Such statements are known as *composition theorems.* In the setting of semi-honest secure computation, which is the focus of this manuscript, suitable composition theorems have indeed been established (see e.g. [Gol09, Section 7.3.1] or [Can00]). Stating these theorems require some additional formalism and definitions, which I will not cover in this manuscript.

2.1.3.6 Historical notes

While secure computation was introduced in the work of Yao [Yao82] (for the two-party case) and of Goldreich, Micali, and Wigderson [GMW87b] (for the N-party case), the formal treatment of the security definitions for t-privacy only appeared later (though these papers already included sketches of what these definitions should be). The first formal treatment of privacy in secure computation appeared in the work of Beaver [Bea92b; Bea91] and of Micali and Rogaway [MR92]. The definitions were later refined and enhanced with composition theorems in the work of Canetti [Can00]. Par of the exposition above follows from the one of Goldreich [Gol09] and from Asharov and Lindell [AL17].

2.1.4 Paradigms for secure computation

There are currently three main paradigms for the design of secure computation protocols in the dishonest-majority setting with semi-honest security:

- **Secret-sharing-based** protocols build upon the seminal protocol of [GMW87b]. They are the most lightweight in terms of computation. They usually have communication proportional to the size of the circuit and require several rounds of interaction proportional to the depth of the circuit. Furthermore, they can be efficiently *preprocessed*: all cryptographic operations can be pushed to a one-time *preprocessing phase*, independent of the inputs. The online phase, where the actual computation takes place, is extremely lightweight (a few bits of communications and a few boolean operations per gate of the circuit).

- **Garbled-circuit-based** protocols build upon the seminal protocol of Yao [Yao82]. They usually require more communication and computation compared to their secret-sharing-based counterpart. However, they only require a constant number of rounds of interaction, making them a good choice over high latency networks. Some recent protocols (such as [WRK17]) combine these two paradigms to get the best of both worlds.
- **FHE-based** protocols build upon a cryptographic primitive called *fully homomorphic encryption*, introduced in a 2009 breakthrough [Gen09], to achieve an extremely low communication footprint (proportional only to the size of the inputs and outputs of the function) in a small number of rounds. This comes at the cost of using a considerably larger amount of computation, with heavy cryptographic operations for every gate of the circuit.

2.2 The GMW Protocol

We will start by describing the celebrated GMW protocol, introduced in the seminal work of Goldreich, Micali, and Wigderson [GMW87b]. It is an interesting starting point, because it has a simple description, but it already illustrates several of the techniques used in secure computation and allows us to explain many of the core challenges that drive modern research in the area. To keep notations simple and intuitive, we will focus on the case of two players, Alice and Bob, with respective private inputs a and b (though everything generalizes easily to an arbitrary number of participants). Let f be the two-input function that Alice and Bob wish to securely evaluate on (a, b). We view f as being represented by a boolean circuit with two types of gates, XOR gates (denoted $\oplus$ and represented by on the diagram of Figure 2.1) and AND gates (denoted by $\wedge$ and represented by on the diagram of Figure 2.1). It is a standard result that any polynomial-time computable function can be represented by a polynomial-size circuit over the $\{\oplus, \wedge\}$ basis. To provide visual support, we give an example of a simple boolean circuit in Figure 2.1.

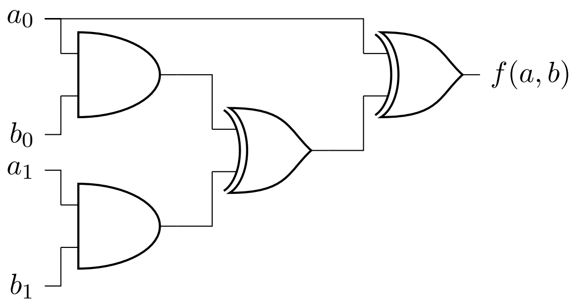


Figure 2.1: Example of a boolean circuit computing the function $f(a, b) = f(a_0|a_1, b_0|b_1) = a_0b_0 + a_1b_1 + a_0 \bmod 2$.

2.2.1 Description of the protocol

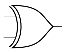
At a high level, *computing* a function f (such as the one of Figure 2.1) is done by writing the input bits on the input wires of the circuit (on the left). Then, every time a gate g has values (v_0, v_1) written on its input wires, we write the output $g(v_0, v_1)$ on the output wire of g. These intermediate steps of the computation are propagated this way through the circuit until values are written on all output wires, at which point they form the output of the computation. Our protagonists, Alice and Bob, will follow the same computation path. However, to maintain security, they must manage to perform the propagation of the intermediate steps of the computation without *seeing* the intermediate values (since this would reveal more than just the output of the function).

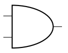
2.2.1.1 Hiding intermediate computations by secret sharing

The core technique used to let Alice and Bob manipulate intermediate values of the computation without seeing them is called *secret sharing*. An (n, t)-secret sharing is a method that distributes a secret value (here, some intermediate value of the computation) between N players (here we have Alice and Bob, so $n = 2$) such that (1) if t participants pool their shares together, they can reconstruct the secret, yet (2) if $t - 1$ (or less) participants collude, they do no learn anything (in an information-theoretic sense) about the secret. The value t is called the *threshold*; here, we will use the simplest possible setting, with $N = t = 2$. Typically, to share a bit v between Alice and Bob, we proceed as follows: Alice receives a random bit r, and Bob receives $s = v + r \bmod 2$. Observe that neither r nor s leak any information about v (they are both individually distributed as a random bit), yet v can be reconstructed given both shares: $v = r + s \bmod 2$.

2.2.1.2 Securely computing gates

The GMW protocol maintains the following invariant: at every step of the protocols, the parties hold shares of the values currently written on the wires. The only missing ingredient at this stage is a method to propagate the shared values: given shares of two inputs to a gate, the parties must obtain shares of the gate output. Of course, this should be done without reconstructing the values (or leaking any information whatsoever about them). Now, assume that the input wires of a gate g have values u and v respectively, and let (u_A, v_A) and (u_B, v_B) denote Alice and Bob's respectives shares of u and v (that is, $u_A + u_B = u \bmod 2$ and $v_A + v_B = v \bmod 2$). The goal is for Alice and Bob to come up with shares of $g(u, v)$. There are two possible gates:

 : this is the easy case. We have $g(u, v) = u + v = (u_A + v_A) + (u_B + v_B) \bmod 2$: Alice and Bob can therefore directly define $u_A + v_A$ and $u_B + v_B$ to be their respective shares of $g(u, v)$. As this does not require any interaction between them, it can of course not leak any information.

 : this is the hard case. We have $g(u, v) = uv = u_A v_A + u_A v_B + u_B v_A + u_B vB \bmod 2$. Here, the computation of $g(u, v)$ contains *cross terms* $u_A v_B$ and $u_B v_A$: Alice and Bob cannot possibly get shares of $g(u, v)$ solely through local computation. To securely evaluate an AND gate, we will need a method to let Alice and Bob obtain *additive shares of the cross terms*: if Alice and Bob get respective shares (s_A, t_A) and (s_B, t_B) of the cross terms $(u_A v_B, u_B v_A)$, they can define their shares of $g(u, v)$ to be $u_A v_A + s_A + t_A$ and $u_B v_B + s_B + t_B$ – the reader will easily check that those indeed sum to uv modulo 2.

The *missing ingredient* is, therefore, a protocol that takes as input a bit x from Alice, a bit y from Bob, and outputs shares of the product xy. Let us call this a *secure product*. Propagating shares through an AND gate can be done as explained above, using two invocations of a secure product (on inputs (u_A, v_B) and (v_A, u_B) respectively). We will explain how to get this missing ingredient in the next section; before that, let's wrap up.

2.2.1.3 Full construction

Without loss of generality, we can consider that Alice and Bob start the protocol with shares of the inputs. For example, if Alice has input bit a_0, she can define her share to be a_0, and Bob define his share to be 0 (alternatively, Alice could send a random r to Bob and define her share to be $a_0 + r \bmod 2$ if we wanted the shares to be uniformly distributed – but for the inputs, this is unnecessary. We will need uniformly distributed shares for the *intermediate values* since they are the values that should not be known to any party). Alice and Bob propagate the shares through all the wires, computing the XOR gates locally, and using two invocations of the secure product protocol for each AND gate. Once they obtain shares of the output wires of the circuit, they can simply exchange their shares and reconstruct the output.

Beyond reconstructing the outputs (which both parties must learn anyway), the secure products are the only parts where Alice and Bob communicate. Hence, if this protocol never leaks any information, and if Alice and Bob honestly play throughout the entire protocol as required by the description, they will indeed learn the output $f(a, b)$, and nothing more. In cryptography, we refer to players that follow the rules as being passive or honest-but-curious: they will try to learn information by looking at the transcript of the protocol, but will not actively cheat by deviating from the specifications of the protocol. The (much more demanding) setting where Alice and Bob could actively cheat is the active or malicious setting, but let us set this harder goal aside for now. We have:

Theorem 2.2.1 (GMW – informal)**.** *Assume that there exists a secure product protocol. Then there exists a two-player protocol securely computing any two-input functionality f in the honest-but-curious model. The protocol uses two invocations of a secure product for each AND gate in the boolean circuit representation of f over the $\{\oplus, \wedge\}$ basis.*

2.3 Oblivious Transfer, the Missing Ingredient

The GMW protocol requires a secure product: a two-party protocol in the honest-but-curious model where Alice has an input bit x, Bob has an input bit y, and both parties obtain random shares of xy (without learning anything). The *de facto* method to construct a secure product is to rely on an *oblivious transfer* (OT).

2.3.1 Defining oblivious transfer

An *oblivious transfer* (OT) is a two-party protocol between a sender and a receiver. The sender holds two entries (s_0, s_1), and the receiver holds a *selection bit b*. At the end of the protocol, the receiver should learn s_b. The security requirements are twofold:

- (*Sender security*) the receiver does not learn anything about s_{1-b} (that is, the receiver learns *at most one* of the two entries).

- (*Receiver security*) the sender does not learn anything about b.

Note that relaxing any of these requirements would make OT trivial: without sender security, we could just let the sender send (s_0, s_1) to the receiver, and without receiver security, the receiver could just send b and receive s_b. We represent in Figure 2.2 an ideal functionality for oblivious transfer. Ideal functionalities are an abstraction of what we want from a primitive: think of them as a trusted third party, that would receive inputs and provide outputs through perfectly secure channels, and who always behaves exactly as intended [2].

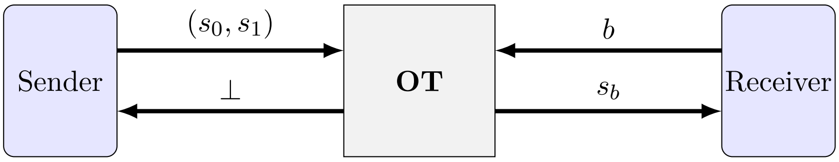


Figure 2.2: Ideal functionality of the (one-out-of-two) oblivious transfer.

2.3.1.1 From OT to secure products

Given access to an OT functionality, it is straightforward to implement the secure product functionality. Alice, with input bit x, plays the role of the sender in an OT protocol. She picks a uniformly random bit r_A and sets her inputs to the OT to $(s_0, s_1) \leftarrow (r_A, x + r_A)$ (from now on, additions are modulo 2 unless indicated otherwise). Bob plays the role of the receiver and uses his input y as the selection bit. Observe that Bob's output, denoted r_B, is therefore r_A if $y = 0$ and $r_A + x$ if $y = 1$; that is, it holds that $r_B = r_A + xy$, hence (r_A, r_B) for uniformly random shares of xy.

2.3.1.2 Historical notes and remarks

Oblivious transfer was first introduced in the work of Rabin [Rab81] (a similar concept was introduced earlier by Wiesner [Wie83] under the name "conjugate coding"). the modern way to define it is due to Even, Goldreich, and Lampel [EGL82]. Oblivious transfer is without a doubt the most fundamental primitive for secure computation with dishonest majority: its existence is both necessary and sufficient for general secure computation in the presence of a dishonest majority [GV88; Kil88; IPS08] (our presentation of GMW demonstrates the sufficiency in the two-party honest-but-curious setting, but this can be generalized to the N-party malicious setting).

2.3.2 Constructing oblivious transfer

The existence of OT can be seen as the basic assumption underlying the existence of secure computation for arbitrary functionalities. This is similar to how the existence of one-way functions forms the basic assumption underlying most of symmetric cryptography, as most symmetric primitives (block ciphers, stream ciphers, pseudorandom generators, digital

[2] Using an ideal functionality allows us to conveniently abstract out the specific implementation of a protocol when using it in a higher level protocol. It is a standard theorem in secure computation that if a higher-level protocol is secure in the honest-but-curious model when the parties have access to ideal functionalities for its components, it remains secure when the ideal functionalities are replaced by secure protocol instantiating the components; see also our discussion in Section 2.1.3.5.

signatures, universal one-way hash functions, commitment schemes, and many more) are equivalent to one-way functions. But more broadly, OT is typically treated as a *resource* for secure computation: the goal of efficient secure computations reduces to the goal of efficiently executing many instances of OTs, or *consuming* OTs. Therefore, one of the most fundamental (perhaps the most fundamental) questions for the study of efficient secure computation is:

How can we generate efficiently a very large number of oblivious transfers?

Due to its fundamental importance for secure computation, countless oblivious transfer protocols have been designed – way, way too many to cite them all (but the reader might want to check [NP01; LGD21; DGH+20; GJJM20; CNs07; Kal05; AIR01; BPT85; GM00; GKM+00; CCM98; CK88; Lip05; CO15; CKWZ13; CCG+21; FMV19; CDLR16; GH11; LZ09; Wul09; Hai08; DFMS04; Hai04; DHRS04; MR19; BC15; MRR21; MS20; CSW20; KN09; GH08; HCR02; Bea98; IKO+11; GWZ09; PVW08; Din01; NP99b; BBCS92; BM90] for an illustrative sample). OTs can be based on physical assumptions, such as the existence of noisy channels [CK88] or tamper-proof hardware tokens [CKS+14], or on the assumption that the receiver has bounded memory [Din01]. It can be realized using quantum communication channels under symmetric assumptions [GLSV21]. More commonly, it can be instantiated under a variety of standard cryptographic assumptions, such as assumptions related to discrete logarithms [BM90], lattices [PVW08], factoring [BPT85], coding theory [Ale03], or isogenies [LGD21], among others.

For completeness, we describe a simple construction of oblivious transfer due to Bellare and Micali [BM90]. The protocol integrates some modifications due to [NP01]. To avoid disrupting the flow of the presentation, we assume some familiarity with cyclic (finite) groups. The parameters define a multiplicative group $\mathbb{G}$ of (known) prime order p, and two generators (g, h) of $\mathbb{G}$. We assume that the receiver does not know the discrete logarithm of h in base g (that is, the unique $x \in \mathbb{Z}_p$ such that $g^x = h$): this can be ensured by letting the sender pick h randomly and relying on a group $\mathbb{G}$ where the discrete logarithm problem is conjectured to be intractable (the existence of such groups is one of the most basic assumptions of cryptography). The protocol also uses a hash function $H : \mathbb{G} \mapsto \{0, 1\}$ (which is modeled as a random oracle in the security analysis). The *Bellare-Micali* protocol involves two messages:

- The receiver, with selection bit b, picks a random $r \xleftarrow{\$} \mathbb{Z}_p$ and set $\mathsf{pk}_b \leftarrow g^r$ and $\mathsf{pk}_{1-b} \leftarrow h/\mathsf{pk}_b$. Observe that by construction, the receiver knows the discrete logarithm of pk_b, but not that of pk_{1-b}. They send pk_0 to the sender.

- The sender with input (s_0, s_1) computes $\mathsf{pk}_1 \leftarrow h/\mathsf{pk}_0$. For $\sigma = 0, 1$, the sender picks $r_\sigma \xleftarrow{\$} \mathbb{Z}_p$ and encrypts s_σ using the *hashed ElGamal* encryption scheme with public key pk_σ, as follows:
$$E_\sigma \leftarrow (g^{r_\sigma}, H(\mathsf{pk}_\sigma^{r_\sigma}) \oplus s_\sigma).$$
They send (E_0, E_1) to the receiver.

- The receiver computes $H((g^{r_b})^r) = H(\mathsf{pk}_b^{r_b})$ and recovers s_b from $H(\mathsf{pk}_b^{r_b}) \oplus s_b$.

The correctness of the protocol can be checked routinely. The protocol is unconditionally secure for the receiver (since c is uniformly distributed over $\mathbb{G}$ for any value of b). The sender security can be proven (in the random oracle model, which models the hash function H as a random function – this is a common heuristic in cryptography) under the decision Diffie-Hellman assumption, a standard cryptographic assumption which states that given (g, g^x, g^y) for random exponents x, y, it is infeasible to distinguish g^{xy} from a random group element. We note that more advanced constructions, such as the one of Naor and Pinkas [NP01], can get rid of the random oracle heuristic, at the cost of a slightly more complex protocol (we focus here on the Bellare-Micali construction since it is among the simplest to describe).

2.4 Considerations on Efficiency

2.4.1 On communication

If we denote by λ the bitlength of the elements of $\mathbb{G}$, the Bellare-Micali OT protocol communicates $3\lambda + 2$ bits in total. The modern approach to instantiate $\mathbb{G}$ is to use an appropriate elliptic curve, and a typical value of λ in this setting is $\lambda = 256$. Hence, the full protocol requires overall 758 bits of communication using a standard instantiation.

This is arguably not much, but we invite the reader to reflect on the cost of using such an OT protocol in the GMW protocol: the above cost would translate to about 1500 bits of communication *for each AND gate of the circuit.* But even for simple functions, boolean circuits can be *really, really big.* For example, in [KsS12], the boolean circuit for computing the edit distance between two strings of length 4095 bits (a task of interest in scenarios such as privacy-preserving computations on genomic data) has 5,901,194,475 AND gates. Using GMW with the Bellare-Micali protocol, this translates to an overall communication of around 1 Terabyte. Needless to say, that is a whole lot of communication.

These back-of-the-enveloppe calculations show that bandwidth is a scarce resource for secure computation, as seminal protocols such as GMW require a gargantuan amount of communication. Whether it is possible to securely compute complex functions using significantly less computation is a crucial and very active research topic:

Can we construct secure computation protocols with a low communication overhead?

This question can be (and has been, and is) studied from many angles, ranging from very practical considerations to highly theoretical questions: can one construct oblivious transfers with significantly less communication? Can one avoid the “communicate at each gate” bottleneck of the GMW protocol (a goal usually referred to as *breaking the circuit-size barrier*)? Can one prove lower bounds on the necessary amount of communication for secure computation?

2.4.2 On computation

Similar back-of-the-enveloppe calculations give an intuition of the computational overhead of GMW. In the Bellare-Micali protocol, the sender computation is dominated by 4 exponentiations in the group $\mathbb{G}$ (2 for the receiver). Looking at a benchmark[3], if we instantiate $\mathbb{G}$ using the elliptic curve curve25519 [Ber06] (one of the most standard choices of cryptographic groups), computing 4 exponentiations over $\mathbb{G}$ requires slightly more than $222\mu s$ over a powerful server (AWS EC2 c5.9xlarge). Using again the circuit of [KsS12] as an example (which computes the edit distance between 512-byte strings, a rather simple function on small

[3] `zka.lc`

inputs), this translates to about 740 hours of computation on the server (this count is based solely on exponentiations and ignores many other factors that could contribute to the overall cost).

2.4.3 The computation/communication tradeoff in modern protocols

The rough calculations above demonstrate that the seminal approach to secure computation requires a tremendous amount of communication and computation. Where do modern protocols stand in this regard? The (very) high-level summary is that the latest proposals can solve either one of these limitations – *but not both at once.* That is, we have protocols that involve considerably lower amounts of computation, but still suffer from a high communication overhead, or protocols with much lower communication, but very high computational overhead. More concretely:

- The first category follows the high-level template of the GMW paradigm, but includes a fundamental twist: the concept of *oblivious transfer extensions*, which we will cover in Section 2.5. This method allows to compute all necessary oblivious transfers using a *constant number* of "expensive" OTs (such as those obtained using the Bellare-Micali protocol), followed by cheap computations to *extend* this constant number of OTs into an *arbitrary* number of oblivious transfers. Concretely, this brings computational costs down by five to six orders of magnitude. However (and though it *also* yields some savings in communications), these techniques still require sending of the order of 100 bits of data *for each gate of the circuit.* The need to send data for each gate of the circuit is probably the strongest limitation of this line of work and is known as the *circuit size barrier.* The remainder of this primer will mostly cover this approach.

- The second category departs entirely from the GMW paradigm and emerged as a consequence of the breakthrough result of Gentry in 2009 [Gen09], who introduced the first *fully homomorphic* encryption (FHE) scheme. Roughly, such a scheme allows computing arbitrary functions on encrypted data. This means that the entire communication complexity can be reduced to just exchanging encryptions of the inputs, then encryptions of the outputs: the rest is just local computation. However, this comes at the cost of running the entire computation over FHE-encrypted data which, to say the least, requires *a lot* of computation. Back in 2009, this was mostly a purely theoretical feasibility result. After one-and-a-half decade of work from hundreds of researchers and companies, the costs became much more manageable, but the most efficient schemes around can still process *at most* a few dozen gates per second – and the room for improvement is becoming tiny[4]. Going back to our example of the edit distance, using a rough figure of 40 gates/second, we are talking about around 60 *months* of computation on a powerful server. Nevertheless, in some scenarios with a very powerful server, a weak client, and a very restricted bandwidth, the FHE-based approach offers some unmatched features.

[4] One of the latest active trends is to develop specialized hardware tailored to FHE operations, which might gain a few more orders of magnitude compared to optimized software implementations, but will likely not become part of your average laptop.

2.4.4 On our approach

So far, we have not addressed any of the work done by the author. In essence, our work fits mostly in the first category: an immense research effort is devoted by the community to reducing the cost of FHE, and this is by all means a well-motivated problem that deserves a high amount of attention. But for the purpose of someday obtaining *truly efficient* secure computation protocols, the author's thesis is that the first approach is the most promising. Truly efficient solutions would require reducing the cost of FHE-based solutions by, say, at least 6 orders of magnitude, which seems (in our opinion) largely out of reach of current cryptographic techniques. In contrast, "GMW-style" secure computation already achieves competitive computational efficiency, and the core problems to address lie in reducing its communication overhead. As we will show, the author had multiple significant contributions to this goal that, on the practical side, already reduced the communication complexity of GMW-style secure computation by a large margin (as low as about 4 bits of communication per AND gate of the circuit). On the theoretical side, observing that the previous results do still not break the *circuit size barrier*, part of our work focuses on one of the most intriguing problems in the area: can the communication complexity of these approaches go beyond the size of the circuit? Looking ahead, we show that the answer is yes in many settings (which is a surprising result, that had been conjectured to be impossible by prominent researchers). This broadens our understanding of the limits of secure computation – but the complexity of the techniques involved currently confines these new approaches to the setting of purely asymptotic results.

2.5 Extending Oblivious Transfers

As we saw above, secure computation *a la* GMW requires computing a very large number of oblivious transfers: almost 12 billion for the edit distance example which we outlined in Section 2.4. In turn, known protocols for oblivious transfer, such as the Bellare-Micali OT, require expensive cryptographic operations. One may ask: could we instead construct OT using cheaper operations? Before we outline the way researchers have approached this problem and the technical barriers it faces, we will take a small detour through the worlds of symmetric cryptography and public key cryptography, which are the key to understanding at a high level the difficulties associated with constructing OTs using only cheap computations.

2.5.1 Symmetric versus public key cryptography

It is common to distinguish between two broad categories of cryptographic primitives: *symmetric key* cryptography, and *public key* cryptography. While this is a very broad and coarse-grained separation, it provides an efficient way to anticipate whether a cryptographic primitive will require some *algebraic structure*, which in turn severely constraints the computational efficiency of the constructions.

2.5.1.1 Symmetric key cryptography

Historically, symmetric key cryptography refers to cryptographic primitives that require a pre-established secret to be shared among the participants. This is the cryptography from the pre-1970 era, where Alice wants to send a message m to Bob, and both share a common key

K, unknown to the attacker, which can be used for both encryption and decryption. In its modern acceptance, the term covers more broadly a vast network of cryptographic primitives which are *equivalent* to this notion of encryption with a shared secret key, in the sense that the existence of one implies the existence of the other. For example, one-way functions (functions which are easy to evaluate, but hard to invert), digital signatures, pseudorandom number generators, (preimage resistant) hash functions, commitments schemes (which allow locking a secret value inside an opaque box, such that the content of the box cannot be modified after the box is opened), and many more, are all known to be (existentially) equivalent to symmetric-key encryption.

All these primitives share a nice unifying feature: *they don't require algebraic structure*. Remember the Bellare-Micali OT construction: we had to introduce a finite cyclic group (which, in the real world, will typically be instantiated with a suitable elliptic curve) endowed with a hard mathematical problem. In contrast, to build (for example) a one-way function, there is no need to invoke any algebraic structure: any "sufficiently complex" function will do. In a sense, anyone can build a one-way function, by composing sufficiently many arbitrary functions (provided, say, that some of them are not too local – their output bits depend on many input bits – and are not too close to linear functions). For example, Oded Goldreich suggested in [Gol00] (perhaps the most celebrated unpublished preprint of cryptography) that evaluating random (hardwired) predicates on random (hardwired) small subsets of the bits of the inputs would likely yield a one-way function.

Their lack of algebraic structure is perhaps the most fundamental aspect that separates them from the rest of cryptography to the point that theoretical cryptographers now typically model "symmetric cryptography" as the set of cryptographic primitives that can be constructed from unstructured functions (formally, whose existence is implied by a random oracle, which is a uniformly random – hence unstructured – function that everyone can access through its input-output behavior). This broadens the coverage of symmetric cryptography to include primitives such as collision-resistant hash functions, which also don't require structure, but are not known to be equivalent to symmetric encryption.

Being free to navigate through the vast space of functions without consideration for their algebraic structure, the cryptosystem designers have typically managed to construct symmetric primitives which are very lightweight[5]: modern computers can hash or encrypt several hundreds of megabytes of data per second on one core of a standard laptop.

2.5.1.2 Public key cryptography

This category covers essentially all primitives that do not fit in the previous category. In the 70s, cryptography was revolutionized by the introduction of the first public key cryptosystems (the RSA cryptosystem [RSA78]) and the first key exchange protocols (the Diffie-Hellman key exchange [DH76]). These schemes emerged from mathematical hardness assumptions tailored to algebraic structures: the hardness of computing the order of the ring $(\mathbb{Z}_n^*, \times)$ of invertible integers modulo a product n of two large primes, or the hardness of computing random discrete logarithms in the multiplicative group $\mathbb{F}_p^*$ of a prime-order field $\mathbb{F}_p$.

Following these breakthrough results, countless advanced cryptographic primitives have been introduced, which all rely in one way or the other on mathematical objects with a rich

[5] Digital signatures are a slightly different beast: even though they technically don't require any algebraic structure, the most efficient constructions so far had to assume structure. But apart from this outsider, the statement is true of any other symmetric primitive we know of.

algebraic structure. Unlike symmetric primitives, which are (for most of them) known to be equivalent to each other, public key primitives are typically not known to be equivalent to each other: what unifies them is precisely their apparent need for algebraic structure. This algebraic structure is what *enables* their advanced features – typically, the ability to agree on a shared key unbeknownst to all observers, or the ability to create keys that can encrypt messages, yet cannot decrypt ciphertexts. At the same time, this algebraic structure is what makes them considerably less efficient: the number of elliptic curve operations one can perform on one core of a standard computer is around a thousand per second, 5 to 6 orders of magnitude slower than symmetric key operations[6].

2.5.1.3 Wrapping up: cheap versus expensive computations

In light of the above, the usual way to categorize which operations are *cheap* versus which operations are *expensive* is to distinguish whether they require symmetric operations (such as evaluating hash functions, encrypting with AES, using a pseudorandom number generator, etc) or public key operations (elliptic curve point multiplication, exponentiations in a large finite field or an RSA group, etc). While this is a relatively coarse-grained notion (using many symmetric operations can become more expensive than using a few public key operations), the large efficiency gap between the two (at least five orders of magnitude in the benchmark we used) makes this a relatively stable and useful distinction.

Of course, a public key primitive is only categorized as such because we do not *know* of a way to construct the primitive using only unstructured hardness (as we do for one-way functions or hash functions). Cryptographers have been surprised in the past: digital signatures were long thought to be a typical example of a public key primitive, as all known constructions required algebraic structure, and some of the first constructions were in a sense dual to public key encryption (where the public key was used for verification and the secret key for signing a message). This led cryptographers to believe that digital signatures and public key encryption were of a similar nature. Yet, in a surprising twist of events, digital signatures were shown to be *equivalent to one-way function* in a sequence of papers by Naor and Yung in 1989 [NY89] and by Rompel in 1990 [Rom90].

Coming back to the focus of this section, equipped with the above considerations, the question of constructing *cheap* oblivious transfers becomes:

Is it possible to build oblivious transfer solely from unstructured primitives? In other terms, does oblivious transfer belong to symmetric cryptography?

2.5.2 On building oblivious transfer from symmetric primitives

Unfortunately, the answer to the above question is essentially negative. Of course, one can never hope to *unconditionally* provide a negative answer to any such question: after all, we believe (though we don't have proof of that) that both oblivious transfer and symmetric cryptography *exist* (unconditionally). Therefore, oblivious transfer is logically implied by symmetric cryptography, and we cannot hope to rule out any construction of oblivious transfer from symmetric primitives (after all, the construction that ignores the symmetric primitive and builds an OT from scratch would be a valid construction, provided that OT exists).

[6] All benchmark mentioned are taken from `https://www.bearssl.org/speed.html`, and ran on the same computer.

However, one can show that *black-box* constructions of oblivious transfers from symmetric primitives are impossible.

Informally, a black-box construction of a primitive A from a primitive B is a construction that ignores specific details about the implementation of B, and relies only on its input-output behavior. This is closely related to the notion of oracle access in complexity theory: a black-box reduction of the existence of A to the existence of B is an efficient oracle algorithm R (the reduction) such that R^B implements the primitive A (*i.e.* the algorithm R, given oracle access to B, yields an instance of the primitive A). This captures a large majority of all known constructions of cryptographic primitives; for example, all known reductions mentioned above (between one-way functions, digital signatures, hash functions, symmetric encryption, pseudorandom generators, or commitment schemes) are black-box.

2.5.2.1 Impossibility of black-box constructions of OT from symmetric cryptography

Now, here comes the catch: an oblivious transfer implies a key exchange protocol, *i.e.*, a two-party protocol that allows Alice and Bob to agree on a shared key, without revealing the key to any external (passive) observer. To see this, consider the following construction of a key exchange protocol:

- Alice, playing the role of an OT sender, samples a random key $K \overset{\$}{\leftarrow} \{0,1\}^\lambda$, and defines $(s_0, s_1) \leftarrow (K, 0^\lambda)$ to be her two OT inputs (here, λ is the security parameter, used as the key length, and 0^λ denotes a string of λ zeroes).
- Bob, playing the role of the receiver, sets his selection bit b to 0.
- At the end of the protocol, Bob receives $s_0 = K$ from the OT. Alice and Bob output K as their shared key.

It is clear that at the end of the above protocol, Alice and Bob agree on the same key. I claim that this protocol is also a *secure* key exchange.

Proof. Assume towards contradiction that there is an efficient attacker $\mathcal{A}$ that can recover K (or distinguish it from random) by observing the transcript of the interaction. We use a sequence of game hops, where we gradually change the protocol:

1. In the first game hop, we let Bob use the selection bit 1 instead of 0. By the receiver security of the OT protocol, this change cannot be noticed by $\mathcal{A}$: otherwise, the sender could emulate the attacker (since we assumed the attacker is efficient) and distinguish between the selection bits $b = 0$ and $b = 1$, breaking receiver security.
2. In the second game hop, we replace Alice's input $(K, 0^\lambda)$ by $(0^\lambda, 0^\lambda)$. By the sender security of the OT protocol, is change cannot be noticed by $\mathcal{A}$: otherwise, the receiver with input $b = 1$ could emulate $\mathcal{A}$ locally, and learn something about $s_{1-b} = s_0$ (here, whether $s_0 = 0^\lambda$ or not), contradicting sender security.

Hence, we made two changes to the original protocol and argued that $\mathcal{A}$ cannot notice any of these changes unless we get a contradiction to the OT security. But this implies that $\mathcal{A}$ must still successfully find K in the last game above (otherwise, $\mathcal{A}$ can be used to build an efficient distinguisher between the original game and the last game), which is impossible: the last game is information-theoretically independent of K! This is a contradiction, which concludes the proof. □

Hence, if we could base oblivious transfers on symmetric primitives, by the above construction, we could base key exchange on symmetric primitives. However, it was shown by Impagliazzo and Rudich in their seminal paper on black-box impossibility results [IR89] that this is impossible via black-box reductions. More generally, Impagliazzo and Rudich showed that there can never exist a black-box reduction of a primitive that implies key agreement (such as oblivious transfer) to a primitive implied by a random oracle (such as hash functions, one-way function, symmetric encryption, and many more). Hence, any attempt to build oblivious transfer from symmetric primitives must rely on non-black-box reductions, which have proven elusive so far (in this context), and would need to base key agreement on symmetric cryptography, which is one of the most challenging open problems in cryptography, of an overture comparable to that of the famous P versus NP problem.

2.5.3 The way around: many cheap OTs from few expensive OTs

For a long time, the above considerations were seen as a clear indication that one had no hope of constructing cheap oblivious transfer: barring a breakthrough, the use of expensive algebraic structures appears unavoidable. This was eventually challenged in two groundbreaking works. First, in 1996, Beaver showed in [Bea96] that assuming only the existence of one way function, it was possible to generate OTs using only cheap symmetric operations *in an amortized sense*. In simple terms, what Beaver showed is the following: even though generating a *single* OT requires expensive algebraic operations, generating n OT for a large n does not require repeating n times the expensive operations used in the single OT generation! Beaver showed something much more powerful:

Theorem 2.5.1 (Beaver OT extension – informal)**.** *Assuming only the existence of a one-way function, there exists a protocol that uses only λ calls to an oblivious transfer primitive, and allows two parties to execute an arbitrary polynomial number $N(\lambda) = \mathrm{poly}(\lambda)$ of oblivious transfers.*

In other words, Beaver showed that after spending some effort generating a small number of "base" OTs (typically, one can think of λ as being 128), two parties can generate an arbitrary amount of OTs, and this process will involve only cheap symmetric operations (beyond the cost of generating the base OTs). Beaver's *OT extension* protocol was mostly of theoretical interest but showed that the black-box barrier for basing OT on symmetric cryptography was not necessarily a nail in the coffin of efficient MPC, for one could still generate arbitrarily many OTs using a small constant amount of expensive algebraic operations. All that was needed was to design a *practical* OT extension protocol.

2.5.3.1 Practical OT extension

This is precisely what was achieved in a second groundbreaking work by Ishai, Kilian, Nissim, and Petrank, in 2003 [IKNP03].

Theorem 2.5.2 (IKNP OT extension – informal)**.** *Under a suitable hardness assumption about cryptographic hash functions, there exists a protocol that uses only λ calls to an oblivious transfer primitive, and allows two parties to execute an arbitrary polynomial number $N(\lambda) = \mathrm{poly}(\lambda)$ of oblivious transfers. Furthermore, the protocol enjoys the following efficiency features (ignoring the one-time cost of the base OTs):*

- *The computation is dominated by three calls to the underlying hash function per OT.*
- *The communication for executing n OT on strings of length ℓ is $2 \cdot (\ell + \lambda) \cdot n$.*

Later works improved over IKNP in various ways, reducing the communication to $(2\ell + \lambda) \cdot n$ [KK13; ALSZ13] (in the important case of $\ell = 1$, this is a factor two reduction) or even to $\approx 2n \cdot \lambda / \log \lambda$ when transferring short secrets [KK13], extending to the setting of security against malicious adversaries [KOS15; ALSZ15; PSS17], or generalizing the functionality to 1-out-of-n OT [OOS17]. Another work [LZ13] showed that the number λ of base OTs was essentially the best possible, by showing that an OT extension protocol using fewer base OT would imply oblivious transfer (hence can again not be based on symmetric primitives, baring breakthroughs). When referring to OT extensions using the IKNP approach or some of its follow-ups, we will use the terminology *IKNP-style* OT extension.

2.5.3.2 Wrapping up: barriers in GMW-based secure computation

Using oblivious transfer extensions, such as IKNP and its follow-ups, considerably mitigates the computational limitations of OTs: the 4 exponentiations required by the Bellare-Micali OT protocol are replaced by three invocations of a hash function, which represents a speed up between 5 and 6 orders of magnitudes. In our example from Section 2.4.2, this translates to a reduction of the computation time from about 740 hours to roughly *one minute*.

The communication is also significantly decreased – e.g. for transferring bit-secrets (as in GMW), about 130 bits of communication are required compared to the 758 bits of Bellare-Micali. Yet, this is by no mean an improvement comparable to the efficiency gain: each OT still require communication of the order of a hundred bits, which translates to Terabytes of communications even for simple functions such as edit distance. This brings us to where GMW and its many variants were until about 2018 or so: OT extension techniques, coupled with the notion of *secure preprocessing* which we will cover in the next section, make it feasible to run reasonably large secure computations on modern hardware in a reasonably short amount of time. However, the amount of communication required makes running these protocols prohibitive in any bandwidth-restricted scenario – in particular, over the Internet.

2.6 Precomputing Oblivious Transfers

We saw above that computing tons of oblivious transfers can be reduced to computing a few hash evaluations per OT. While this represents a tremendous improvement over directly running expensive OTs such as the Bellare-Micali protocol, this can still be too slow for real-world applications. In our example of securely computing the edit distance between two length-4095 strings, we observed that using OT extension reduces the runtime on a standard laptop from hundreds of hours to merely a minute; however, *merely a minute* is too long of a waiting time in an online setting, where many participants could want to run secure computations. Consider for example a website that would like to show personalized ads to its users, but without compromising their privacy: this could be done by securely evaluating an algorithm on the private content of the user browsing Cookie. But if running this computation takes one minute, this means that users will need to wait for a full minute before reaching the website, which is an unacceptable slowdown[7].

[7] This is one of the reasons why, so far, no legislation prohibits websites from collecting private browsing information, even though secure computation could in principle allow executing any task the websites were

In this section, we describe another fundamental observation of Beaver (yes, the same Beaver who introduced OT extension) which, in many settings, can strongly mitigate the overhead of running a secure computation protocol, and enable its use in scenarios where even a few seconds of delay would be too much.

The key insight: preparing the computation ahead of time. In many scenarios, the parties receive their inputs (x, y), and would like to obtain the result *on the spot*, shortly after. In our example of a user accessing a website where an ad provider wants to print a targeted banner, computed securely from the user's browsing history (*i.e.* without revealing the browsing history), the participants learn their inputs at the last minute (their browsing history depends on recent events) and want to obtain the result of the computation, say, within a matter of milliseconds.

In this context, a natural question to ask is: is it necessary to start the heavy computation only when the parties receive their inputs? In other words, could the parties somehow *pre-execute* part of the computation, before knowing their inputs (and before getting the result becomes urgent)? This question was answered positively by Beaver in 1995, in a seminal paper [Bea95] that paved the way to efficient secure computation protocols. Beaver's observation was the following: secure computation via GMW boils down to executing many oblivious transfers on entries that depend on partial evaluations of the function on the parties' inputs. Yet, Beaver noticed, the parties can execute *all* their oblivious transfers on *uniformly random* inputs (this can be done long ahead of time, and all these oblivious transfers can be executed in parallel), and later *derandomize* each oblivious transfer, once the inputs to the protocol are known. Furthermore, this derandomization procedure is extremely fast (a few XORs and ANDs), requires little communication (only three bits per OT), and is information-theoretically secure (it does not rely on any unproven assumption).

2.6.1 The preprocessing model

Beaver's result suggests the following natural approach to speed up secure computation in practice:

- First, during a **preprocessing phase**, the parties jointly run many oblivious transfer protocols in parallel on uniformly random inputs and store the outputs. This phase might be expensive, but (crucially) it is entirely independent of the inputs to the protocol, and can therefore be executed ahead of time.

- Second, during the **online phase**, when the parties receive their inputs, they execute the GMW protocol. For each OT they need, the parties retrieve a random OT from the preprocessing phase, and derandomize it – this can be viewed as *consuming* a part of the preprocessing material computed ahead of time (since, as we will see shortly, a random OT can only be used once to derandomize an OT). This allows the online phase to run fast (it does not involve any "cryptographic" operations) and use little communication.

This two-phase structure is called the *preprocessing model.* When focussing on optimizations to the online phase, it is also common to abstract out the preprocessing phase by assuming that *somehow*, the parties received random bits sampled with some pre-defined correlations

running, without collecting any private data.

(random OTs are a particular example of that, but other correlations are possible) from some external trusted dealer (this is to abstract out the fact that this correlated randomness distribution was done securely), and use this material in the online phase. This gives rise to an information-theoretic model of secure computation given access to a trusted source of correlated random coins, which is called the *correlated randomness model*.

2.6.2 Beaver's protocol

We now describe Beaver's derandomization protocol. Assume that, ahead of time, Alice and Bob have obtained the result of an oblivious transfer executed on random inputs: Alice (playing the role of the sender) has two random bits (r_0, r_1), and Bob received (σ, r_σ), where σ is a random selection bit. Once Alice receive her two OT inputs (s_0, s_1), and Bob receives his selection bit b, the protocol proceeds as follows:

- Bob tells Alice whether $\sigma = b$. This is done by sending a single bit $\ell = \sigma \oplus b$. Note that since σ is random and unknown to Alice, this leaks no information about b.

- If $\sigma = b$ $(\ell = 0)$, Alice sends $(s_0 \oplus r_0, s_1 \oplus r_1)$ to Bob; else, she sends $(s_0 \oplus r_1, s_1 \oplus r_0)$ (that is, Alice sends $(u_0, u_1) = (s_0 \oplus r_\ell, s_1 \oplus r_{1 \oplus \ell})$ to Bob). Observe that in any case, Bob (who knows only r_σ) will only be able to mask one of (s_0, s_1), the one masked by r_σ – which is exactly s_b by construction (since $u_b = s_b \oplus r_{b \oplus \ell} = s_b \oplus r_\sigma$).

It is a simple exercise to check that this protocol is correct and perfectly secure, and we leave it to the interested reader.

2.6.3 Secure computation with correlated randomness

One way to abstract out Beaver's result is as follows: if the parties are given access to a *trusted* source of $2n$ random oblivious transfer, then they can securely compute any boolean circuit with n AND gates (in the honest-but-curious setting), with *perfect* (information-theoretic) security. This stands in stark contrast with secure computation in the plain model (*i.e.* without any trusted setup taking place before the execution of the protocol): in the plain model, it is well-known that secure two-party computation is *impossible* without assuming computationally bounded parties (see the discussion in Section 2.1.2).

The random oblivious transfers form a special case of a source of *correlated randomness*: each random OT $(s_0, s_1), (b, s_b)$ is exactly a uniformly random 4-tuple 4-tuple (s_0, s_1, b, α) of bits sampled conditioned on

$$\alpha = b \cdot (s_0 \oplus s_1) \oplus s_0. \tag{2.1}$$

In other words, (s_0, s_1) and (b, α) are uniformly random pairs of bits sampled conditioned on satisfying the degree-2 correlation given by equation 2.1.

Definition 2.6.1 (Correlated randomness model)**.** *We call secure computation in the* correlated randomness model *a model of information-theoretic secure computation where before the start of the protocol, all participants securely receive random correlated strings from a trusted dealer, of a predefined length and for a predefined correlation.*

Using the terminology above, Beaver's result establishes that information-theoretic 2-party computation is possible in the correlated randomness model:

Theorem 2.6.2 (Beaver + GMW, informal)**.** *For any boolean circuit C with $2n$ AND-gates and m output gates, there exists an information-theoretic two-party computation protocol that securely computes C with semi-honest security in the correlated randomness model, with the following features:*

- *The two players request $2n$ random oblivious transfers from the trusted dealer.*
- *The protocol communicates $4n + 2m$ bits in total.*

While the above theorem is merely a restatement of Theorem 2.2.1 coupled with Beaver's derandomization technique, it highlights an important concept: in secure computation with correlated randomness, we typically distinguish the *amount* of resources consumed by the protocol (*i.e.* the length of the correlated randomness) and the *type* of correlated randomness (above, random OTs). Our focus so far has been on two-party secure computation of boolean circuits in the honest-but-curious setting. However, as we will see in the next section, many other important settings exist: there can be more than two parties, one might want to achieve security against malicious participants, and the function that the parties want to evaluate might not have a nice representation as a boolean circuit – for example, it can be more suitably represented by an arithmetic circuit over a larger field.

For each of these variants, efficient information-theoretic protocols in the correlated randomness model exist, but the *type* of correlated randomness that they consume differs. Looking ahead, this suggests that the fundamental question we raised in Section 2.3.2 (How can we generate efficiently a very large number of oblivious transfers?) should be generalized to the following question:

How can we generate efficiently a large amount of suitable correlated randomness?

Above, "suitable" refers to any type of correlated randomness that the setting at hand might call for. As one would expect, the more one wants to handle complex types of functions or achieve advanced features (e.g. strong security, or handling many participants), the more complex the necessary correlated randomness becomes. Furthermore, *specific applications* of secure computation can often benefit from tailored types of correlated randomness, chosen to optimize the application. The study and design of efficient methods to generate different types of correlated randomness, motivated by different applications in secure computation, is a rich and diverse field. This manuscript will cover some of the ways this field has been explored in my work.

2.7 Extensions of GMW

In this section, we introduce extensions of the GMW protocol to the setting of more than two parties, and for general arithmetic circuits. The seminal GMW protocol of Goldreich, Micali, and Wigderson [GMW87a], was originally formulated for N parties directly; the extension to arithmetic circuits is straightforward.

Until now, we avoided formal definitions in favor of more intuitive descriptions, to prevent harming readability. However, as we delve further into slightly more complex (or notation-heavy) variants of secure computation, the intuitive descriptions become increasingly less actionable, and introducing more rigor and formalism helps in introducing more complex notions without disrupting too much the flow of the discourse. Therefore, we will introduce, when needed, more formal definitions for the objects we manipulate.

Definition 2.7.1. *An arithmetic circuit C with n inputs $(x_1, \cdots, x_n)$ over a finite field $(\mathbb{F}, +, \times)$ is a directed acyclic graph of indegree 2 whose outdegree-0 nodes are called* output gates*. Each internal node (all nodes except indegree-0 nodes) is labeled with either $+$ or $\times$, representing the evaluation of either of the field operations. There are two types of indegree-0 nodes:* input nodes *are labeled with a variable x_i for $i \leq n$, and* constant nodes *are labeled with a value $c \in \mathbb{F}$. Evaluating C on inputs $(x_1, \cdots, x_n) \in \mathbb{F}^n$ is done by inductively writing on each output wire of a gate the result of the gate operation applied to the value written on the input wires, once it has been defined. The output of the evaluation is the tuple of values written on the output wires.*

2.7.1 N-party MPC of arithmetic circuits, honest-but-curious setting

From now on, we will view GMW as an information-theoretic protocol in the correlated randomness model of Beaver [Bea92a; Bea95]. We now define the notion of a Beaver triple (sometimes also called multiplication triple):

Definition 2.7.2. *An N-party Beaver triple over a field $\mathbb{F}$ is obtained by sampling $(u, v) \xleftarrow{\$} \mathbb{F}^2$ and distributing uniformly random additive shares of $(u, v, u \cdot v)$ between the N parties. Equivalently, each party $(P_1, \cdots, P_{N-1})$ receives a uniformly random 3-tuple (u_i, v_i, w_i), and P_N receives $(u - \sum_{i<N} u_i, v - \sum_{i<N} v_i, u \cdot v - \sum_{i<N} w_i)$.*

For convenience, we will introduce the following shorthand: given $u \in \mathbb{F}$, the notation $\langle u \rangle$ means that the parties hold additive *shares* $(u_1, \cdots, u_N)$ of u. This means that each party P_i holds a value u_i, where the joint distribution of $(u_1, \cdots, u_N)$ is uniformly random conditioned on $u = \sum_{i=1}^N u_i$. Note that this implies that for any subset $S \subsetneq [N]$, no coalition $(P_i)_{i \in S}$ of $|S| < N$ parties can learn anything about u from their joint shares $(u_i)_{i \in S}$, yet u can be reconstructed given all N shares. With this notation, we write $(\langle u \rangle, \langle v \rangle, \langle u \cdot v \rangle)$ to denote a Beaver triple without explicitly specifying the share of each party. Then, if the parties are given shares $(\langle u \rangle, \langle v \rangle)$ of elements u, v and if $(\alpha, \beta) \in \mathbb{F}^2$ are public values, we write $\langle \alpha u + \beta v \rangle \leftarrow \alpha \cdot \langle u \rangle + \beta \cdot \langle v \rangle$ to indicate that the parties with shares $(u_i)_i$ and $(v_i)_i$ of u, v can locally (without interaction) compute shares $(\alpha u_i + \beta v_i)_i$ of $\alpha u + \beta v$ (in other words, the sharing function is linear). Eventually, we say that the parties *reconstruct* (or *open*) a shared value u when each party broadcasts their share u_i to every other party, and all parties compute $u \leftarrow \sum_{i=1}^N u_i$.

Beaver triples are the main type of correlated randomness that we will use to get information-theoretic N-party secure computation of arithmetic circuits over $\mathbb{F}$ in the honest-but-curious setting (which is a synonym of t-private). It is a standard exercise to see that given two instances of random oblivious transfers, one can get (without further interaction) a 2-party Beaver triple over $\mathbb{F}_2$. This can be generalized to show that a single N-party Beaver triple can be obtained by distributing two random OTs between each pair of participants – but this generalization requires communication.[8]

Theorem 2.7.3 (GMW + Beaver, generalized)**.** *There exists a $(N-1)$-private information-theoretically secure N-party protocol for computing any functionality $f : (\mathbb{F}^*)^N \mapsto (\mathbb{F}^*)^N$, in the correlated randomness model, with the following features:*

[8] The interested reader can try to show this by themself – this is not too hard – or jump to Section 2.7.2 for a protocol that gives essentially the solution.

- *If f is computed by an arithmetic circuit C with n $\times$-gates and m output nodes, the parties need n Beaver triples from the trusted dealer.*
- *The total communication of the protocol is $(n+m) \cdot N$ elements of $\mathbb{F}$.*

2.7.1.1 The protocol

Let $P_1, \cdots, P_N$ be N parties with respective inputs $(x_1, \cdots, x_N)$, and let C_f be an arithmetic circuit computing the function f. The parties will evaluate the circuit gate by gate, starting from the inputs and computing the value of a gate when the values of its two parent nodes (which are either input nodes or gates themselves) have been computed. The protocol maintains the following invariant: after evaluating a gate, each party P_i will hold an additive share v_i of the value v on this gate. Without loss of generality, we assume that the parties always hold shares of the inputs to a gate when evaluating it: if an input to the gate is an input node carrying a field element b_i belonging to party P_i, we define the shares of $(P_1, \cdots, P_i, \cdots, P_n)$ to be $(0, \cdots, b_i, \cdots, 0)$ (while this is technically not a random share, this is sufficient for our purpose).

- Initialization: before the protocol, all parties ask for $2B$ Beaver triples to the trusted dealer, where B is a bound on the number of AND gates in C_f.
- Evaluating a $+$-gate $+(a, b)$: the parties locally sum their shares of u and v. No communication is required.
- Evaluating a $\times$-gate $\times(a, b)$: the parties must compute additive shares of $u \cdot v$. This is done using the secure product protocol described below. This step consumes a Beaver triple.
- Output: after evaluating the output gate, all parties broadcast their share of the output, and all parties reconstruct the output.

2.7.1.2 Secure product

On input additive shares $\langle u \rangle, \langle v \rangle$ of two field elements a, b, the *secure product* protocol proceeds as follows: the parties

- retrieve a (not previously used) Beaver triple $(\langle u \rangle, \langle v \rangle, \langle u \cdot v \rangle)$,
- compute $\langle u + a \rangle \leftarrow \langle u \rangle + \langle a \rangle$ and $\langle v + b \rangle \leftarrow \langle v \rangle + \langle b \rangle$;
- reconstruct $z_u = u + a$ and $z_v = v + b$ (this step, which is the only one where the parties interact, does not leak any private information because u and v are uniformly random, therefore z_u and z_v perfectly mask a and b);
- compute $\langle a \cdot b \rangle \leftarrow z_u \cdot \langle b \rangle - z_v \cdot \langle u \rangle + \langle u \cdot v \rangle$.

2.7.1.3 Bottom line

In the N-party honest-but-curious setting, for general arithmetic circuits, Beaver triples form the core resource to enable fast information-theoretic secure computation. Concretely, it allows to implement the N-party secure product functionality, generalizing in a natural way the protocol we described for the 2-party secure product functionality over $\mathbb{F}_2$ in Section 2.3.1.1. Note that we used oblivious transfers in our description of the protocol of Section 2.3.1.1, but we could have equivalently used random OTs, using Beaver's derandomization method. Then, it is a simple exercise to check that the protocol instantiated with two instances of a random OT is syntactically equivalent to the secure product described above, using a 2-party Beaver triple over $\mathbb{F}_2$ (since two instances of a random OT immediately yield a 2-party $\mathbb{F}_2$-Beaver triple).

Similarly, as for the 2-party case, the generalized GMW protocol reduces the question of achieving *efficient* secure multiparty computation to the following question:

Is it possible to generate efficiently a large number of random Beaver triples?

I will cover in the next section what was the state of the art regarding this question when I started working on the subject.

2.7.2 Generating Beaver triples

Generating Beaver triple is more involved than generating oblivious transfers and, as we will see, the best solution can vary depending on the field $\mathbb{F}$ and the number N of parties.

2.7.2.1 Over $\mathbb{F}_2$, for small N

Over $\mathbb{F}_2$, generating an N-party Beaver triple reduces to executing $N \cdot (N-1)$ oblivious transfers (two OTs for each pair of parties). The reduction works as follows:

- Each party samples a random pair $(u_i, v_i) \xleftarrow{\$} \{0,1\}^2$ of bits. Collectively, these bits form shares $(\langle u\rangle, \langle v\rangle)$ of two values $u = \bigoplus_{i\leq N} u_i$ and $v = \bigoplus_{i\leq N} v_i$. To obtain a Beaver triple, it remains for the parties to construct shares of $u \cdot v$. Expanding the product yields:

  $$u \cdot v = \bigoplus_{i\neq j} u_i \cdot v_j \oplus \bigoplus_{i\leq N} u_i \cdot v_i.$$

  We treat the $u_i \cdot v_j$ with $i \neq j$ and the $u_i \cdot v_i$ separately since the latter is known to party P_i, while the former requires cross-party computation.

- Each pair of parties (P_i, P_j) constructs $(\langle u_i v_j\rangle, \langle u_j v_i\rangle)$ (where $\langle\cdot\rangle$ denote 2-party shares) as follows:
  - P_i picks a uniformly random value r, which will form their share of $u_i v_j$. Then, it remains for P_j to obtain the corresponding share $r \oplus u_i v_j$.
  - P_i and P_i execute an oblivious transfer protocol, where P_i plays the role of the sender with input $(r, r \oplus u_i)$, and P_j plays the role of the receiver with input v_j. P_j receives an output s; by definition of the OT, s is equal to r if $v_j = 0$, and to $r \oplus u_i$ if $v_j = 1$: in other words, $s = r \oplus u_i v_j$, as desired.
  - P_i and P_j proceed identically, using an OT, to obtain 2-party shares of $u_j v_i$.

- Each party P_i define their share of $u \cdot v$ to be $u_i v_i$ XORed with their shares of the $u_i v_j$ and the $u_j v_i$ for every $j \neq i$. Observe that

  $$\langle u \cdot v\rangle = \bigoplus_{i\neq j} \langle u_i \cdot v_j\rangle \oplus \bigoplus_{i\leq N} u_i \cdot v_i,$$

from which correctness follows.

Proving that the protocol satisfies $(N-1)$-privacy (Definition 2.1.4) is a standard exercise, and we leave it to the interested reader.

Lemma 2.7.4. *There is an N-party protocol that securely generates a random Beaver triple over $\mathbb{F}_2$ (in the honest-but-curious setting) using $N \cdot (N-1)$ invocations of an oblivious transfer, and no further communication between the parties.*

2.7.2.2 Over $\mathbb{F}$, for small N

The above solution has two clear downsides: (1) it is restricted to $\mathbb{F}_2$ and (2) its complexity scales quadratically with the number N of parties (and it is therefore best suited for small values of N). Here, we explain how to lift the first downside. To this end, we introduce the notion of *oblivious linear evaluation* (OLE).

Definition 2.7.5 (OLE, informal)**.** *An oblivious linear evaluation (OLE) over a field $\mathbb{F}$ is a two-party protocol between a sender with input $(u, v) \in \mathbb{F}^2$ and a receiver with input $x \in \mathbb{F}$. At the end of the protocol, the receiver should learn $u \cdot x + v$, and the sender does not get anything.*

The OLE functionality is summarized in Figure 2.3. The term oblivious linear evaluation stems from the following interpretation: in an OLE protocol, the sender holds the description of an affine function $f_{u,v} : x \mapsto ux + v$, and the receiver holds an input x. The goal of the protocol is to let the sender *obliviously* evaluate $f_{u,v}$ on x, revealing only the result of the evaluation to the receiver. The term "affine function evaluation" might seem a better choice since $f_{u,v}$ is not a linear function, but here "linear" should be seen as a property of the space of the functions $f_{u,v}$ (that is, $\lambda f_{u,v} + f_{u',v'} = f_{\lambda u+v, \lambda u'+v'}$) rather than a property of the functions f themselves (that is, $\lambda f(x) + f(y) = f(\lambda x + y)$)).

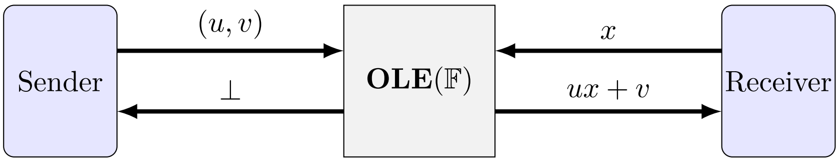


Figure 2.3: Ideal functionality of the oblivious linear evaluation over $\mathbb{F}$.

Similarly as for random OTs, we say that two parties receive a random OLE if they receive uniformly random (u, v) and (x, w) sampled over $\mathbb{F}$ conditioned on $w = ux + v$.

Remark 2.7.6. *OLEs are the natural generalization of OTs over larger fields. In fact, up to a minor syntactical difference, an OLE over $\mathbb{F}_2$, where the receiver gets $ux \oplus v$ is just an oblivious transfer between the sender inputs $(u, u \oplus v)$ using the receiver selection bit x. Therefore, any OLE over $\mathbb{F}_2$ can be locally mapped to an OT by the sender via the reversible mapping $(u, v) \mapsto (u, u \oplus v)$, and conversely.*

A useful variant of OLE, which we will rely upon later on in this manuscript, is the notion of vector-OLE:

Definition 2.7.7 (Vector-OLE, informal)**.** *A vector oblivious linear evaluation (vector-OLE, or VOLE) of size n over a field $\mathbb{F}$ is a two-party protocol between a sender with input vectors $(\vec{u}, \vec{v}) \in \mathbb{F}^n \times \mathbb{F}^n$ and a receiver with input $x \in \mathbb{F}$. At the end of the protocol, the receiver should learn $\vec{u} \cdot x + \vec{v}$, and the sender does not get anything.*

Vector-OLE have been introduced in [ADI+17]. They generalize the notion of oblivious transfers of large strings in the same way that OLE generalizes the notion of oblivious transfer between pairs of bits. A vector-OLE of size n can always be realized using n instances of an OLE over $\mathbb{F}$, by letting the receiver use the same input x accross all instances. However, vector-OLEs can often be realized more efficiently than n instances of OLE [ADI+17; AK23] and are a useful building block in many applications, which motivates defining them independently.

From OLEs to Beaver triples. Using $N \cdot (N-1)$ OLEs over $\mathbb{F}$, N parties can generate a Beaver triple over $\mathbb{F}$. The protocol is an immediate generalization of the OT-based protocol over $\mathbb{F}_2$:

- Each party samples a random pair $(u_i, v_i) \xleftarrow{\$} \mathbb{F}^2$. Collectively, these values form shares $(\langle u \rangle, \langle v \rangle)$ of $u = \sum_{i \leq N} u_i$ and $v = \sum_{i \leq N} v_i$. It remains to construct shares of $u \cdot v$. We have:
$$u \cdot v = \sum_{i \neq j} u_i \cdot v_j + \sum_{i \leq N} u_i \cdot v_i.$$
- Each pair of parties (P_i, P_j) constructs $(\langle u_i v_j \rangle, \langle u_j v_i \rangle)$ (where $\langle \cdot \rangle$ denote 2-party shares) as follows:
  - P_i picks a uniformly random value r, which will form their share of $u_i v_j$. It remains for P_j to obtain the corresponding share $r + u_i v_j$: this is done by executing an OLE with input (r, u_i) from the sender, and input v_j from the receiver.
  - P_i and P_j proceed identically, using an OLE, to obtain 2-party shares of $u_j v_i$.
- Each party P_i define their share of $u \cdot v$ to be $u_i v_i$ summed with their shares of the $u_i v_j$ and the $u_j v_i$ for every $j \neq i$. As before,
$$\langle u \cdot v \rangle = \sum_{i \neq j} \langle u_i \cdot v_j \rangle + \sum_{i \leq N} u_i \cdot v_i,$$
from which correctness follows.

Security follows from the same observations as in the case of $\mathbb{F}_2$.

Lemma 2.7.8. *There is an N-party protocol that securely generates a random Beaver triple over $\mathbb{F}$ (in the honest-but-curious setting) using $N \cdot (N-1)$ invocations of an oblivious linear evaluation, and no further communication between the parties.*

OLEs from OTs. We now show that an OLE over a field $\mathbb{F}$ can be obtained using $\log |\mathbb{F}|$ invocations of an OT. This protocol was first described in [Gil99]. Let $(u, v) \in \mathbb{F}^2$ be the sender inputs, and let $x \in \mathbb{F}$ be the receiver input. Assume for simplicity that $\mathbb{F}$ is a prime-order field (and can therefore be identified with the set of integers modulo $|\mathbb{F}|$) – the protocol can be easily generalized to work with general finite fields. We denote by $(x_0, \cdots, x_{t-1})$ the bits of the binary representation of x, where $y = \log |\mathbb{F}|$. The protocol proceeds as follows:

- The sender samples uniformly random shares $(s_0, \cdots, s_{t-1})$ of v over $\mathbb{F}$ (that is, the s_i are random conditioned on $\sum_i s_i = v$).
- The sender and the receiver execute t parallel instances of an oblivious transfer, where in the i-th instance (from 0 to $t-1$), the sender inputs $(s_i, s_i + 2^i \cdot u)$, and the receiver inputs x_i. Observe that the receiver obtains $s_i + x_i \cdot 2^i \cdot u$.
- The receiver outputs $\sum_{i=0}^{t-1} s_i + x_i \cdot 2^i \cdot u$.

Correctness is straightforward:

$$\sum_{i=0}^{t-1} s_i + x_i \cdot 2^i = \sum_{i=0}^{t-1} s_i + \left(\sum_{i=0}^{t-1} x_i \cdot 2^i\right) \cdot u = v + x \cdot u.$$

Security follows also easily from the security of the OTs and the fact that the s_i are uniformly random shares of v. Combining this protocol with the OLE-based protocol for generating Beaver triples, we get:

Lemma 2.7.9. *There is an N-party protocol that securely generates a random Beaver triple over $\mathbb{F}$ (in the honest-but-curious setting) using $N \cdot (N-1) \cdot \log |\mathbb{F}|$ invocations of an oblivious transfer, and no further communication between the parties.*

Bottom line. It is possible to securely generate Beaver triples over arbitrary fields, and for any number of parties, using only invocations of oblivious transfer protocols. However, the complexity of this approach scales with $\Omega(N^2 \cdot \log |\mathbb{F}|)$. Yet, for not-too-large values of N and $\log \mathbb{F}$, this remains the most computationally efficient way of generating Beaver triples, due to the existence of super-fast OT extension protocols. However, communication-wise, this becomes very quickly prohibitive: assuming 130 bits of communication per OT, if there are $N = 10$ parties and the field $\mathbb{F}$ is chosen to be $\mathbb{F}_{2^{64}}$ (a common choice since it matches the size of machine words), this yields already about 750.000 bits of communication *for each Beaver triple*. When evaluating an arithmetic circuit containing as little as ten million ×-gates (which is quite small), this already represents 800 Gigabytes of communication. Below, we will see that there is a better solution when the $\Omega(N^2 \cdot \log |\mathbb{F}|)$ becomes prohibitive.

2.7.2.3 Over $\mathbb{F}$, for large N

When the size of the field $\mathbb{F}$ grows too much, an alternative solution to the above protocol is to rely on threshold homomorphic encryption. This approach to secure computation dates back to [CDN01]. The technique has multiple flavors, using different types of homomorphic encryption. The approach I will present below, using the threshold additively homomorphic encryption notion, is somewhat folklore. Depending on the context, other variants might perform better, but my main purpose is to illustrate this type of method. These approaches use a type of cryptographic machinery quite different from the ones we used so far (albeit perhaps closer to the type of primitives one sees in a standard cryptography curriculum).

Definitions. To introduce the construction, we need to recall the notion of public-key encryption over a field $\mathbb{F}$ first:

Definition 2.7.10 (Public-Key Encryption)**.** *A public-key encryption scheme Π is a triple of polytime algorithms* $(\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{Dec})$*, such that*

- $\Pi.\mathsf{KeyGen}(1^\lambda)$, *generates a pair* $(\mathsf{pk}, \mathsf{sk})$ *of public and private keys. The key* pk *specifies the ciphertext space* $\mathcal{C}$ *and the random source* $\mathcal{R}$.

- $\Pi.\mathsf{Enc}(\mathsf{pk}, m; r)$, *given* $m \in \mathbb{F}$ *and random coins* $r \in \mathcal{R}$, *outputs a ciphertext* c.

- $\Pi.\mathsf{Dec}(\mathsf{sk}, c)$, *outputs a message* $m \in \mathbb{F}$.

In addition Π *must satisfy the correctness and* $\mathsf{IND\text{-}CPA}$ *security properties defined below.*

Definition 2.7.11 (Correctness)**.** Π *is* correct *if for any pair* $(\mathsf{pk}, \mathsf{sk}) \overset{\$}{\leftarrow} \Pi.\mathsf{KeyGen}(1^\lambda)$, *message* $m \in \mathbb{F}$, *and random coin* $r \in \mathcal{R}$, *it holds that* $\Pi.\mathsf{Dec}(\mathsf{sk}, \Pi.\mathsf{Enc}(\mathsf{pk}, m; r)) = m$.

The experiments $\mathsf{Exp}_{\mathscr{A}}^{\texttt{ind-cpa}-0}(1^\lambda)$ and $\mathsf{Exp}_{\mathscr{A}}^{\texttt{ind-cpa}-1}(1^\lambda)$ for the $\mathsf{IND\text{-}CPA}$ property of the encryption scheme Π are represented on Figure 2.4.

```
$\mathsf{Exp}_{\mathscr{A}}^{\texttt{ind-cpa}-0}(1^\lambda)$ :
    $(\mathsf{pk}, \mathsf{sk}) \overset{\$}{\leftarrow} \Pi.\mathsf{KeyGen}(1^\lambda)$
    $(m_0, m_1, \mathsf{st}) \overset{\$}{\leftarrow} \mathscr{A}(\mathsf{pk})$
    $r \overset{\$}{\leftarrow} \mathcal{R}$
    $c \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk}, m_0; r)$
    $b \leftarrow \mathscr{A}(\mathsf{pk}, c, \mathsf{st})$
```

```
$\mathsf{Exp}_{\mathscr{A}}^{\texttt{ind-cpa}-1}(1^\lambda)$ :
    $(\mathsf{pk}, \mathsf{sk}) \overset{\$}{\leftarrow} \Pi.\mathsf{KeyGen}(1^\lambda)$
    $(m_0, m_1, \mathsf{st}) \overset{\$}{\leftarrow} \mathscr{A}(\mathsf{pk})$
    $r \overset{\$}{\leftarrow} \mathcal{R}$
    $c \leftarrow \Pi.\mathsf{Enc}(\mathsf{pk}, m_1; r)$
    $b \leftarrow \mathscr{A}(\mathsf{pk}, c, \mathsf{st})$
```

Figure 2.4: Experiments for the $\mathsf{IND\text{-}CPA}$ property of the encryption scheme Π

Definition 2.7.12 ($\mathsf{IND\text{-}CPA}$ Security)**.** Π *is* $\mathsf{IND\text{-}CPA}$ secure *if for any polytime adversary* $\mathscr{A}$, *it holds that* $|\Pr[\mathsf{Exp}_{\mathscr{A}}^{\textit{ind-cpa}-1}(1^\lambda) = 1] - \Pr[\mathsf{Exp}_{\mathscr{A}}^{\textit{ind-cpa}-0}(1^\lambda) = 1]| = \mathsf{negl}(\lambda)$, *where* negl *denotes a negligible function.*

An N-party threshold public-key encryption scheme is an encryption scheme where the decryption can be distributed among N parties holding shares sk_i of the secret key sk. Here, for simplicity, we will focus on a notion of additive threshold decryption, where running the partial decryption algorithm with sk_i yields an additive share, over $\mathbb{F}$, of the message m. The definition below is adapted to our setting:

Definition 2.7.13 (Threshold Public-Key Encryption, informal)**.** *An* N*-party threshold public-key encryption scheme* Π *is a PKE* $(\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{Dec})$ *satisfying correctness and* $\mathsf{IND\text{-}CPA}$ *security, as in Definition 2.7.11 and 2.7.12 respectively, together with additional algorithms* $(\Pi.\mathsf{Share}, \Pi.\mathsf{Rec})$, *and a distributed decryption protocol* Π_{dec}*:*

- $\Pi.\mathsf{Share}(\mathsf{sk})$, *generates* N *secret key shares* $(\mathsf{sk}_1, \cdots, \mathsf{sk}_N)$

- $\Pi.\mathsf{Rec}(\mathsf{sk}_1, \cdots, \mathsf{sk}_N)$, *reconstructs the secret key* sk

- Π_{dec} *is an* N*-party protocol secure in the semi-honest model where all parties hold a ciphertext* c *as common input, and each party holds a share* sk_i *of* sk. *The protocol securely compute the functionality* $\Pi.\mathsf{Dec}(\Pi.\mathsf{Rec}(\mathsf{sk}_1, \cdots, \mathsf{sk}_N), c)$.

In addition, the sharing algorithm must satisfy the standard property that for any sk, *the distribution of any strict subset of the keys* $(\mathsf{sk}_i)_{i \in S}$ *with* $S \subsetneq [N]$ *can be statistically simulated without* sk.

Eventually, we will want our threshold public-key encryption scheme to be *additively homomorphic*. We sketch one possible definition below;

Definition 2.7.14 (Additively Homomorphic Encryption, informal)**.** *A PKE* $(\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Enc}, \Pi.\mathsf{Dec})$ *is an additively homomorphic encryption scheme if there is an efficient randomized algorithm* $\Pi.\mathsf{Add}$ *where, for any* $\alpha \in \mathbb{F}$ *and any pair* (c_0, c_1) *of ciphertexts,* $\Pi.\mathsf{Add}(\alpha, c_0, c_1)$ *outputs a new ciphertext* c*, such that the following holds:*

- *(homomorphic correctness) If* $\Pi.\mathsf{Dec}(\mathsf{sk}, c_0) = m_0$ *and* $\Pi.\mathsf{Dec}(\mathsf{sk}, c_1) = m_1$*, then* $\Pi.\mathsf{Dec}(\mathsf{sk}, c) = \alpha \cdot m_0 + m_1$*.*
- *(circuit privacy) The output distribution of* $\Pi.\mathsf{Add}$ *is statistically independent of* α*.*

We also write $\Pi.\mathsf{Add}(\alpha, c_0, \beta)$ *with* $\beta \in \mathbb{F}$ *to denote the homomorphic computation of* $\alpha m_0 + \beta$ *(this can always be done by computing an arbitrary encryption of* β *first).*

There are several ways to instantiate an N-party threshold additively homomorphic encryption scheme. If we replace the field $\mathbb{F}$ by an integer ring $\mathbb{Z}_n$ (where n is a product of large primes), a common choice is to rely on (threshold variants of) the Paillier cryptosystem [Pai99]. Otherwise, over arbitrary prime-order fields $\mathbb{F}$, one can rely on threshold variants of lattice schemes such as the Regev cryptosystem [Reg05]. All these works admit efficient threshold variants where the distributed decryption protocol has a communication comparable to exchanging $O(n)$ ciphertexts in a constant number of rounds.

Beaver triples from threshold additively homomorphic encryption. Equipped with the above definition, we can sketch a simple protocol to generate a Beaver triple. Fix N parties $P_1, \cdots, P_N$ and assume that, in a prior setup phase, they distributively generated (or received from a trusted dealer) a public key pk as well as shares $\mathsf{sk}_1, \cdots, \mathsf{sk}_N$ of the corresponding secret key (each party holds one share) for a threshold additively homomorphic encryption scheme Π. The protocol proceeds as follows:

- Each party P_i samples $(x_i, y_i) \xleftarrow{\$} \mathbb{F}^2$ and broadcasts $c_i \xleftarrow{\$} \Pi.\mathsf{Enc}(\mathsf{pk}, x_i)$.
- All parties publicly compute using $\Pi.\mathsf{Add}$ a ciphertext c encrypting the sum $x = \sum_{i=1}^{N} x_i$.
- Each party P_i broadcasts $c'_i \xleftarrow{\$} \Pi.\mathsf{Add}(y_i, c_i, z_i)$.
- All parties publicly compute using $\Pi.\mathsf{Add}$ a ciphertext c' encrypting the sum
$$\sum_{i=1}^{N} \Pi.\mathsf{Dec}(\mathsf{sk}, c'_i) = \sum_i y_i \cdot x + z_i = x \cdot y + \sum_i z_i.$$
- All parties execute the distributed decryption protocol Π_{dec} to obtain $z = x \cdot y + \sum_i z_i$.
- Each party P_i outputs $(x_i, y_i, s_i = -z_i)$, except party P_N who outputs $(x_N, y_N, s_N = z - z_N)$.

Security follows from the $\mathsf{IND\text{-}CPA}$ security of the scheme, together with the fact that the value $z = x \cdot y + \sum_i z_i$ is perfectly masked to any subset of at most $N - 1$ parties. Correctness can be checked routinely:

$$\sum_{i=1}^{N} s_i = z - \sum_{i=1}^{N} z_i = x \cdot y = \left(\sum_{i=1}^{N} x_i\right) \cdot \left(\sum_{i=1}^{N} y_i\right).$$

Communication-wise, the cost of this protocol is dominated by the communication of $2N$ ciphertexts, plus the cost of the distributed decryption protocol (typically comparable to N ciphertexts). Using the Paillier encryption scheme, one ciphertext has a size comparable to two elements of $\mathbb{Z}_n$. Furthermore, when generating many Beaver triples in parallel using a lattice-based encryption scheme, one can rely on "packing" techniques to achieve an amortized communication of $O(N)$ elements of $\mathbb{F}$ per Beaver triple. Overall, the cost of generating Beaver triples with these methods typically scales as $O(N)$, which is a factor $N \cdot \log |\mathbb{F}|$ smaller than the OT-based method of Section 2.7.2.2. This comes at the cost of using public key homomorphic encryption, and $O(N)$ public key operation per party for *each* Beaver triple (while using IKNP-style OT extension, the protocol requires $O(N)$ public key operation per party *in total*).

Historical notes. Oblivious linear evaluation was originally introduced in the work of Naor and Pinkas [NP99a]. Using threshold additively homomorphic encryption in secure computation was originally introduced in [CDN01] (from now on, CDN). Though it was initially described as a method to securely compute multiplication gates, its use for generating Beaver triples for MPC in the correlated randomness model is generally deemed folklore. The idea of using a CDN-style approach in the preprocessing phase was first introduced in [BDOZ11]. Following the seminal SPDZ paper [DPSZ12], later works [KPR18; GLM22] have relied mostly on a variant where the Beaver triples are generated using somewhat homomorphic encryption instead of threshold additive encryption. The CDN approach has recently got some renewed attention [BDO23] in the context of the recent YOSO framework for massively large-scale secure computation.

2.7.3 N-party MPC of arithmetic circuits, malicious setting

All protocols described so far have been restricted to the honest-but-curious setting, meaning that they are only guaranteed to be secure if the participants follow the specifications of the protocol. A much more challenging goal is that of secure computation in the malicious setting, where participants might behave arbitrarily. This is a fundamental topic, but providing even a superficial coverage of the techniques and challenges involved would take us too far. Hence, before ending this chapter, I will limit myself to making two short remarks about malicious security:

- Most maliciously secure protocols proceed by first designing a protocol in the honest-but-curious model, and then designing mechanisms (relying on cryptographic primitives such as *commitments* and *zero-knowledge proofs*) to force the participants to adhere to the specifications (or, equivalently, guaranteeing that any attempt to deviate from the prescribed strategy will be detected by the participants). This is the reason why addressing the design of secure computation protocols in the honest-but-curious setting remains well-motivated even if one finds the assumption of honest behavior to be undesirable.

- Maliciously secure MPC protocols in the preprocessing model typically rely on a variant of Beaver triples called *authenticated Beaver triple*, where in addition to shares $(\langle a \rangle, \langle b \rangle, \langle ab \rangle)$ for random $a, b \in \mathbb{F}$, the parties also get $(\langle \Delta \cdot a \rangle, \langle \Delta \cdot b \rangle, \langle \Delta \cdot ab \rangle)$ where Δ is a global authentication value (typically from $\mathbb{F}$ or some extension field thereof) shared between the parties. Roughly, the purpose of Δ will be to check that the parties have honestly computed some target linear relation, by computing the same linear relation on the authenticated values and checking afterward an equation of the form $\Delta \cdot L(\vec{x}) =_? L(\Delta \cdot \vec{x})$. All the methods described in this section can be adapted relatively easily to generate authenticated Beaver triples instead.

With this, I conclude this chapter, which I hope conveyed some intuition about the motivations for using correlated randomness in secure computation and some of the challenges associated with generating this correlated randomness. The next chapter will focus on an alternative approach to correlated randomness generation which I have developed with my coauthors over the past six years, and whose aim is to enable the generation of a large amount of correlated randomness using a minimal amount of communication.

Chapter 3

Silent Secure Computation

In this chapter, I cover one of the main themes of my research on secure computation. As the previous chapter made clear, or so I hope, secure computation *a la GMW* in the preprocessing model is one of the most promising paths towards truly efficient MPC protocols. With the combination of IKNP-style OT extensions and preprocessing of the OTs, it exhibits sufficient performances, from a computational point of view, in many real-world applications (including allowing our two lovebirds, Alice and Bob, to discover whether there is a mutual romantic interest – but also including, say, securely running statistical analyses on the joint private data of the patients of several hospitals to evaluate the efficiency of a new medication without compromising the patient's privacy). However, its *communication overhead* – a few hundred bits per AND gate – is prohibitive for most applications, especially in a WAN setting.

The results and techniques that I will introduce in this chapter form the foundations of the *silent preprocessing model* of secure computation. In this model, which I introduced in 2018 with my coauthors, the entire preprocessing phase boils down to a short interactive phase, with little communication and computation, followed solely by local computations. In the same way that OT extension confines all expensive operations to a one-time generation of a small number of base OTs, silent preprocessing confines the entire *communication* of the preprocessing phase to a one-time small interaction.

I asked Dall-E to add a mask to Alice and Bob, to indicate that they compute silently

Contents

Historical notes. The results and techniques covered in this section span multiple papers. In what follows, "we" refers to myself and my coauthors (which might differ between articles). Naming them for each article listed below would harm readability, but for proper credit: the results which I cover in the chapter have been obtained through collaborations with Maxime Bombar, Elette Boyle, Dung Bui (•), Alain Couvreur, Clément Ducros (•), Niv Gilboa, Yuval Ishai, Lisa Kohl, Pierre Meyer (•), Michele Orrù, Alain Passelègue, Srinivasan Raghuraman, Nicolas Resch, Mahshid Riahinia, Peter Rindal, Peter Scholl, and Maryam Zarezadeh. In this list, • denotes the Ph.D. students whom I have supervised.

We initially introduced the notion of silent secure computation (under the name of *cryptographic capsules*) in a CCS'2017 paper [BCG+17]. We achieved the first concretely efficient silent preprocessing protocol, for a specific correlation, a year later in [BCGI18] (CCS'2018). The real breakthrough, achieving concretely efficient silent preprocessing for precomputing oblivious transfers, came in our follow-up work at CRYPTO'2019 [BCG+19b], which we further optimized at CCS'2019 [BCG+19a]. A year later, we generalized silent preprocessing to the OLE correlation over large fields [BCG+20b] (CRYPTO'2020) and introduced the notion of pseudorandom correlation function (PCF) as a mean to achieve *unbounded, on-demand* silent preprocessing [BCG+20a] (FOCS'2020) (we used the techniques developed in this paper to obtain further results in [BCG+21b] (CRYPTO'21), albeit outside of the context of silent secure computation). Our paper at EUROCRYPT'21 [CM21] showed how to use the results of our CRYPTO'19 paper to achieve low-communication secure computation (a line of work that will be covered in the next chapter). We improved the efficiency of our CRYPTO'19 paper in [CRR21a] (CRYPTO'2021) and of both our CRYPTO'19 and FOCS'20 paper in [BCG+22] (CRYPTO'2022). Eventually, we further refined the results of our FOCS'20 paper in [CD23] (PKC'2023), introduced the notion of *precomputable* PCFs (a strengthening of PCFs which allows generating preprocessing material even before knowing the identity of the other participants) in [CMPR23] (EUROCRYPT'2023), and obtained silent preprocessing for the OLE correlation over small fields in [BCCD23], improving over the result of our CRYPTO'20 paper.

3.1 Introduction

The main conceptual message of Chapter 2 is that the task of designing efficient protocols for securely computing a function reduces to the problem of securely and efficiently distributing long correlated random strings among the parties, where the type of correlation is tied to the target functionality and security guarantees. We have seen that for *random oblivious transfers*, using OT extension allows generating an arbitrary amount of correlation with an amortized computational cost of three hash evaluations per OT, and a few hundred bits of communication per OT. Since even simple functions can require a huge number of OTs, the amount of communication forms the core bottleneck of this protocol – computation-wise, it is remarkably efficient[1]. However, the large communication is a big hurdle that severely limits the practicality of these methods.

For Beaver triples, the situation is even less satisfying: one has to choose between a fast protocol using IKNP-style OT extension, at the cost of the (often prohibitive) $\Omega(N^2 \log|\mathbb{F}|)$ of these approaches, or give up on OT extension techniques altogether and use much slower solutions (using public key cryptography instead of symmetric cryptography) based on additively homomorphic encryption.

3.2 Secure Computation with Silent Preprocessing from PCGs

So far, we have seen that secure computation with preprocessing proceeds in two phases:

1. In the preprocessing phase, the parties generate correlated randomness (*e.g.* random OTs, Beaver triples, OLEs...), using a protocol of their choice (typically from oblivious transfer or homomorphic encryption).

2. In the online phase, the parties *consume* this correlated randomness using an appropriate protocol, *e.g.* GMW.

The communication of the preprocessing phase dominates the total communication by a large factor: even in the two-party setting for semi-honest secure computation of boolean circuits using GMW, one needs a few hundred bits per AND gate of the circuit in the preprocessing phase, and only 3 bits per AND gate in the online phase, using Beaver's protocol (Section 2.6.2). For more parties, over arithmetic circuits, or in the malicious setting, the gap grows further.

3.2.1 The core insight: pseudorandomness is enough

Secure computation with preprocessing pays a huge communication cost to distributively generate long, *truly random* correlated string. Below, in a big, thick, red box, I outline the core insight at the heart of the new approach I am going to describe. If you take one message from this section:

[1] In fact, it is possible to instantiate the hash function using a simple tweak on the AES block cipher, which is the universally used standard cipher. Because it is so ubiquitously used, a set of hardware instructions for AES, the AES-NI instruction set, has even been included on the recent Intel processors. When using hardware instructions to implement calls to AES, the efficiency becomes incredible – about 1.3 clock cycles on a single processor to encrypt a Byte of data. On such processors, OT extension is insanely fast.

The long correlated random strings do not have to be truly random: they only have to *look* random

Here, by *look random*, I mean the following: it should be infeasible, from the viewpoint of the participants (modeled as polynomial-time algorithms) to distinguish the strings received by the other participants from truly random strings (correlated in the right way with their string). The cryptographic terminology to denote this type of “random-lookingness” is *pseudorandomness*.

3.2.1.1 Pseudorandom generators

To make it a bit more concrete, let us look at the definition of a pseudorandom generator:

Definition 3.2.1. *A pseudorandom generator (PRG) is a polynomial-time algorithm* $G : \{0,1\}^\lambda \mapsto \{0,1\}^m$*, with* $m \gg \lambda$*, such that for any polynomial-time distinguisher* $\mathcal{A}$*, it holds that*

$$|\Pr[\mathcal{A}(y) = 0 \mid x \xleftarrow{\$} \{0,1\}^\lambda, y \leftarrow G(x)] - \Pr[\mathcal{A}(y) = 0 \mid y \xleftarrow{\$} \{0,1\}^m]| \leq \mu(\lambda),$$

where μ *is a negligible function.*

A negligible function is any function that converges towards 0 faster than any inverse polynomial (*e.g.* $1/2^\lambda$, $\lambda^{-\log\lambda}$...). Concretely, when the probabilities of two events are negligibly close (their absolute difference is a negligible function), this corresponds to saying that the events cannot be distinguished by any polynomial-time adversary. Hence, the above says that no polynomial-time algorithm $\mathcal{A}$ can behave differently (output something different – *i.e.* notice a difference) in the two following scenarios:

1. It receives the value $y = G(x)$ on a uniformly random *short* input $x \xleftarrow{\$} \{0,1\}^\lambda$, or
2. It receives a uniformly random *long* value $y \xleftarrow{\$} \{0,1\}^m$.

Note that since x is n-bit long, $y = G(x)$ cannot have more than $\lambda \ll m$ bits of entropy: thus, y is the first scenario is very far from random (it was deterministically computed from a short input), yet all polytime algorithms will still fail to distinguish them from uniformly random inputs.

3.2.1.2 From one-time pads to stream ciphers

Let us take a short break from secure computation and have a quick look at the easier (and more well-known) task of *secure communication*: Alice wants to transmit a message $M \in \{0,1\}^m$ to Bob over a public channel while hiding M from a potential eavesdropper. The reader has probably heard of the one-time pad construction of Vernam: assume Alice and Bob share a pre-established uniformly random key $K \xleftarrow{\$} \{0,1\}^m$ of the same length as M. Then Alice can send $C = M \oplus K$ to Bob: this lets Bob retrieve M as $M \leftarrow C \oplus K$, but to any external observer, since K is uniformly distributed, so is C (and in particular, it carries no information about M). Of course, for long messages, pre-establishing (securely) a uniformly random key K of the same length is impractical. Unfortunately, by a celebrated result of Shannon, this protocol is the best possible: there cannot be any information-theoretically secure protocol using a key K shorter than M.

However, if we do not insist on *information-theoretic* (*i.e.* perfect) security, but rather are happy to settle for security against polynomial-time adversaries, there is a more efficient variant: Alice and Bob only need to pre-share a *short* random key $k \in \{0,1\}^\lambda$, and locally construct a "fake long random key" $K = G(k) \in \{0,1\}^m$ to use in the above protocol. Of course, K is not truly random, but no polytime adversary can distinguish it from random! We leave it to the interested reader – it is a standard exercise – to show that if an adversary could guess with good probability whether, say, M is the all-zero message or the all-one message (assuming M is initially picked to be either of those at random) from the transcript $C = M \oplus G(k)$ of the interaction, then one could use this adversary to construct an algorithm $\mathcal{A}$ contradicting the assumption that G is a pseudorandom generator.

This PRG-based construction of a secure communication protocol is essentially what is known as a *stream cipher*. To put it in perspective with our real target (which is secure computation), one could reformulate the one-time pad as a kind of protocol in the preprocessing model:

1. In the preprocessing phase, the two parties use a protocol to securely generate a long shared key K (this is called a *key exchange* protocol), and

2. In the online phase, the parties use the (information-theoretic) one-time pad protocol to exchange M using their shared key K.

Of course, one expects the preprocessing phase of this protocol to dominate the overall cost. But by using a PRG, one gets a much nicer, three-phase protocol:

1. In the preprocessing phase, the two parties use a protocol to securely generate a *short* shared key k.

2. In an offline phase, without any interactions, the parties locally compute the long key $K = G(k)$.

3. In the online phase, the parties use the one-time pad protocol to exchange M using their shared key K.

Observe that, above, the (expensive) preprocessing phase has been replaced by two phases: a much shorter preprocessing phase (to generate a short key k), followed solely by offline computation (which does not require any communication between Alice and Bob).

3.2.1.3 From MPC with correlated pseudorandomness to pseudorandom correlation generators

In one sentence, the goal of the silent preprocessing model is to achieve the same three-phase structure as the protocol above, but for the much harder task of secure computation. In the case of secure communication, the "correlated randomness" is as simple as it gets: Alice and Bob should just receive *the same string.* Hence, any PRG producing random-looking strings suffices. In secure computation, the correlated randomness is not as nice: the participants need to receive distinct, but appropriately correlated random-looking strings (for example, pairs (s_0, s_1) and (b, s_b) respectively, in the case of the random OT correlation).

The notion of pseudorandom correlation generator (PCG), which I will formally introduce later on, aims to achieve just that: a PCG $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ yields a way to generate to *distinct, but related* short strings $(k_0, k_1) \leftarrow \mathsf{PCG.Gen}$ such that evaluating $\mathsf{PCG.Expand}(k_0)$

and PCG.Expand(k_1) yields long, *random-looking but correlated* strings. As we will see, formalizing this notion is not straightforward and requires some care. Building a PCG is an even harder task: it requires a particularly delicate balance to achieve pseudorandomness while preserving some specific correlation – *id est*, to destroy all visible local patterns in the outputs while maintaining a specific global pattern among them.

3.2.2 The template

Equipped with the above ideas, we put forth the notion of secure computation in the *silent preprocessing model*. Concretely, a protocol in this model has three phases:

1. A one-time short interaction, where the participants generate the short correlated keys output by a pseudorandom correlation generator for their correlation of interest (*e.g.* random OTs), using a secure protocol to distributively evaluate the Gen algorithm of a PCG.

2. A "silent" computation phase, where the parties go offline and locally expand their short keys into long pseudorandom strings, which are indistinguishable from long random correlated strings, using the PCG.Expand algorithm.

3. A "non-cryptographic" online phase, where the participants use these long correlated strings in the usual way to securely compute the target function, for example with GMW.[2]

This template is summarized in Figure 3.2.2. What remains to do is to formally define pseudorandom correlation generators, in a way that allows to simultaneously (1) get a construction that instantiates this notion (ideally with an efficient construction), and (2) obtain a secure protocol when following the template above with this notion.

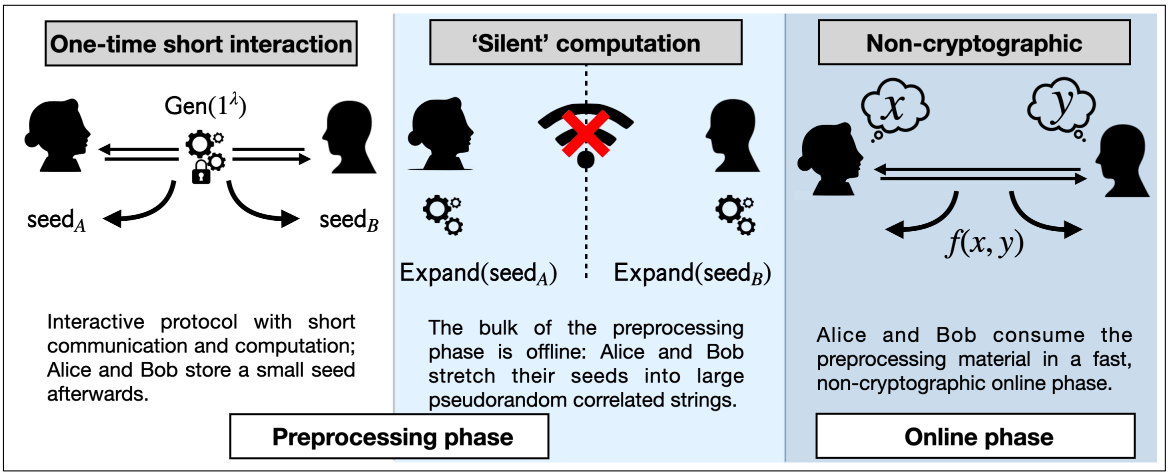

One-time short interaction
Gen(1^λ)
seed$_A$
seed$_B$
Interactive protocol with short communication and computation; Alice and Bob store a small seed afterwards.
'Silent' computation
Expand(seed$_A$)
Expand(seed$_B$)
The bulk of the preprocessing phase is offline: Alice and Bob stretch their seeds into large pseudorandom correlated strings.
Non-cryptographic
x
y
$f(x,y)$
Alice and Bob consume the preprocessing material in a fast, non-cryptographic online phase.
Preprocessing phase
Online phase

Figure 3.1: The three steps of an MPC protocol in the silent preprocessing model

[2] Note, however, that this phase is not information-theoretic anymore, because the correlated randomness used is not truly random, but only pseudorandom. But this is a minor inconvenience; the most important thing is that this phase does not use any heavy cryptographic operations, hence it is highly efficient.

3.3 Pseudorandom Correlation Generators: a Definition

We now overview the formal definition of pseudorandom correlation generators. At a high level, a PCG is a pair of algorithms that enables the two steps of the preprocessing phase represented in Figure 3.2.2:

- A seed generation algorithm, denoted Gen, outputs two short, correlated seeds. Each party will receive one seed (we use the terms "seed" and "key" interchangeably here).
- An expansion algorithm, that takes one party's seed, and expands it into a long pseudorandom string. When taken together, the two pseudorandom strings should satisfy the prescribed correlation (but appear random beyond that).

This definition abstracts out *how* the participants will run the Gen algorithm (note that one cannot simply let one of the parties run it, because it would reveal to this party the seed of the other party, which would compromise privacy). Looking ahead, Gen will typically be either run by a trusted third party (whose only trusted in generating some short, input-independent correlated seeds) or securely executed through a distributed protocol. The point is that whatever the cost of this distributed protocol, it only depends on Gen's runtime, which is *independent* of (or, rather, logarithmic in) the size of the target computation. Then, the expansion procedure is the bulk of the computation (that depends linearly on the amount of correlated randomness to be produced, which is proportional to the size of the target function), but it is applied locally: at this stage, the participants do not have to exchange messages, or even to be online.

3.3.1 A slightly more formal overview

More formally, for correctness, we require that the expanded output of a PCG is indistinguishable from truly random correlated strings that are sampled from the ideal correlation. Security, however, turns out to be much harder to formally define. We first explain a failed attempt and then sketch the right definition.

A straightforward (but broken) solution would be the following: a PCG for a target correlation $\mathcal{C}$ is secure if for any secure computation protocol Π in the correlated randomness model, the protocol Π' obtained by replacing the (true) correlated random string by short correlated seeds (locally expanded by the parties) is secure. Alas, in the paper where we introduced the notion of PCG [BCG+19b], we showed that this ideal security requirement would be impossible to meet: one can craft artificial protocols that are secure in the correlated randomness model, yet provably become insecure as soon as the true correlated randomness is replaced by short seeds[3].

Instead, in [BCG+19b], we introduced a weaker (but achievable) indistinguishability-based security notion. The notion requires that an adversary, given access to one of the short seeds k_σ, cannot distinguish the pseudorandom string $R_{1-\sigma}$ from a pseudorandom string that is chosen at random conditioned on (R_0, R_1) being appropriately correlated (where $R_\sigma = \mathsf{PCG}(\mathsf{k}_\sigma)$ is the expansion of the short seed k_σ). In other words, an adversary given

[3] To give a rough intuition of this impossibility result, the idea is to have a protocol that instructs the participants to reveal all their private inputs if their opponents ever manage to produce a succinct description of their correlated randomness. Since truly random strings cannot be compressed beyond their entropy, this remains secure in the correlated randomness model, but leaks all inputs whenever we use a PCG, since its purpose is exactly to distribute succinct representations of the correlated randomness!

access to a short seed cannot learn more about the other party's pseudorandom string than what is obvious given access to its own pseudorandom *output string*. What makes this notion a useful notion for PCGs is that

1. it is achievable: many constructions of PCGs, starting with [BCGI18; BCG+19b] were provably shown to achieve it (under standard cryptographic assumptions related to decoding problems in suitable linear codes), and

2. it can provably replace *securely* the correlated randomness in any protocol Π proven secure in the *corruptible correlated randomness model*, a weakening of the correlated randomness model that, informally, allows the adversary to choose themself what the correlated randomness of the corrupted participants will be (and the correlated randomness of the remaining parties is sampled afterward to be consistent with the choice of the adversary). This model avoids the impossibility result of [BCG+19b][4]. More importantly, essentially all known protocols ever designed in the correlated randomness model are secure in the corruptible correlated randomness model! This includes in particular GMW and its many variants.

Equipped with the above intuition, we now outline the full formal definition of PCGs. The definition comes from [BCG+19b], and builds upon an earlier definition of pseudorandom VOLE generator which we had introduced in [BCGI18] (retrospectively, a pseudorandom VOLE generator is a PCG for a specific correlation, the vector-OLE correlation, which is a variant of the OLE correlation covered in Section 2.7.2.2). The unified definition below uses also some later material that I wrote with Elette Boyle, Niv Gilboa, and Yuval Ishai for a book chapter of a yet-unpublished book[5] (as of 2023).

3.3.2 Formal definition of PCGs

To formally define PCGs, we first introduce the concept of a *correlation generator* as a PPT algorithm outputting correlated strings. We will use a correlation generation generator to define an ideal target correlation $(\mathcal{R}_0, \mathcal{R}_1)$. For simplicity, we assume that $(\mathcal{R}_0, \mathcal{R}_1)$ are two bit-strings of the same length n, though in some of the useful instances instances we will discuss they are more naturally interpreted as vectors over some finite ring.

Definition 3.3.1 (Correlation Generator)**.** *A PPT algorithm $\mathcal{C}$ is called a* correlation generator*, if $\mathcal{C}$ on input 1^n outputs a pair of strings in $\{0,1\}^n \times \{0,1\}^n$.*

Our security definition requires the target correlation to satisfy a technical requirement, which roughly says that it is possible to efficiently sample from the conditional distribution of $\mathcal{R}_0$ given $\mathcal{R}_1 = r_1$ and vice versa.

[4] That's because, in the corruptible correlated randomness model, the adversary can freely decide that the corrupted parties will receive strings with a succinct description; therefore, the protocol given as a counter-example in the correlated randomness model cannot be secure in the corruptible correlated randomness model.

[5] The book chapter is available on my webpage: `https://geoffroycouteau.github.io/assets/pdf/HSS_FSS.pdf`. It briefly covers PCGs, though its main focus is on the notions of function secret sharing and homomorphic secret sharing. I cover function secret sharing later in this manuscript.

Definition 3.3.2 (Reverse-sampleable Correlation Generator)**.** *Let $\mathcal{C}$ be a correlation generator. We say $\mathcal{C}$ is* reverse sampleable *if there exists a PPT algorithm* $\mathsf{RSample}$ *such that for $\sigma \in \{0,1\}$ the correlation obtained via:*

$$\{(\mathcal{R}'_0, \mathcal{R}'_1) \,|(\mathcal{R}_0, \mathcal{R}_1) \xleftarrow{\$} \mathcal{C}(1^n), \mathcal{R}'_\sigma := \mathcal{R}_\sigma, \mathcal{R}'_{1-\sigma} \xleftarrow{\$} \mathsf{RSample}(\sigma, \mathcal{R}_\sigma)\}$$

is computationally indistinguishable from $\mathcal{C}(1^n)$.

To illustrate the above notion, observe that the random OT correlation is reverse-sampleable: consider a random OT correlation, defined as a pair $((s_0, s_1), (b, s_b))$, where $s_0, s_1, b \xleftarrow{\$} \{0,1\}$. The reverse sampling algorithm $\mathsf{RSample}(\sigma, y_\sigma)$ works as follows: if $\sigma = 0$, parse y_σ as $y_\sigma = (s_0, s_1)$, sample $b \xleftarrow{\$} \{0,1\}$, and output (b, s_b); otherwise (*i.e.* if $\sigma = 1$) parse y_σ as $y_\sigma = (b, s)$, sample $s' \xleftarrow{\$} \{0,1\}$, and output $((1-b) \cdot s + b \cdot s', b \cdot s + (1-b) \cdot s')$. This immediatly generalizes to the reverse-sampleability of the n-fold random OT correlation, where the parties receive n pairs of random OT correlations.

The following definition of *pseudorandom correlation generators* generalizes an earlier definition of pseudorandom VOLE generator in [BCGI18].

Definition 3.3.3 (Pseudorandom Correlation Generator (PCG) [BCG+19b])**.** *Let $n = n(\lambda) \in \mathrm{poly}(\lambda)$ be a polynomial, and let $\mathcal{C}$ be a reverse-sampleable correlation generator. A* PCG for $\mathcal{C}$ *is a pair of algorithms* $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *with the following syntax:*

- $\mathsf{PCG.Gen}(1^\lambda)$ *is a PPT algorithm that given a security parameter λ, outputs a pair of seeds* $(\mathsf{k}_0, \mathsf{k}_1)$*;*
- $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ *is a polynomial-time algorithm that given party index $\sigma \in \{0,1\}$ and a seed k_σ, outputs a bit string $\mathcal{R}_\sigma \in \{0,1\}^n$.*

The algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *should satisfy the following:*

- ***Pseudorandomness.*** *The correlation obtained via:*

  $$\{(\mathcal{R}_0, \mathcal{R}_1) \,|(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), (\mathcal{R}_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma))_{\sigma=0,1}\}$$

  is computationally indistinguishable from $\mathcal{C}(1^n)$.

- ***Security.*** *For any $\sigma \in \{0,1\}$, the following two distributions are computationally indistinguishable:*

  $$\{(\mathsf{k}_{1-\sigma}, \mathcal{R}_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), \mathcal{R}_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \textit{ and}$$
  $$\{(\mathsf{k}_{1-\sigma}, \mathcal{R}_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), \mathcal{R}_{1-\sigma} \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_{1-\sigma}),$$
  $$R_\sigma \xleftarrow{\$} \mathsf{RSample}(\sigma, \mathcal{R}_{1-\sigma})\}$$

  where $\mathsf{RSample}$ *is the reverse sampling algorithm for correlation $\mathcal{C}$.*

In words, security in Definition 3.3.3 means that from the view point of party 0 holding a key k_0, the string $\mathcal{R}_1$ obtained by the other party is indistinguishable from a uniformly random string reverse-sampled from $\mathcal{R}_1 \leftarrow \mathsf{PCG.Expand}(0, \mathsf{k}_0)$, i.e., a string sampled uniformly at random conditioned on satisfying the target correlation with $\mathcal{R}_0$ (and a similar security condition holds in the other direction).

Note that the above definition is trivial to achieve in general: We can let $\mathsf{PCG.Gen}$ on input 1^λ return $(R_0, R_1) \leftarrow \mathcal{C}(1^{n(\lambda)})$, and simply define Expand to be the identity. Typically, we will be interested in non-trivial constructions of PCGs, in which $\mathsf{PCG.Expand}$ stretches a short seed to a long output. As a simple example, a standard pseudorandom generator $G : \{0,1\}^\lambda \rightarrow \{0,1\}^{n(\lambda)}$ naturally defines the following PCG for the target correlation $\mathcal{C}(1^n) = (r, r)$, where $r \overset{\$}{\leftarrow} \{0,1\}^{n(\lambda)}$: $\mathsf{PCG.Gen}$ output a pair of identical random seeds and $\mathsf{PCG.Expand}$ applies G locally on each seed. In the following, we will consider more involved constructions of PCGs for useful target correlations where neither of the outputs determines the other. These include Oblivious Transfer (OT) correlations, Oblivious Linear Evaluation (OLE) correlations, and (authenticated) multiplication triples.

3.3.3 Historical notes

The notion of pseudorandom correlation generators has its roots in the work of Beaver [Bea96], where the potential of the notion was first envisioned, but could not be realized without an amount of communication scaling with the target number of correlations. The first design of a PCG (beyond the simple equality correlation) dates back to Gilboa and Ishai's work [GI99], which gave an efficient multiparty PCG for any *linear* additive correlations, where a linear correlation refers to a correlation where the strings $(\mathcal{R}_1, \cdots, \mathcal{R}_N)$ received by the parties are in the kernel of a linear function L: $L(\mathcal{R}_1, \cdots, \mathcal{R}_N) = 0$ (for example, this is the case of the equality correlation, where two parties receive $\mathcal{R}_0 = \mathcal{R}_1$, since $\mathcal{R}_0 - \mathcal{R}_1 = 0$).

Linear correlations enjoy various applications but are not sufficient for the application to silent secure computation, where the target correlations (random OTs, OLEs, Beaver triples...) are all at least degree-2 polynomials. What I mean there is that the strings $(\mathcal{R}_0, \mathcal{R}_1)$ output by the correlation generator $\mathcal{C}$ can be parsed as vectors over a field $\mathbb{F}$ satisfying $P(\mathcal{R}_0, \mathcal{R}_1) = 0$ over $\mathbb{F}$, where P is a (vector of multivariate) degree-2 polynomial(s). For example, if $\mathcal{C}$ generates n copies of an OLE correlation, one party gets n pairs $(u_i, v_i) \in \mathbb{F}^2$ and the other party gets $(x_i, w_i) \in \mathbb{F}_2$, which are zeroes of $P(\vec{u}, \vec{v}, \vec{x}, \vec{w}) = \vec{w} - (\vec{u} \odot \vec{x} + \vec{v})$, where $\odot$ denotes the component-wise product.

The first PCG for a degree-2 correlation was achieved in [BCGI18], albeit still for a correlation that does not suffice for general circuits (but rather for circuits with high fan-in multiplications). The work also introduced a security notion for their pseudorandom correlation generator, which later formed the basis of the general definition we presented in this section. Eventually, the first "full-fledged" PCG for a secure-computation-complete correlation, the OT correlation, was presented in [BCG+19b], together with a formal definition of the general notion, and the proof that it suffices to instantiate the silent preprocessing framework for any protocol proven secure in the corruptible correlated randomness model.

3.4 Pseudorandom Correlation Generators: a Template

The purpose of this section is to outline a general template for building pseudorandom correlation generators. This template emerged progressively and was refined throughout many of my works [BCG+17; BCGI18; BCG+19b; BCG+19a; BCG+20b; BCG+20a; CRR21a; BCG+22; BCCD23], and captures all known efficient constructions of PCGs for concrete correlations of interest to date. I will attempt to provide a step-by-step, intuitive exposition of how one arrives at this template, and formally introduce along the way the necessary building blocks. For the sake of concreteness, I will focus in this overview on the notion of two-party additive correlation:

Definition 3.4.1. *A two-party additive correlation of length m is a correlation with the following structure: the two parties receive uniformly random additive shares of $C(r_i)$ for $i = 1$ to m, where C is a public function defining the correlation, and $(r_1, \cdots, r_m)$ are uniformly random inputs.*

For example, Beaver triples are an additive correlation, where the function C is given by $C : (a, b) \to (a, b, a \cdot b)$. Random OTs and OLEs are not immediately additive correlations (they are isomorphic to random shares of random products $a \cdot b$ over some field, where a and b are given to the two players instead of being shared between them), but they are in a sense simpler (because part of the correlation is given to the parties instead of being shared between them), and PCGs for these correlations will follow a similar template.

The purpose of a PCG for an additive correlation C is to generate two short keys $(\mathsf{k}_0, \mathsf{k}_1)$ which, once expanded, yield pseudorandom shares of $(C(r_1), \cdots, C(r_m))$, where $(r_1, \cdots, r_m)$ itself is a pseudorandom string. Notice how pseudorandomness appears in two places: internally, to generate $(r_i)_{i \leq m}$, and externally, to yield pseudorandom shares of the evaluation of C on $r_1 \cdots r_m$. Looking ahead, the template combines two distinct ingredients for each notion of pseudorandomness, which we now outline.

3.4.1 PCGs from shares of a function

Fix for now a pseudorandom generator G, which will be made explicit later on. At a high level, G will be a suitable PRG that generates generates $(r_1, \cdots, r_m)$ from a short seed k. What we mean by "suitable" depends heavily on how we instantiate the external component. We first cover this external component: a method to distribute the shares of $(C(r_1), \cdots, C(r_m))$ from short keys. With a slight abuse of notation, we will write $G_k(i)$ to denote the function which has k hardcoded in its description, and which returns on input i the i-th component of the output $G(k)$. This is equivalent to viewing $G(k)$ as the *truth table* of the function $G_k : i \to G_k(i) = G(k)_i$.

The template will rely on a method to distribute short keys $(\mathsf{k}_0, \mathsf{k}_1)$ to the parties that encode the secret function

$$F_k = C \circ G_k.$$

This encoding will be a pair of functions $(\mathsf{Gen}, \mathsf{Eval})$, where Gen generates the short keys $(\mathsf{k}_0, \mathsf{k}_1)$ encoding the secret function F_k, and Eval is an evaluation algorithm that outputs a string y_σ given an input i and a key k_σ for $\sigma \in \{0, 1\}$. We will want this encoding to satisfy three fundamental properties:

- It has to be *succinct*: the size of the keys given by the parties should be not much larger than the description length of $F_k = C \circ G_k$ (which is essentially the length of k, since C and G are already public information).

- It has to be *private*: given its part k_σ of the encoding, the party P_σ should not be able to infer anything about the secret input k.

- It has to *preserve evaluation*: given their respective keys, the parties should be able to compute additive shares of the evaluation of $C \circ G_k$ on any input i. That is, for any i, it must hold that $\mathsf{Eval}(\mathsf{k}_0, i) + \mathsf{Eval}(\mathsf{k}_1, i) = F_k(i) = (C \circ G)(k)_i = C(r_i)$.

Note that any two of the above requirements are easy to satisfy via trivial construction: one gets privacy and evaluation preservation by distributing shares of the entire truth table of the function (hence Eval simply requires a lookup to the table share), but this does not satisfy succinctness. Defining the encodings to be simply k yields succinctness while preserving evaluation (the parties can compute $C \circ G_k$ in the clear), but breaks security. Of course, satisfying succinctness and privacy without preserving evaluation is also trivial (just define the keys to be the empty string). The definition becomes however non-trivial as soon as one insists on achieving all three properties at once.

It is not too hard to see that if we find an encoding of $C \circ G_k$ that satisfies the three properties above, we are done: $\mathsf{PCG.Gen}$ will simply sample a random key k and output the encoding $(\mathsf{k}_0, \mathsf{k}_1)$ of the function $F_k = C \circ G_k$. Then, $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ computes and returns $\mathsf{Eval}(\mathsf{k}_\sigma, i)$ for $i = 1$ to m. Because the encoding is evaluation preserving, the outputs of $\mathsf{PCG.Expand}$ form additive shares of $((C \circ G)(k)_1, \cdots, (C \circ G)(k)_m) = (C(r_1), \cdots C(r_m))$. Security follows from the privacy of the encoding, and non-triviality from the succinctness. This was for the intuition – now, let's make this more formal.

3.4.2 Function secret sharing

The above way of encoding a secret function $C \circ G_k$ is called function secret sharing. In full generality, an N-party FSS scheme splits a *function* $f : \{0,1\}^\ell \to \mathbb{G}$ from a function class $\mathcal{F}$ into N additive shares $f_i : \{0,1\}^\ell \to \mathbb{G}$, each represented by a key k_i, where every strict subset of the keys k_i hides f. The correctness requirement is that the N functions f_i represented by the keys add up to f. Namely, there is a function Eval (defining the function f_i represented by k_i), such that for every $x \in \{0,1\}^\ell$ we have $f(x) = \sum_i \mathsf{Eval}(k_i, x)$. The challenge is to design efficient FSS schemes in which the key size grows polynomially with the input length ℓ. The short keys k_i can be viewed as *compressed* additive shares of the truth table of f. This is only possible for classes $\mathcal{F}$ of *structured* functions f that have a short (polynomial-size) description, and inevitably require one to settle for *computational* hiding of f. Later, I will focus mostly on the two-party setting, which is the one I use for our target application to PCGs – and also the one for which suitable efficient constructions are known. Part of the formal definition outlined below is adapted from a book chapter (for an upcoming book) which I co-authored with Elette Boyle, Niv Gilboa, and Yuval Ishai, available on my webpage.

3.4.2.1 Modeling function families

Formally, a *function family* is defined by a pair $\mathcal{F} = (P_\mathcal{F}, E_\mathcal{F})$, where $P_\mathcal{F} \subseteq \{0,1\}^*$ is an infinite collection of function descriptions $\hat{f}$, and $E_\mathcal{F} : P_\mathcal{F} \times \{0,1\}^* \to \{0,1\}^*$ is a polynomial-time algorithm defining the function described by $\hat{f}$. Concretely, each $\hat{f} \in P_\mathcal{F}$ describes a corresponding function $f : D_f \to R_f$ defined by $f(x) = E_\mathcal{F}(\hat{f}, x)$. We assume for simplicity that $D_f = \{0,1\}^\ell$ for a positive integer ℓ and always require R_f to be a finite Abelian group, denoted by $\mathbb{G}$. For simplicity, we will typically identift f with its description $\hat{f}$ and write $f \in \mathcal{F}$ instead of $\hat{f} \in P_\mathcal{F}$, and assume when passing f as input that this also includes an

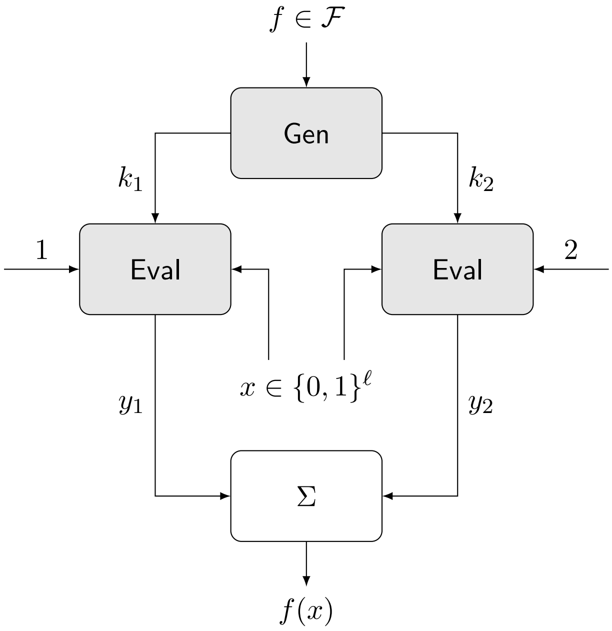


Figure 3.2: Representation of a 2-party function secret sharing scheme for a function class $\mathcal{F} = \{f : \{0,1\}^\ell \to \mathbb{G}\}$. Σ denotes the sum over $\mathbb{G}$.

explicit description of both D_f and R_f, as well as a parameter $|f|$ which denotes the size (in bits) of the description of f.

3.4.2.2 Defining function secret sharing

The definition given below follows the exposition given in [BGI16b; BGI15], but simplifies some of the formalism for clarity of the exposition.

Definition 3.4.2 (FSS: Syntax)**.** *An N-party function secret sharing* (FSS) scheme *is a pair of PPT algorithms* (Gen, Eval) *with the following syntax:*

- Gen$(1^\lambda, f)$*, on input 1^λ and a function $f : \{0,1\}^\ell \to \mathbb{G}$, outputs an N-tuple of keys $(k_1, \dots, k_N)$.*
- Eval(i, k_i, x) *is a polynomial-time* evaluation algorithm*, which on input $i \in [N]$ (party index), key k_i and input $x \in \{0,1\}^\ell$, outputs a group element $y_i \in \mathbb{G}$.*

When N is omitted, it is understood to be 2.

Definition 3.4.3 (FSS: Requirements)**.** *Let $\mathcal{F}$ be a function family and define* Leak(f) *to be $(1^\ell, |f|, \mathbb{G})$ for any $f : \{0,1\}^\ell \to \mathbb{G}$.*[6] *Let N (the number of parties) and t (the secrecy threshold) be positive integers. An N*-party *t*-secure FSS for $\mathcal{F}$ with leakage Leak *is a pair* (Gen, Eval) *as in Definition 3.4.2, satisfying the following requirements.*

[6] Leak models the leakage that the adversary is allowed to get about the secret function f; more general notions of allowed leakage can be considered, see [BGI16b] for more discussions on this aspect

- ***Correctness:*** *For all $f : \{0,1\}^\ell \to \mathbb{G}$ in $\mathcal{F}$, and every $x \in \{0,1\}^\ell$, it holds that*

$$\Pr\left[\sum_{i=1}^{N} \mathsf{Eval}(i, k_i, x) = f(x)\right] = 1,$$

where the probability is taken over the random sampling of $(k_1, \ldots, k_N) \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda, f)$.

- ***Secrecy:*** *For every set of corrupted parties $S \subset [N]$ of size t, there exists a PPT simulator* Sim *such that for every sequence $f_1, f_2, \ldots$ of polynomial-size functions from $\mathcal{F}$, the outputs of the following experiments* Real *and* Ideal *are computationally indistinguishable:*
  - *Real*(1^λ)*: $(k_1, \ldots, k_N) \leftarrow \mathsf{Gen}(1^\lambda, f_\lambda)$; Output $(k_i)_{i \in S}$.*
  - *Ideal*(1^λ)*: Output $\mathsf{Sim}(1^\lambda, \mathit{Leak}(f_\lambda))$.*

When t is omitted it is understood to be $N - 1$.

3.4.2.3 PCG from FSS

Equipped with this formal definition, let us state the core theorem that underlies our template:

Theorem 3.4.4. *Let $C : \mathbb{F} \mapsto \mathbb{G}$ (where $\mathbb{F}$ is some field and $\mathbb{G}$ is some Abelian group) be a function and let $G : \{0,1\}^\ell \mapsto \mathbb{F}^m$ be a PRG. Let $\mathcal{F} = \{C \circ G_k : [m] \mapsto \mathbb{G}\}$ be the family of functions $F_k = C \circ G_k$ indexed by the hardcoded input k (we view C and G as public functions, and k as the secret description of $C \circ G_k$). Let* (Gen, Eval) *be a 2-party function secret sharing for the class of functions $\mathcal{F}$. Consider the following construction:*

- PCG.Gen(1^λ)*: sample $k \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and output $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{Gen}(1^\lambda, k)$*
- PCG.Expand$(\sigma, \mathsf{k}_\sigma)$*: for $i = 1$ to m, compute $y_i \leftarrow \mathsf{Eval}(\sigma, \mathsf{k}_\sigma, i)$. Output $\mathcal{R}_\sigma \leftarrow (y_1, \cdots, y_m)$.*

The construction (PCG.Gen, PCG.Expand) *is a secure PCG for the two-party additive correlation defined by C.*

The proof of this theorem is omitted from this manuscript: it is rather elementary but slightly tedious. For the interested reader, it is covered in the book chapter available on my webpage.

3.5 Instantiating the Template I: a Step-by-Step FSS Construction

So far, we have established the following: if we have a PRG G and an FSS scheme for the function class $\mathcal{F} = \{C \circ G_k\}$, then we can build a PCG for the additive correlation defined by C. This begs the question: for what function classes $\mathcal{F}$ do we have efficient FSS constructions? In particular, what kind of PRGs G and correlations C yield functions $C \circ G_k$ for which efficient FSS schemes exist?

Here, I want to stress the keyword "efficient". Under suitable cryptographic assumptions, it has been shown in the work of [BGI15] (which introduced the notion of FSS) that FSS can be

Chapter 3

constructed for *all* polytime-computable functions. However, this construction employs very heavy machinery – in particular, it requires very strong forms of fully homomorphic encryption, which is extremely expensive (and if the reader remembers our discussions from Section 2.4.2, this is exactly the type of primitive that we want to avoid in the GMW paradigm). For a PCG construction to be useful, it must instead build upon a more realistically usable flavor of FSS.

3.5.1 Sharing the all-zero function

Our starting point is the standard naive and inefficient approach to FSS: to secretly share a function, it suffices to share its truth table. Then, the question becomes: which functions have truth tables whose random shares can be compressed? A seemingly stupid, but ultimately illuminating example of one such function is the all-zero function (yes, the one which always outputs zero) $Z : [m] \mapsto \{0\}$. The truth table of Z is a length-m vector of zeroes. To randomly share the truth table of Z, it suffices to sample a uniformly random key $K \overset{\$}{\leftarrow} \{0,1\}^m$ and to give K to each of the two parties: $K \oplus K$ is the length-m all-zero vector, which is indeed the truth table of Z.

Fortunately, we already know how to compress identical random shares: just use a pseudorandom generator! Let $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^m$ be a PRG. To deal compressed shares of Z to the parties, sample a random short key $s \overset{\$}{\leftarrow} \{0,1\}^\lambda$, and give s to both participants. To obtain shares of $Z(i)$ for some $i \in [m]$, the participants reconstruct $K \leftarrow \mathsf{PRG}(s)$ and output the i-th bit of K.[7]

3.5.2 Sharing point functions

Of course, the all-zero function is not a useful function. But let us take a tiny step from there: what if our function still evaluates to zero everywhere, except on a *single* point? Such a function is called a point function. We will write $f_{\alpha,\beta} : [m] \mapsto \{0,1\}^*$ to denote the point function such that $f_{\alpha,\beta}(x) = \beta$ if $x = \alpha$, and $f_{\alpha,\beta}(x) = 0$ otherwise. To succinctly share $f_{\alpha,\beta}$ (without revealing (α, β)), the key idea is to carefully balance between the succinct sharing of the all-zero function (because most of the truth table of $f_{\alpha,\beta}$ contains zeroes anyway) and a naive (non-succinct) sharing of the part of the truth table that contains β.

3.5.2.1 A first construction

Write the length-m truth table of $f_{\alpha,\beta}$ as a $\sqrt{m} \times \sqrt{m}$ square. We will share $f_{\alpha,\beta}$ row by row. Let (α_0, α_1) denote the coordinate of the nonzero entry in this square (that is, (α_0, α_1) is the decomposition of α in basis $\sqrt{m}$). Each row i with $i \neq \alpha_0$ can be seen as the $\sqrt{m}$-length truth table of an all-zero function: which we already know how to share: sample a short key $k_i \overset{\$}{\leftarrow} \{0,1\}^\lambda$ for a PRG PRG, and give the key to both participants. This way, $(\mathsf{PRG}(k_i), \mathsf{PRG}(k_i))$ form two pseudorandom shares of the zero vector.

For the α_0-th row (which we denote r_{α_0}), instead, we sample two different short keys $(k_{\alpha_0}, k'_{\alpha_0})$ and hand one to each participant – notice that from their viewpoint, nothing distinguishes α_0 from the other rows (for each row, they received a uniformly random short

[7] Of course, because Z does not contain any secret, the participants could also just always define their shares of Z to be the all-zero string and output 0 on each evaluation. But we will use the PRG-based approach as a starting point to build more advanced forms of FSS, so bear with me!

key). However, $(\mathsf{PRG}(k_{\alpha_0}), \mathsf{PRG}(k'_{\alpha_0}))$ form shares of $\mathsf{PRG}(k_{\alpha_0}) \oplus \mathsf{PRG}(k'_{\alpha_0})$, which is not the correct row r_{α_0}. Hence, it remains to find a way to correct the value of this row obliviously (*i.e.* without leaking α_0) without touching the other rows.

To do so, we define $\Delta = r_{\alpha_0} \oplus \mathsf{PRG}(k_{\alpha_0}) \oplus \mathsf{PRG}(k'_{\alpha_0})$, which is the offset between the correct value r_{α_0} and the incorrect value $\mathsf{PRG}(k_{\alpha_0}) \oplus \mathsf{PRG}(k'_{\alpha_0})$, and give it to both participants. Then, the goal is to let the participants obliviously XOR the offset Δ to their α_0-th share, without XORing it to their other shares. To do so, we sample $\sqrt{m}$ additional random bits $(b_1, \cdots, b_{\sqrt{m}})$, which we give to the first participant. To the second participant, we give the same bitstring *except that we flip the* α_0*-th bit.*

Then, to evaluate their share of $f_{\alpha,\beta}(i)$, the first participant computes the i_1-th bit of $b_{i_0} \cdot \Delta \oplus \mathsf{PRG}(k_{i_0})$ (where (i_0, i_1) are the coordinates of $i \in [m]$ in the $\sqrt{m} \times \sqrt{m}$ square). For every $i_0 \neq \alpha_0$, the second participant will output the i_1-th bit of $b_{i_0} \cdot \Delta \oplus \mathsf{PRG}(k_{i_0})$ as well: their outputs are indeed shares of 0. When $i_0 = \alpha_0$, the second participant will instead output the i_1-th bit of $(1 \oplus b_{\alpha_0}) \cdot \Delta \oplus \mathsf{PRG}(k'_{\alpha_0})$. If we XOR the shares of both participants, we get:

$$\begin{aligned}
&(b_{\alpha_0} \cdot \Delta \oplus \mathsf{PRG}(k_{\alpha_0})) \oplus ((1 \oplus b_{\alpha_0}) \cdot \Delta \oplus \mathsf{PRG}(k'_{\alpha_0}))\\
=&\Delta \cdot (b_{\alpha_0} \oplus b_{\alpha_0} \oplus 1) \oplus (\mathsf{PRG}(k_{\alpha_0} \oplus \mathsf{PRG}(k_{\alpha_0})\\
=&\Delta \oplus (\mathsf{PRG}(k_{\alpha_0} \oplus \mathsf{PRG}(k_{\alpha_0})\\
=&r_{\alpha_0} \text{ by definition of } \Delta,
\end{aligned}$$

which is again the right value. To make this a bit more visual, the shares received by each participant are represented in Figure 3.3.

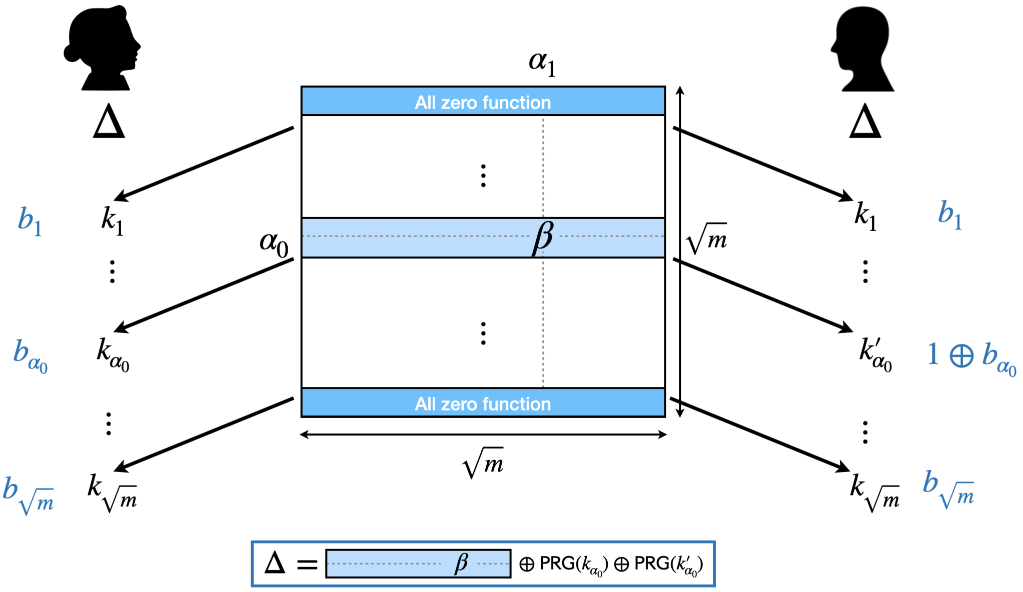


Figure 3.3: A function secret sharing for the point function $f_{\alpha,\beta}$ with keys of length proportional to $\sqrt{m}$, from any PRG PRG.

If we denote by λ the length of the short PRG keys (which is typically independent of m), the FSS scheme which we just constructed has keys of size $\Theta(\lambda \cdot \sqrt{m})$ (one can reduce this to

$\Theta(\sqrt{\lambda m})$ with a better balancing, using a $\sqrt{m/\lambda} \times \sqrt{\lambda m}$-sized rectangle instead of a square). This is already an important improvement compared to the naive solution of sharing the entire truth table, which yields keys of size $\Theta(m)$.

3.5.2.2 Improved constructions

We can do even much better: the work of Gilboa and Ishai [GI14] followed by subsequent works by Boyle, Gilboa, and Ishai [BGI15; BGI16b] showed that using clever variants of the above construction together with more careful balancing strategies, one can recursively compress the length of the shared keys. Concretely, and denoting $n = \log m$ the bitsize of an input to a point function f with domain $[m]$:

- The work of [GI14], which introduced the notion of FSS for point functions, also gave a recursive construction with key size $O(n^{\log_2 3} \cdot \lambda)$.
- This key length was improved to $O(n\lambda)$ bits in [BGI15] via a tree-based construction.
- The current best construction [BGI16b] has key size $\approx n\lambda + |\beta|$. More precisely, the key size is $\lambda + n(\lambda + 2) - \lfloor \log \lambda / |\beta| \rfloor$ bits.

Formally,

Definition 3.5.1 (Distributed Point Function)**.** *A point function $f_{\alpha,\beta}$, for $\alpha \in [m]$ and $\beta \in \mathbb{G}$, is defined to be the function $f : [m] \to \mathbb{G}$ such that $f(\alpha) = \beta$ and $f(x) = 0$ for $x \neq \alpha$. A* Distributed Point Function *(DPF) is an FSS for the family of all point functions, with the leakage $\mathsf{Leak}(\hat{f}) = (m, \mathbb{G})$.*

The theorem below summarizes the state of the art regarding efficient point functions. Remember also that our application to building pseudorandom correlation generators requires running the FSS evaluation algorithm on the entire domain, which can typically be done more efficiently than running the FSS Eval algorithm individually on each entry. We denote by FullEval an algorithm that outputs the result on Eval on the full domain.

Theorem 3.5.2 (PRG-based DPF [BGI16b], Theorems 3.3 and 3.4)**.** *Given a PRG* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, *there exists a DPF for point functions $f_{\alpha,\beta} : [m] \to \mathbb{G}$ with key size $\log m \cdot (\lambda+2) + \lambda + \lceil \log_2 |\mathbb{G}| \rceil$ bits. For $\ell = \lceil \frac{\log |\mathbb{G}|}{\lambda+2} \rceil$, the key generation algorithm* Gen *invokes* PRG *at most $2(\log m + \ell)$ times, the evaluation algorithm* Eval *invokes* PRG *at most $\log m + \ell$ times, and the full evaluation algorithm* FullEval *invokes* PRG *at most $m(1 + \ell)$ times.*

3.5.3 Sharing "matrix × sparse vector"

At this stage, the reader might feel like we have not made much progress: to instantiate the template, we need an FSS scheme for a function class that contains $F_k = C \circ G_k$, where C describes a correlation (typically a low-degree multivariate polynomial, for example $C(a, b) = (a, b, a \cdot b)$ for Beaver triples) and G is a pseudorandom generator. So far, the only functions we managed to compress are point functions, *i.e.*, functions with a single nonzero entry, which are way too restricted to capture the functions we care about.

The surprising twist is that, quite on the contrary, FSS for point functions suffices to instantiate the template, provided that C is a sufficiently low-degree polynomial, and for a suitable choice of G. The key insight is that because the Eval algorithm of an FSS scheme outputs additive shares, FSS naturally supports composition with linear combinations. This observation will allow us to significantly expand the class of functions, making only a black box use of an FSS scheme for point functions.

Concretely, fix a "sparsity" parameter t, whose value will be discussed later (but the reader can think of it as being some fixed security parameter, with $t \approx 50$ being a reasonable choice). Define a *t-point function* to be a function $f_{\vec{\alpha},\vec{\beta}}$, with $\vec{\alpha} \in [m]^t$ has distinct entries, and $\vec{\beta} \in \mathbb{G}^t$, such that $f_{\vec{\alpha},\vec{\beta}}(\alpha_j) = \beta_j$ for $j = 1$ to t, and $f_{\vec{\alpha},\vec{\beta}}(i) = 0$ otherwise (that is, the truth table of $f_{\vec{\alpha},\vec{\beta}}$ is 0 everywhere, except on t points indicated by $\vec{\alpha}$). Because we have FSS for point functions and because the sum of additives shares is also an additive share of the sum, we immediately have:

Corollary 3.5.3 (Multi-Point Function Secret Sharing)**.** *Given a PRG* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$*, there exists an FSS scheme for t-point functions $f_{\vec{\alpha},\vec{\beta}} : [m] \to \mathbb{G}$ with key size $t \cdot (\log m \cdot (\lambda + 2) + \lambda + \lceil \log_2 |\mathbb{G}| \rceil)$ bits. For $\ell = \lceil \frac{\log |\mathbb{G}|}{\lambda+2} \rceil$, the key generation algorithm* Gen *invokes* PRG *at most $2t(\log m + \ell)$ times, the evaluation algorithm* Eval *invokes* PRG *at most $t \cdot (\log m + \ell)$ times, and the full evaluation algorithm* $\mathsf{FullEval}$ *invokes* PRG *at most $t \cdot m \cdot (1+\ell)$ times.*

The construction is immediate: write $f_{\vec{\alpha},\vec{\beta}}$ as the sum of point functions $\sum_{j=1}^{t} f_{\alpha_j,\beta_j}$. The key generation algorithm generates t independent key pairs, one for each f_{α_j,β_j}, and the evaluation algorithm, on input i, runs Eval with each of the t keys to get shares $(y_1, \cdots, y_t)$ of the $f_{\alpha_j,\beta_j}(i)$. The output share is just the sum over $\mathbb{G}$ of the y_j's.

To put it differently, when two parties (P_0, P_1) evaluate an FSS for a t-point function on its entire domain $[m]$, they obtain pseudorandom additive shares $(\vec{e}_0, \vec{e}_1)$ of a length-m sparse vector $\vec{e}$, which has exactly t nonzero entries. The FSS keys, which have total length $O(t \cdot \lambda \cdot \log m)$, form a succinct encoding of shares of $\vec{e}$ which does not reveal anything about $\vec{e}$, beyond the fact that it is a t-sparse vector. Furthermore, because the shares are additive, when the output group $\mathbb{G}$ is a ring $\mathcal{R}$, the participants can post-process their share of $\vec{e}$ by multiplying it with a public matrix H: their post-processed shares $(H \cdot \vec{e}_0, H \cdot \vec{e}_1)$ form additive shares of $H \cdot \vec{e}_0 + H \cdot \vec{e}_1 = H \cdot \vec{e}$ by linearity. Summing up, we have:

Corollary 3.5.4 (Matrix × Sparse Vector Function Secret Sharing)**.** *Given a matrix $H \in \mathcal{R}^{\ell \times m}$ and a t-sparse vector $\vec{e} \in \mathcal{R}^m$, let $F_{H,\vec{e}} : [m] \mapsto \mathcal{R}$ which, on input $i \in [m]$, output the i-th entry of $H \cdot \vec{e}$. Let $\mathcal{F} = \{F_{H,\vec{e}}\}$ denote the class of all such functions. Then assuming the existence of a PRG* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$*, there exists an FSS scheme for the function class $\mathcal{F}$ with leakage* $\mathsf{Leak}(F_{H,\vec{e}}) = (t, \mathcal{R}, H)$*, with key size $O(t \cdot \lambda \cdot \log m)$.*

As long as t is not too large, this yields a significant compression compared to sharing the m-sized truth table of $F_{H,\vec{e}}$ directly. Before we finally move on to the choice of the PRG G that will fit in the class for which we have efficient FSS, let us make one final step.

Observation 3.5.5. *We can further expand the function class to the class of functions $F'_{H,\vec{e}}$ which, on input $i \in [m]$, outputs the i-th entry of $P(H \cdot \vec{e})$, where P is a multi-output low-degree multivariate polynomial over $\mathcal{R}$ (that is, each output of P is a low-degree multivariate polynomial over $\mathcal{R}$), by observing that computing $P(H \cdot \vec{e})$ reduces to computing a linear function on $\vec{e}^{\otimes \deg P}$, which denotes $\vec{e}$ tensored with itself* $\deg P$ *times, and that $\vec{e}^{\otimes \deg P}$ is a $t^{\deg P}$-sparse vector. In particular, this captures the class of functions $Q \circ F_{M \cdot \vec{v}}$ which, on input*

$i \in [m]$, outputs $Q(F_{M \cdot \vec{v}}(i))$ (where Q is a multi-ouput low-degree multivariate polynomial). The FSS scheme for this extended function class $\mathcal{F}'$ has key size $O(t^{\deg Q} \cdot \lambda \cdot \log m)$.

For simplicity, let us focus on the case where P is a degree-2 polynomial (the general case follows similarly). Given a vector $\vec{u}$, a degree-2 polynomial in $\vec{u}$ can be written as a linear function L of the tensor product $\vec{u} \cdot \vec{u}^\intercal$ of $\vec{u}$ with itself.[8] Hence, we can rewrite $P(H \cdot \vec{e})$ as $L(H \cdot (\vec{e} \cdot \vec{e}^\intercal) \cdot H^\intercal)$, for a suitable linear function L. Because the map $X \to H \cdot X \cdot H^\intercal$ is itself linear, this computation boils down to evaluating a suitable linear function $L_{\vec{H}}$ of the matrix $E = \vec{e} \cdot \vec{e}^\intercal$.

Now, observe that since $\vec{e}$ is a t-sparse vector, E is a t^2-sparse matrix: the two parties can obtain shares of E using a sum of t^2 point functions with domain $[m^2]$, and locally evaluate the linear function $L_{\vec{H}}$ on their shares afterward, obtaining shares of $P(H \cdot \vec{e})$. This yields an FSS scheme for the class of functions $F'_{H,\vec{e}}$ with key size $O(t^2 \cdot \lambda \cdot \log m)$. Generalizing this approach to degree-d multivariate polynomials yields a scheme with key size $O(t^d \cdot \lambda \cdot \log m)$, which remains much smaller than m for small values of d.

3.5.4 Concluding words

It took us a bit of time, but I hope that I made it clear to the reader that arriving at FSS for functions like $Q \circ F_{H,\vec{e}}$ only takes elementary steps – we did not use any heavy cryptographic machinery or advanced mathematical reasoning. Now, getting back to our original goal: what we want FSS for is the function $F_k = C \circ G_k$, where $G_k(i)$ output the i-th entry of the PRG evaluation $G(k)$, and C is our correlation. For most applications, C will be a low-degree polynomial – for example, a degree-two polynomial in the case of Beaver triples. This is perfect: we can set the low-degree polynomial Q in our construction above to be exactly the correlation C we care about. The function $F_{H,\vec{e}}$ outputs the i-th entry of $H \cdot \vec{e}$: *if only* the map $\vec{e} \to H \cdot \vec{e}$ could turn out to be a PRG, we would be done! Because the description of $\vec{e}$ can be encoded using $O(t \cdot \log m)$ bits (remember, it's a t-sparse vector) and $H \cdot \vec{e}$ is a length-ℓ vector, for a large enough ℓ, this mapping is indeed expanding. But is it pseudorandom? This will be the focus of the next section.

3.6 Instantiating the Template II: Coding Theory to the Rescue

Before we continue, we're going to need a bit of background on coding theory. I will keep it relatively short – there are countless resources out there providing an extensive treatment of the subject – but understanding notions such as *parity-check matrices* and the *minimum distance* of a code is a must to understand the design principles behind PCG constructions.

3.6.1 Preliminaries on coding theory

A code, or error-correcting code, is (according to Wikipedia) "a technique used for controlling errors in data transmission over unreliable or noisy communication channels". Concretely, it is a method that associates a *codeword* to every possible message, such that even if some of the codeword entries are corrupted by some noise during the transmission, retrieving the

[8] That is, the square matrix containing all degree-2 monomials in the entries of $\vec{u}$. If we want P to also contain degree-1 or constant monomials, we can append a 1 to $\vec{u}$ – we ignore this minor technicality to avoid making notations heavier.

original message remains possible. A linear code is a code such that the mapping between messages and codewords is a linear map. Linear codes capture many (if not most) of the codes used in the wild. Formally:

Definition 3.6.1 (*Linear Codes*)**.** *Let $\mathbb{F}$ be a field, and (k, n) be integers with $k < n$. A linear code $\mathcal{C}$ with dimension k and codeword length n over $\mathbb{F}$ is a linear subspace of $\mathbb{F}^n$:*

$$\mathcal{C} := \{\vec{c} = G \cdot \vec{x} \mid \vec{x} \in \mathbb{F}^k\},$$

where $G \in \mathbb{F}^{n \times k}$ is called the generator matrix *of $\mathcal{C}$. We say that $\mathcal{C}$ is an $[n, k, d]_q$-code if $|\mathbb{F}| = q$ and $d = \min_{\vec{x} \in \mathbb{F}^k \setminus \{0\}^k} \mathsf{HW}(G \cdot \vec{x})$, where HW denotes the Hamming weight (number of nonzero entries). The value d is called the* minimum distance *of $\mathcal{C}$. Equivalently, d is the smallest Hamming distance between any pair of codewords.*

We say that the generator matrix G is in *systematic form* if

$$G = \begin{bmatrix} I_k \\ G' \end{bmatrix},$$

where I_k is the $k \times k$ identity matrix. The systematic form of a generator matrix G can be obtained by performing elementary row operations (assuming the columns of G are linearly independent).

Parity-check matrix. Let $m \leftarrow n - k$. A parity-check matrix of a code $\mathcal{C}$ is a maximal rank matrix such that $H \cdot G = 0$. Formally:

Definition 3.6.2 (*parity-check matrix*)**.** *We say that a matrix H is a* parity-check matrix *of a code $\mathcal{C}$ if $\mathcal{C} = \ker(H) = \{\vec{c} \mid H \cdot \vec{c} = 0\}$. It is not too hard to see that $H \in \mathbb{F}^{m \times n}$. The code generated by $H^\intercal$ is called the* dual code *of $\mathcal{C}$ and is denoted $\mathcal{C}^\perp$.*

One can verify that the minimum distance d of a code $\mathcal{C}$ also corresponds to the smallest number of linearly dependent columns in a parity-check matrix of $\mathcal{C}$.

3.6.2 The hardness of syndrome decoding

The historical purpose of a code is to enable error correction: given a corrupted codeword $\vec{c} + \vec{e}$, where $\vec{e}$ is some noise added by the transmission channel, it should be possible to retrieve the original message $\vec{x}$ provided that the amount of noise is not too large. Concretely, let us assume that the channel corrupts t entries in $\vec{c}$, where an entry c_i of $\vec{c}$ is said to be *corrupted* when it is replaced by $x_i + e_i$, where e_i is a uniformly random element of $\mathbb{F}^*$. This corresponds to adding a t-sparse vector to $\vec{c}$. Because the Hamming distance between any two codewords is at least d, it is easy to see that whenever $t < d/2$, it is possible to uniquely decode a corrupted codeword $\vec{c}' = \vec{c} + \vec{e}$, by returning the message $\vec{x}$ corresponding to the element of $\mathcal{C}$ which is the closest to $\vec{c}'$. This is called "minimum distance decoding", and is equivalent to the maximum likelihood decoding (*i.e.* returning the most likely message given a codeword) whenever the channel corrupts strictly less than $d/2$ entries with high probability.

The above, however, only says that decoding is possible *in principle* whenever the amount of noise is not too large, but it says nothing about whether it is possible to *efficiently* decode a corrupted codeword: in general, finding the element of $\mathcal{C}$ closest to $\vec{c}'$ with a brute-force search could take up to $|\mathbb{F}|^t \cdot \binom{n}{t}$ time! Of course, brute force is far from the best strategy: starting with the seminal work of Prange in the sixties [Pra62], coding theorists have developed much better algorithms for decoding corrupted codewords over general linear codes. At a high level, these methods perform minimum distance decoding by computing $\vec{y} = H \cdot \vec{c}'$ of a corrupted codeword $\vec{c}' = \vec{c} + \vec{e}$, where H is a parity-check matrix of $\mathcal{C}$. Notice that

$$H \cdot \vec{c}' = H \cdot (\vec{c} + \vec{e}) = H \cdot \vec{e},$$

hence searching for a low-weight solution to the equation $H \cdot \vec{e} = \vec{y}$ enables retrieving the noise $\vec{e}$, and therefore the original codeword $\vec{c}$. The vector $\vec{y}$ is called a *syndrome* of $\vec{c}'$, and this approach to decoding is referred to as *syndrome decoding*.

However, as of today, all known syndrome decoding algorithms run in time exponential in t. There are even many concrete families of linear codes for which given a random code from the family, we do not know of any better way to decode corrupted codewords than using these generic exponential-time decoding methods. Coding theorists have long adapted to this bad news by designing concrete families of linear codes (with various useful features) that *do* admit efficient decoding algorithms. But the hardness of solving the general case, or that of finding an efficient decoding algorithm for many concrete families of linear codes, has stood for decades. And when something looks very hard to solve, we cryptographers like to formulate it as an assumption:

Definition 3.6.3 (Syndrome Decoding Assumption)**.** *Let $\mathbb{F}$ be a finite field, and $(t, k, n) = (t(\lambda), k(\lambda), n(\lambda))$ be polynomials in the security parameter λ with $k, t < n$. Let $m \leftarrow n - k$. Let $\mathcal{S}_t(\mathbb{F}^n)$ denote the set of all t-sparse vectors over $\mathbb{F}^n$. The syndrome decoding assumption over $\mathbb{F}$ with dimension k, codeword length n, and noise t, denoted (t, k, n)-$\mathsf{SD}(\mathbb{F})$, states that for every polynomial-time algorithm $\mathscr{A}$, it holds that*

$$\Pr[H \xleftarrow{\$} \mathbb{F}^{m \times n}, \vec{e} \xleftarrow{\$} \mathcal{S}_t(\mathbb{F}^n) \; : \; \mathscr{A}(H, H \cdot \vec{e}) = \vec{e}] \leq \mathsf{negl}(\lambda).$$

In the above, H is sampled uniformly at random. This induces the same distribution as picking a uniformly random generator matrix $G \xleftarrow{\$} \mathbb{F}^{n \times k}$ for a linear code, and setting H to be a random parity-check matrix of G. In other words, the assumption above states that syndrome decoding with parameters (t, k, n) is hard to solve for a random linear code.

Coding theorists and cryptographers have a good understanding of which parameters (t, k, n) yield a hard syndrome decoding problem. I will not provide a detailed overview of this aspect to avoid straying too much from our path: the extensive literature on the subject spans multiple research fields, including coding theory (through the information set decoding algorithms [Pra62; Ste88; FS09; BLP11; MMT11; BJMM12; MO15; EKM17; BM18], and more recently statistical decoding algorithms [Al 01; FKI06; Ove06; DT17; CDMT22]), constraint satisfaction problems [Fei02; Ale03; AIK06], learning theory (through the BKW algorithm and its variants [BKW00; Lyu05]) and cryptography/cryptanalysis [BFKL94; Wag02; Kir11] (this list gives only a short sample of pointers to the literature). For our target application to pseudorandom correlation generators, we will be more specifically interested in the following parameter setting: $n = \mathrm{poly}(\lambda)$, $k = c \cdot n$ for a constant c (typically $c \in [1/8, 1/2]$), and $t = \Theta(\lambda)$. In this regime of parameters, the best-known attacks run in time $2^{\Omega(\lambda)}$. We formulate this parameter setting as a concrete version of the syndrome decoding assumption below:

Definition 3.6.4 (Syndrome Decoding Assumption, concrete version)**.** *Let $\mathbb{F}$ be a finite field, $0 < c < 1$ be a constant, $n = \text{poly}(\lambda)$, $k = c \cdot n$, and $t = \Theta(\lambda)$. Let $m \leftarrow n - k$. The (t, k, n)-$\mathsf{SD}(\mathbb{F})$ problem is (T, ε)-hard if for every algorithm $\mathscr{A}$ running in time T, it holds that*

$$\Pr[H \xleftarrow{\$} \mathbb{F}^{m \times n}, \vec{e} \xleftarrow{\$} \mathcal{S}_t(\mathbb{F}^n) \ : \ \mathscr{A}(H, H \cdot \vec{e}) = \vec{e}] \leq \varepsilon.$$

Furthermore, the conjecture is believed to hold for $\varepsilon = \text{poly}(T) \cdot 2^{-\Omega(\lambda)}$.

3.6.3 Pseudorandomness from syndrome decoding

The (conjectured) hardness of syndrome decoding for random linear codes asserts that given $(H, H \cdot \vec{e})$, it is hard to recover the noise vector $\vec{e}$. For our application to PCGs, however, we want the mapping from (a short description of) $\vec{e}$ to $H \cdot \vec{e}$ to be a pseudorandom generator: that is, $H \cdot \vec{e}$ should be indistinguishable from random, even given $\vec{e}$. Fortunately, it turns out that this stronger assumption is implied by the syndrome decoding assumption (up to some loss in the parameters), a result that is commonly referred to as a search-to-decision reduction. Before we state it, let us formulate the decision version of the syndrome decoding problem (here, specialized to our parameter setting):

Definition 3.6.5 (Decisional Syndrome Decoding Assumption)**.** *Let $\mathbb{F}$ be a finite field, $0 < c < 1$ be a constant, $n = \text{poly}(\lambda)$, $k = c \cdot n$, and $t = \Theta(\lambda)$. Let $m \leftarrow n - k$. The (t, k, n)-$\mathsf{dSD}(\mathbb{F})$ problem is (T, ε)-hard if for every algorithm $\mathscr{A}$ running in time T, it holds that*

$$|\Pr[H \xleftarrow{\$} \mathbb{F}^{m \times n}, \vec{e} \xleftarrow{\$} \mathcal{S}_t(\mathbb{F}^n) \ : \ \mathscr{A}(H, H \cdot \vec{e})] - \Pr[H \xleftarrow{\$} \mathbb{F}^{m \times n}, \vec{x} \xleftarrow{\$} \mathbb{F}^m \ : \ \mathscr{A}(H, \vec{x})]| \leq \varepsilon.$$

Early search-to-decision reductions [BFKL94; KSS10] had an important loss in the parameters which would not be suitable for our applications. Fortunately, Applebaum, Ishai, and Kushilevitz [AIK09] gave a "sample preserving" search-to-decision reduction:

Lemma 3.6.6 ([AIK09])**.** *Assume that the (t, k, n)-$\mathsf{dSD}(\mathbb{F})$ problem is not $(\text{poly}(\lambda), \varepsilon)$-hard. Then the (t, k, n)-$\mathsf{SD}(\mathbb{F})$ problem is not $(\text{poly}(\lambda), \varepsilon^2/8)$-hard.*

The above lemma shows that, up to a polynomial loss in the runtime (and a quadratic loss in the advantage), the hardness of search and decisional versions of the syndrome decoding problem are equivalent. We note that in practice no faster algorithm is known to solve the decision version of the syndrome decoding problem compared to solving its search version.[9]

Corollary 3.6.7 (a PRG from syndrome decoding)**.** *Let $0 < c < 1$ be a constant and $n = \text{poly}(\lambda)$. Assume that the $(\lambda, c \cdot n, n)$-$\mathsf{SD}(\mathbb{F})$ problem is hard. Let $H \xleftarrow{\$} \mathbb{F}^{m \times n}$. Then the map $G_H : \{0,1\}^{\lambda \cdot (\log n + \log |\mathbb{F}^*|)} \mapsto \mathbb{F}^{n-k}$ which, on input the description of a λ-sparse vector $\vec{e}$ over $\mathbb{F}^n$ (given as a list of λ pairs $(i, x) \in [n] \times \mathbb{F}^*$), outputs $H \cdot \vec{e}$, is a pseudorandom generator.*

We note that technically, the above only works with high probability over the choice of H, and hides the use of a small probability lemma called the splitting lemma (which states, roughly, that if sampling $(H, H \cdot \vec{e})$ yields a hard instance of SD with high probability, then

[9] This is almost true, with a slight disclaimer: when the codeword length n becomes very large, the polynomial cost $\Omega(n^\omega)$ with $\omega \in [2.3, 3]$ of solving a system of linear equations (which is required in all algorithms for syndrome decoding) becomes significant, and it can be avoided when solving the decisional version.

when sampling H *once* and fixing it, it holds with high probability that H is *good*, in the sense that sampling $\vec{e}$ yields a hard instance $(H, H \cdot \vec{e})$ of SD with high probability over the choice of $\vec{e}$). We omit further details on these minor technicalities from now on.

Remark 3.6.8. *In this manuscript, we will cover several constructions which achieved improved efficiency guarantees by relying on variants of the syndrome decoding assumption, where the matrix H is not sampled from the uniform distribution over $\mathbb{F}^{m \times n}$. It should be noted that in general, known results on search-to-decision reductions do not carry over to other distributions for H, though some reductions have been established for some other choices of distributions. When using variants of the syndrome decoding assumption, we will therefore typically have to make the stronger decisional assumption.*

3.6.4 Wrapping-up: a PCG for low-degree correlations from syndrome decoding

And just like that, we're done! Combining this PRG from syndrome decoding with our FSS construction for the family of functions $F'_{H,\vec{e}}$ whose truth-table is $P(H \cdot \vec{e})$, where P is some multi-output low-degree multivariate polynomial, yields a PCG for any low-degree additive correlation.

Theorem 3.6.9. *Let $0 < c < 1$ be a constant, $t = \Theta(\lambda)$, and $n = \mathrm{poly}(\lambda)$. Assume that the $(\lambda, c \cdot n, n)$-$\mathsf{SD}(\mathbb{F})$ problem is hard, and a PRG $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda+2}$. Then for any degree-$d$ additive correlation described by a function C over $\mathbb{F}$, there exists a pseudorandom correlation generator for C with the following characteristics:*

- $\mathsf{PCG.Gen}(1^\lambda)$ *outputs keys* $(\mathsf{k}_0, \mathsf{k}_1)$ *of length* $O(\lambda \cdot t^d \cdot \log n)$*;*
- $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ *requires* $O(t^d \cdot n)$ *invocations of* PRG *and* $O(n^2)$ *arithmetic operations over* $\mathbb{F}$*, and outputs vectors of length* $m = (1 - c) \cdot n$*.*

Again, technically we only get a variant of a pseudorandom correlation generator *with setup*, where the setup algorithm samples the matrix H and passes it as a common reference string to all other algorithms (and security holds over the sampling of H as well).

A concrete example. Before we move on, let us look at a concrete example to get a better intuition: suppose the two parties want to generate $m/2$ Beaver triples – that is, additive shares of pseudorandom triples $(a, b, a \cdot b)$ over some field $\mathbb{F}$. Assume that the common parameters include the description of a matrix $H \in \mathbb{F}^{m \times n}$ (which has been sampled at random in some setup phase). The algorithm $\mathsf{PCG.Gen}$ proceeds as follows:

- It samples a uniformly random t-sparse vector $\vec{e}$ ($t = \Theta(\lambda)$ is a parameter, think $t \approx 50$ in case this is helpful). Let $(e_1, \cdots, e_t) \in \mathbb{F}^t$ are the nonzero entries of $\vec{e}$ and $(i_1, \cdots, i_t) \in [n]^t$ be their respective positions.
- It computes $\vec{e} \cdot \vec{e}^{\intercal}$ and writes it as a sum of t^2 unit $n \times n$ matrices over $\mathbb{F}$, where each matrix has a single nonzero entry $e_i \cdot e_j$ at position (i_i, i_j).
- It samples a pair of DPF keys for each point function $f_{(i_i,i_j),e_i \cdot e_j}$ over $[n^2]$. It also samples a pair of DPF keys for each point function f_{i_i,e_i} over $[n]$. It outputs all the DPF keys.

Now, given these $t^2 + t$ pairs of DPF keys, the parties can expand them as follows:

- Evaluate each DPF on its entire domain and sum all the results over $\mathbb{F}$. Note that this yields additive shares of the matrix $E = \vec{e} \cdot \vec{e}^\intercal$, together with shares of $\vec{e}$. Let E_0, E_1 denote the shares of E of the parties P_0, P_1 respectively, and $\vec{e}_0$, $\vec{e}_1$ denote their shares of $\vec{e}$.

- Each party P_σ computes $V_\sigma \leftarrow H \cdot E_\sigma \cdot H^\intercal$ and $\vec{v}_\sigma \leftarrow H \cdot \vec{e}_\sigma$. Note that $V_0 + V_1 = H \cdot (\vec{e} \cdot \vec{e}^\intercal) \cdot H^\intercal = (H \cdot \vec{e}) \cdot (H \cdot \vec{e})^\intercal$ and that $\vec{v}_0 + \vec{v}_1 = H \cdot \vec{e}$.

- Let us write $\vec{v} = H \cdot \vec{e}$ as a concatenation $\vec{a}||\vec{b}$ of two pseudorandom vectors $(\vec{a}, \vec{b}) \in \mathbb{F}^{m/2} \times \mathbb{F}^{m/2}$ (note that their pseudorandomness follows from the syndrome decoding assumption). For $i = 1$ tp $m/2$, the parties output shares of (a_i, b_i) together with shares of $a_i \cdot b_i$ (which can be "read" from their shares of $(\vec{a}||\vec{b}) \cdot (\vec{a}||\vec{b})^\intercal$).

Together, these $m/2$ shares of $(a_i, b_i, a_i \cdot b_i)$ form $m/2$ pseudorandom Beaver triples. For concrete parameters, the reader can imagine that $m/2$ is about 2^{30} and $n = 2m = 2^{32}$. Using $\lambda = 128$ and $t = 50$, the keys output by $\mathsf{PCG.Gen}$ have length $\approx 2t^2 \cdot \lambda \cdot \log n$ (ignoring low order terms), which amounts to 2.5 Megabytes of keys to generate 128 Megabytes worth of triples (here, I used $\mathbb{F} = \mathbb{F}_2$). The construction could have been slightly optimized further by generating $\vec{a} = H \cdot \vec{e}_a$ and $\vec{b} = H \cdot \vec{e}_b$ as two independent pseudorandom vectors (together with their tensor product) instead of viewing them as portions of a single pseudorandom vector $H \cdot \vec{e}$ (I kept the less efficient variant to remain closer to the general construction).

3.7 The Quest for the Right Code

The above numbers might sound attractive: we can generate 128MB worth of Beaver triples from keys of length 2.5MB. Indeed, using efficient protocols to generate these keys (which I have not described so far, but they exist), one gets silent two-party computation over $\mathbb{F}_2$ using a communication *much below* one bit per Beaver triples for the preprocessing phase – about three orders of magnitudes better than previous IKNP-style approaches to preprocessing, that required a few hundred bits of communication per Beaver triple.

But do not get fooled, wise reader: however appealing it might seem, the constructions I have described so far are *horribly* inefficient. The devil lies not in their communication cost (which is great), but in their *computation* cost. It helps here to think in terms of concrete numbers: to generate 2^{30} Beaver triples, the parties had to first compute shares of E, a matrix with $2^{32} \times 2^{32} = 2^{64}$ entries. We are talking here about *two million Terabytes* of storage. Even worse, computing V from E requires about $(2^{32})^{2.7} \approx 2^{86}$ operations, assuming you use Strassen's algorithm to implement the matrix multiplications (and ignoring the constants in Strassen) since asymptotically faster matrix multiplication algorithms have impractical constants. I cannot emphasize how inefficient performing 2^{86} operations is, except perhaps by saying that until quite recently, a cryptosystem that requires 2^{86} operations to be broken could have been deemed secure enough for most use cases.

The two related issues I just mentioned – these millions of Terabytes of storage and this horrendous computational cost – are quite distinct. The first issue, the $\Omega(n^2)$ storage cost, is independent of the matrix distribution H: it appears even before the parity-check matrix H is involved. Looking ahead, overcoming this issue has been done historically on a per-correlation basis: for several specific correlations of interests, it turns out that clever workarounds can be designed to reduce this $\Omega(n^2)$ cost to $O(n \cdot \log n)$, or even $O(n)$.[10] Sections 3.8 and 3.9 will be devoted to these clever workarounds. Even after solving this issue, however, the matrix multiplication cost remains: at the very least, the cost of generating shares of $C(G(r))$ has to scale with the cost of computing G, which itself involves multiplication by the $\Omega(n^2)$-sized matrix H. Overcoming this severe limitation of our template will be the focus of this section.

Let's jump straight ahead on the solution since there are no two ways about it: *we have to pick a different matrix H.* Concretely, the only thing we need from H is that $H \cdot \vec{e}$, where $\vec{e}$ is a sparse vector, cannot be distinguished from a random vector. Efficiency-wise, what we want from H is that the map $\vec{x} \to H \cdot \vec{x}$ can be computed fast – ideally much faster than the $\Omega(n^2)$ time required when H is a random matrix. When I discussed the syndrome decoding problem, I already hinted at the fact that its apparent hardness extends way beyond random linear codes: there are many concrete families of codes for which no fast syndrome decoding algorithms are known. And for some of them, the map $\vec{x} \to H \cdot \vec{x}$ can be computed much faster. Choosing the *right* map amounts to finding the code that strikes the best possible balance between the security of the associated syndrome decoding assumption, and the efficiency of the map $\vec{x} \to H \cdot \vec{x}$.

3.7.1 The linear test framework

Finding the right code is about finding the best balance between efficiency and security. But how do we know which codes are secure? The traditional answer is that the codes most likely to yield secure variants of syndrome decoding are those that have been studied the most by the community, and for which there is a general agreement of experts that the assumption is solid. However, the reasons that originally motivated the study of some codes – whether in coding theory or cryptography – were removed from the efficiency considerations we are interested in. This is especially true in the PCG setting, where the codeword length n is close to the target length of the correlated randomness we want to generate, which will often be way beyond 2^{20}. From the viewpoint of coding theory, or that of more traditional code-based cryptography, this is an extremely exotic parameter setting, and the design and analysis of standard codes have never been motivated by (or optimized for) this parameter setting. This creates an uncomfortable situation, where we have to choose between using well-studied codes, whose design has not been optimized for use with PCGs and which might be suboptimally efficient for this task, or using seemingly much better codes, but whose underlying assumption lacks extensive study from the research community.

In this section, I cover a folklore approach that provides a rule of thumb to decide whether a new candidate code yields a plausible variant of the syndrome decoding assumption. The ideas underlying this rule of thumb are far from new – they have their roots in the seminal work of Naor and Naor [NN90], and these ideas have been used on multiple occasions to provide arguments in favor of syndrome-decoding-style assumptions, including for example the work of [Zic17; ADI+17]. Earlier, it was also employed to study the pseudorandomness of other constructions, such as that of random local functions (see [MST03] and follow-ups).

In short, we know that avoiding short linear dependencies in the output is a necessary condition for the pseudorandomness of variants of syndrome decoding. Assuming that it is

[10] In fact, the chronology went the other way around: we first designed $O(n)$ solutions for a specific correlation – random oblivious transfers – before achieving an $O(n \cdot \log n)$ solution for other ones – OLEs and Beaver triples and observing that both were optimized instantiation of the generic but inefficient construction which I described in this writeup.

a *sufficient* condition for pseudorandomness appears to be a viable heuristic (though one that is known to fail in some settings, see Section 3.7.1.3). Formalizing the connection, as we will do afterwards, allows to quantify precisely the security one can hope to reach using this heuristic. Explicitly stating this framework has proven very fruitful in the past few years, and has been used to design and refine new linear codes tailored to PCG applications in several works [BCG+20a; CRR21a; BCG+22; CD23; BCCD23; RRT23].

3.7.1.1 Basic observation

Known attacks against the syndrome decoding assumption and its variants include attacks based on Gaussian elimination and the BKW algorithm [BKW00; Lyu05; LF06; EKM17] and variants based on covering codes [ZJW16; BV16; BTV16; GJL20], information set decoding attacks [Pra62; Ste88; FS09; BLP11; MMT11; BJMM12; MO15; EKM17; BM18], statistical decoding attacks [Al 01; FKI06; Ove06; DT17], generalized birthday attacks [Wag02; Kir11], linearization attacks [BM97; Saa07], or on finding correlations with low-degree polynomials [ABG+14; BR17]. While trying each known attack against each candidate new variant would be excessively cumbersome, the key observation is that all the above attacks (and more generally, essentially all known attacks against syndrome decoding and its variants) fit in a common framework, which I will call the linear test framework. Concretely, an attack against syndrome decoding in the linear test framework proceeds in two stages:

1. First, a matrix H is sampled, and fed to the (unbounded) adversary $\mathcal{A}$. The adversary returns a (nonzero) *test vector* $\vec{v} = \mathcal{A}(H)$.
2. Second, a noise vector $\vec{e}$ is sampled. The *advantage* of the adversary $\mathcal{A}$ in the linear test game is the bias of the induced distribution $\vec{v} \cdot H \cdot \vec{e}^\intercal$.

To formalize this notion, we recall the definition of the bias of a distribution:

Definition 3.7.1 (Bias of a Distribution)**.** *Given a distribution $\mathcal{D}$ over $\mathbb{F}^n$ and a vector $\vec{u} \in \mathbb{F}^n$, the bias of $\mathcal{D}$ with respect to $\vec{u}$, denoted $\mathsf{bias}_{\vec{u}}(\mathcal{D})$, is equal to*

$$\mathsf{bias}_{\vec{u}}(\mathcal{D}) = \left| \Pr_{\vec{x} \sim \mathcal{D}}[\vec{u} \cdot \vec{x}^\intercal = 0] - \Pr_{\vec{x} \sim \mathcal{U}_n}[\vec{u} \cdot \vec{x}^\intercal = 0] \right| = \left| \Pr_{\vec{x} \sim \mathcal{D}}[\vec{u} \cdot \vec{x}^\intercal = 0] - \frac{1}{|\mathbb{F}|} \right|,$$

where $\mathcal{U}_n$ denotes the uniform distribution over $\mathbb{F}^n$. The bias of $\mathcal{D}$, denoted $\mathsf{bias}(\mathcal{D})$, is the maximum bias of $\mathcal{D}$ with respect to any nonzero vector $\vec{u}$.

We say that an instance of the syndrome decoding problem is *secure against linear test* if, with very high probability over the sampling of H in step 1, for any possible adversarial choice of $\vec{v} = \mathcal{A}(H)$, the bias of $\vec{v} \cdot H \cdot \vec{e}^\intercal$ induced by the random sampling of $\vec{e}$ is negligible. Intuitively, the linear test framework captures any attack where the adversary is restricted to computing a linear function of the syndrome $\vec{b}^\intercal = H \cdot \vec{e}^\intercal$, but the choice of the linear function itself can depend arbitrarily on the code. Hence, the adversary is restricted in one dimension (it has to be linear in $\vec{b}^\intercal$), but can run in unbounded time given H.

Resistance against linear test is a property of both the code distribution (this is the “with high probability over the choice of H” part of the statement) and of the noise distribution (this is the “the bias of the distribution induced by the sampling of $\vec{e}$ is low” part of the statement). It turns out that to be secure against all attacks that fit in the linear test framework, it suffices to satisfy the following two simple conditions:

1. (good code) the code generated by $H^\intercal$ has a high minimum distance, and
2. (well-spread noise) for any large enough subset S of coordinates, with high probability over the choice of $\vec{e} \leftarrow \mathcal{D}$, at least one of the coordinates in S of $\vec{e}$ will be nonzero.

The above characterization works for any noise distribution whose nonzero entries are uniformly random over $\mathbb{F}^*$, which is the case for all standard choices of noise distributions. In particular, taking a random sparse vector $\vec{e}$ (as we discussed so far) suffices to satisfy the second condition. The first criterion is therefore the one we should focus on: to resist linear attacks, we need a code with a parity-check matrix H such that $H^\intercal$ generates a code with a large minimum distance. In other words, the code should be chosen such that its *dual code* has a high minimum distance.

To see why these conditions are sufficient, recall that the adversarial advantage is the bias of $\vec{v} \cdot H \cdot \vec{e}^\intercal$. By condition (2), if the subset S of nonzero entries of $\vec{v} \cdot H$ is sufficiently large, then $\vec{e}$ will "hit" one of these entries with large probabilities, and the output will be uniformly random. But the condition that S is sufficiently large translates precisely to the condition that $\vec{v} \cdot H$ has large Hamming weight for any possible (nonzero) vector $\vec{v}$, which is equivalent to saying that H generates a code with a large minimum distance. We recall the formalization below.

3.7.1.2 Formal framework

Formally, we will from now on be interested in variants of the syndrome decoding assumption – typically by changing the matrix H, but later on we will also consider changing the distribution of the noise $\vec{e}$. We, therefore, formulate a generalized variant of the (decisional) syndrome decoding assumption, tailored to a given choice of the matrix distribution (denoted $\mathcal{M}$) and the noise distribution (denoted $\mathcal{N}$):

Definition 3.7.2 (Generalized Decisional Syndrome Decoding Assumption)**.** *Let $\mathbb{F}$ be a finite field, $0 < c < 1$ be a constant, $n = \mathrm{poly}(\lambda)$, $k = c \cdot n$, $m \leftarrow n - k$, and $t = \Theta(\lambda)$. Let $\mathcal{M} = \mathcal{M}_{n,m}(\mathbb{F})$ be a distribution over $\mathbb{F}^{m \times n}$. Let $\mathcal{N} = \mathcal{N}_{n,t}(\mathbb{F})$ denote a distribution over $\mathbb{F}^n$ with $\mathbb{E}_{\vec{e} \sim \mathcal{N}}[\mathsf{HW}(\vec{e})] = t$. The $(\mathcal{M}, \mathcal{N})$-$\mathsf{dGSD}(\mathbb{F})$ problem is (T, ε)-hard if for every algorithm $\mathscr{A}$ running in time T, it holds that*

$$|\Pr[H \xleftarrow{\$} \mathcal{M}, \vec{e} \xleftarrow{\$} \mathcal{N} \; : \; \mathscr{A}(H, H \cdot \vec{e})] - \Pr[H \xleftarrow{\$} \mathcal{M}, \vec{x} \xleftarrow{\$} \mathbb{F}^m \; : \; \mathscr{A}(H, \vec{x})]| \leq \varepsilon.$$

Our goal is now to characterize which choices of $\mathcal{M}, \mathcal{N}$ yield plausible flavors of the generalized syndrome decoding assumption. To this end, we formally define below security in the *linear test framework*.

Definition 3.7.3 (Security against Linear Test)**.** *Let $\mathbb{F}$ be an arbitrary finite field, and let $\mathcal{N} = \mathcal{N}_{n,t}(\mathbb{F})$ denote a noise distributions over $\mathbb{F}^n$. Let $\mathcal{M} = \mathcal{M}_{n,m}(\mathbb{F})$ be a distribution over $\mathbb{F}^{m \times n}$. Let $\varepsilon, \delta : \mathbb{N} \mapsto [0,1]$ be two functions. We say that the $(\mathcal{M}, \mathcal{N})$-$\mathsf{dGSD}(\mathbb{F})$ assumption with dimension k and codeword length n is* (ε, δ)-secure against linear tests *if for any (possibly inefficient) adversary $\mathcal{A}$ which, on input a matrix $H \in \mathbb{F}^{m \times n}$, outputs a nonzero $\vec{v} \in \mathbb{F}^n$, it holds that*

$$\Pr[H \xleftarrow{\$} \mathcal{M}, \vec{v} \xleftarrow{\$} \mathcal{A}(A) \; : \; \mathsf{bias}_{\vec{v}}(\mathcal{D}_H) \geq \varepsilon(\lambda)] \leq \delta(\lambda),$$

where $\mathcal{D}_H$ denotes the distribution induced by sampling $\vec{e} \leftarrow \mathcal{N}$, and outputting the syndrome $H \cdot \vec{e}$.

Then, we have the following straightforward lemma:

Lemma 3.7.4. *Let $\mathbb{F}$ be an arbitrary finite field, and let $\mathcal{N} = \mathcal{N}_{n,t}(\mathbb{F})$ denote a noise distributions over $\mathbb{F}^n$. Let $\mathcal{M} = \mathcal{M}_{n,m}(\mathbb{F})$ be a distribution over $\mathbb{F}^{m\times n}$. Then for any integer $d \in \mathbb{N}$, the $(\mathcal{M},\mathcal{N})$-$\mathsf{dGSD}(\mathbb{F})$ problem is (ε_d, η_d)-secure against linear tests, where*

$$\varepsilon_d = \max_{\mathsf{HW}(\vec{u})>d} \mathsf{bias}_{\vec{u}}(\mathcal{N}), \qquad \text{and} \qquad \eta_d = \Pr_{H \overset{\$}{\leftarrow} \mathcal{M}}[\mathsf{d}(H) \geq d],$$

where $\mathsf{d}(H)$ denotes the minimum distance of the code generated by the columns of H.

The proof is straightforward: fix any integer d. Then with probability at least δ_d, $\mathsf{d}(H) \geq d$. Consider any (possibly unbounded) adversary $\mathcal{A}$ outputting $\vec{v} \neq \vec{0}$. If $\mathsf{d}(H) \geq d$, denoting $\vec{u} = H \cdot \vec{v}$, it holds that $\mathsf{HW}(\vec{u}) > d$, in which case the advantage of the adversary is bounded by $\max_{\vec{v}\neq\vec{0}} \mathsf{bias}_{\Rightarrow v}(\mathcal{D}_H) \geq \varepsilon_d = \max_{\mathsf{HW}(\vec{u})>d} \mathsf{bias}_{\vec{u}}(\mathcal{N})$. We leave it as an interesting exercise for the reader to apply the above lemma to evaluate the security of syndrome decoding against linear tests for random linear codes, with random t-sparse noise.

3.7.1.3 When security against linear attacks does not suffice

There are two important cases where security against linear test does not yield security against *all* attacks.

1. When the code is strongly algebraic. For example, Reed-Solomon codes, which have a strong algebraic structure, have high dual minimum distance but can be decoded efficiently with the Welch–Berlekamp algorithm, hence they do not lead to a secure syndrome decoding instance (and indeed, Welch–Berlekamp does not fit in the linear test framework).

2. When the noise is structured (e.g. for regular noise) and the code length is at least quadratic in the dimension. This opens the door to algebraic attacks such as the Arora-Ge attack [AG11]. However, when $n = O(k)$ (which is the case we consider), the Arora-Ge attack does not apply.

The above are, as of today, the only known cases where security against linear attacks is known to be insufficient. Algebraic decoding techniques have a long history and are only known for very restricted families of codes, and the Arora-Ge type of attack typically never applies in the $n = O(k)$ regime which we usually consider for PCGs. Therefore, a reasonable rule of thumb is that a variant of syndrome decoding yields a plausible assumption if (1) it probably resists linear attacks, and (2) finding an algebraic decoding algorithm is a longstanding open problem.

3.7.2 The chronology

Finding out which code is the best fit has been a long journey. Initially, in [BCGI18], we suggested relying on LDPC codes: the variant of the syndrome decoding assumption where H is set to be the parity-check matrix of a random LDPC code (*i.e.* a code with a random sparse generator matrix) dates back to the work of Alekhnovich in 2003 [Ale03] and has withstood the test of time so far. Furthermore, there is extensive literature on the fast encoding of the dual of LDPC codes (which, by a principle known as the transposition principle, implies a fast $\vec{x} \to H \cdot \vec{x}$ mapping). In later works [BCG+19a], we switched back to quasi-cyclic codes, for which optimized implementations based on the fast Fourier transform had been designed, and which were generating a lot of interest as a basis for efficient variants of the syndrome decoding assumption in the context of submissions to the NIST post-quantum competitions (e.g. in [ABB+20]).

In [CRR21a], we advocated a more aggressive choice, building a new concrete linear code highly optimized for correlated randomness generation and where the minimum distance was heuristically estimated using extensive computer simulations, but we stressed that the lack of a provable analysis of the minimum distance required a lot of caution – and indeed, the minimum distance of these codes was later shown to scale poorly to large dimensions (which the computer simulations could not reach) in [RRT23]. In [BCG+22], we introduced Expand-Accumulate code, a new type of LDPC code with a very efficient "online-offline" mapping (concretely, computing $\vec{x} \to H \cdot \vec{x}$ boils down to one very fast "offline" accumulation step on $\vec{x}$, followed by an "on-the-fly" sparse mapping) and strong provable guarantees on their minimum distance. In a recent work [RRT23], building upon [BCG+22], the authors introduced Expand-Convolute codes, a generalization of Expand-Accumulate codes which retain their fast mapping properties, but achieve even better parameters for the minimum distance.

The latest proposals exhibit impressive efficiency performances, where computing $\vec{x} \to H \cdot \vec{x}$ is done in time $O(n)$, with small constants, yet their minimum distance guarantees are essentially as good as that of random linear codes. But are they the best possible? This remains an intriguing and well-motivated question, as any progress in the area yields faster protocols for silent secure computation. As of today, there remain several strong candidates to be investigated, with very appealing efficiency properties, and the potential to exhibit good minimum distance properties. The quest continues!

3.8 PCGs for Oblivious Transfers

So far, our focus has been on (1) providing a general framework for pseudorandom correlation generators (with an instantiation for low-degree additive correlations, under the syndrome decoding assumption) and (2) explaining the selection process of the right flavor of the syndrome decoding assumption to make the mapping $\vec{x} \to H \cdot \vec{x}$. If the reader managed to remember everything we said so far[11], they might remember that making this mapping efficient was only half of the problem: the other, more fundamental problem is that generating a degree-2 correlation in the coordinates of $\vec{v} = H \cdot \vec{e}$ (which is required in all correlations discussed so far in this writeup, such as random OTs, OLEs, and Beaver triples) requires computing the tensor product $\vec{v} \cdot \vec{v}^\intercal$, which takes time (and space) quadratic in the target amount of correlated randomness.

Fortunately, for all correlations listed so far, it turned out that this cost can be circumvented via dedicated constructions. This was first observed in [BCG+19b] for the OT correlation, then a year later in [BCG+20b] for the OLE and Beaver triple correlations over a large field $\mathbb{F}$. Eventually, we recently filled the remaining gap, with a construction for OLEs and Beaver triples over all fields $\mathbb{F}$ with $|\mathbb{F}| > 2$ in a recent work [BCCD23]. In this section, I will focus on the case of the oblivious transfer correlation. The construction described below evolved throughout three papers [BCGI18; BCG+19b; BCG+19a].

[11] If that's really the case, congrats!

3.8.1 A PCG for subfield vector-OLE

We start by describing a PCG for a specific correlation, called the subfield vector-OLE correlation. The OLE correlation distributes many pairs (a_i, b_i) together with additive shares of $a_i \cdot b_i$ over $\mathbb{F}$ to the parties. The vector-OLE (or VOLE) correlation is a variant of the OLE correlation that restricts one of the values, say, the b_i's, to be a single global value x. That is, instead of receiving $(\vec{a}, \vec{b})$ and shares of $\vec{a} \odot \vec{b}$ (where $\odot$ denotes the component-wise or Schur, product), the parties receive $\vec{a}, x$ respectively (where x is a single $\mathbb{F}$-element) together with shares of $x \cdot \vec{a}$. The subfield vector-OLE (or sVOLE) is another variant of the vector-OLE correlation, where $\vec{a}$ is sampled from a subfield $\mathbb{F}'$ of $\mathbb{F}$.

An equivalent way to formulate the *subfield vector-OLE* correlation is as follows: fix $\mathbb{F} = \mathbb{F}_q$ and a subfield $\mathbb{F}_p$ of $\mathbb{F}$ (with $q = p^r$). The correlation samples $(\vec{u}, \vec{v}_0) \xleftarrow{\$} \mathbb{F}_p^n \times \mathbb{F}^n$ and $x \xleftarrow{\$} \mathbb{F}$. It outputs $(\vec{u}, \vec{v}_0)$ to one party P_0, usually called the sender, and $(x, \vec{v}_1 = \vec{u} \cdot x + \vec{v}_0)$ to the second party P_1, usually called the receiver.

From Section 3.5.3, we know that (assuming a length-doubling PRG) there exists an FSS scheme for the function class $\mathcal{F} = \{F_{H,\vec{e}}\}$ with leakage $\mathsf{Leak}(F_{H,\vec{e}}) = (t, \mathbb{F}, H)$, with key size $O(t \cdot \lambda \cdot \log n)$, where $F_{H,\vec{e}} : [m] \mapsto \mathbb{F}$ is a function which, given a matrix $H \in \mathbb{F}^{m \times n}$ and a t-sparse vector $\vec{e} \in \mathbb{F}^n$ and on input $i \in [m]$, output the i-th entry of $H \cdot \vec{e}$. The core observation underlying the design of an efficient PCG for this correlation is that the multiplication by an arbitrary field element x is for free in this construction: simply use FSS for the function $F_{H,x\cdot\vec{e}}$. Everything works out because $x \cdot \vec{e}$ is still a t-sparse vector, and because $H \cdot (x \cdot \vec{e}) = x \cdot H \cdot \vec{e}$. The construction also works identically when $H, \vec{e}$ are sampled over the subfield $\mathbb{F}_p$ while x is sampled from the field $\mathbb{F}$. The full construction is given in Figure 3.4.

Plugging the construction of DPF from Section 3.5.2.2 in the construction of Figure 3.4, we get:

Theorem 3.8.1. *Fix a security parameter λ, a field $\mathbb{F} = \mathbb{F}_q$, and a subfield $\mathbb{F}_p$ of $\mathbb{F}$ (with $q = p^r$). Let m be the target length of the subfield vector-OLE correlation, and $n = m + k$. Let t be a noise parameter. Let $\mathcal{M} = \mathcal{M}_{n,m}(\mathbb{F}_p)$ be a distribution over $\mathbb{F}_p^{m \times n}$. Let $\mathcal{N} = \mathcal{N}_{n,t}(\mathbb{F}_p)$ denote the distribution of random weight-t vectors over $\mathbb{F}_p^n$ Assume the hardness of the $(\mathcal{M}, \mathcal{N})$-$\mathsf{dGSD}(\mathbb{F}_p)$ problem, and that there exists a pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda+2}$. Then there exists a PCG for the subfield vector-OLE correlation over $(\mathbb{F}_p, \mathbb{F})$ with key size $O(\lambda \cdot t \cdot \log n)$ and where the Expand algorithm boils down to $2 \cdot t \cdot m$ calls to PRG and one computation of the mapping $\vec{v} \to H \cdot \vec{v}$.*

Correctness follows from the fact that $\vec{v}_0 - \vec{v}_1 = H \cdot \sum_{j=1}^{t} (\vec{y}_0^j + \vec{y}_1^j) = H \cdot (x \cdot \vec{e}) = x \cdot H \cdot \vec{e} = x \cdot \vec{u}$ by linearity and correctness of the DPF schemes (the $(-1)^\sigma$ term in the construction is just used to get subtractive secret sharing, in line with how we define subfield VOLE). Security follows from the security of the DPF scheme (here again, we omit a formal security analysis – the interested reader can check [BCGI18]).

3.8.2 From Subfield VOLE to OT

We now explain how to turn a PCG for the subfield VOLE correlation into a PCG for the random OT correlation. At the heart of this transformation is a technique from [IKNP03] which relies on the observation that a VOLE correlation can be seen, by reversing the roles of the sender and the receiver, as an OT correlation with correlated OTs. Then, the correlation

Parameters. Fix a security parameter λ, a field $\mathbb{F} = \mathbb{F}_q$, and a subfield $\mathbb{F}_p$ of $\mathbb{F}$ (with $q = p^r$). Let m be the target length of the subfield vector-OLE correlation, and $n = m + k$ (in concrete instances, it typically suffices to have $k = m = n/2$). Let t be a noise parameter (typically $t = \Theta(\lambda)$). Let $\mathcal{M} = \mathcal{M}_{n,m}(\mathbb{F}_p)$ be a distribution over $\mathbb{F}_p^{m \times n}$ (choices of the matrix distribution are discussed in Section 3.7). Fix a matrix $H \xleftarrow{\$} \mathcal{M}$. Let $\mathcal{N} = \mathcal{N}_{n,t}(\mathbb{F}_p)$ denote the distribution of random weight-t vectors over $\mathbb{F}_p^n$ (we will discuss alternative distributions later).

Gen. Sample $x \xleftarrow{\$} \mathbb{F}_q$, $\vec{e} \xleftarrow{\$} \mathcal{N}$ and let $(i_1, \cdots, i_t), (e_1, \cdots, e_t)$ denote the positions of the nonzero entries of $\vec{e}$ and their values respectively. For $j = 1$ to t, sample $(\mathsf{k}_0^j, \mathsf{k}_1^j) \leftarrow \mathsf{DPF.Gen}(1^\lambda, f_{i_j, x \cdot e_j})$ (where $f_{i_j, x \cdot e_j} : [n] \mapsto \mathbb{F}_q$ denotes the point function which evaluates to $x \cdot e_j$ on i_j). Output $\mathsf{k}_0 = (i_j, e_j, \mathsf{k}_0^j)_{j \leq t}$ and $\mathsf{k}_1 = (x, (\mathsf{k}_1^j)_{j \leq t})$.

Expand. Let σ be the party index. For $j = 1$ to t, compute $\vec{y}_\sigma^{\,j} \leftarrow \mathsf{DPF.FullEval}(\sigma, \mathsf{k}_\sigma^j)$ and set $\vec{y}_\sigma \leftarrow \sum_{j=1}^t \vec{y}_\sigma^{\,j}$.

- If $\sigma = 0$, set $v_0 \leftarrow H \cdot y_0$, reconstruct $\vec{e}$ from $(i_1, \cdots, i_t), (e_1, \cdots, e_t)$ and output $(\vec{u} = H \cdot \vec{e}, \vec{v}_0)$.
- If $\sigma = 1$, set $v_1 \leftarrow -H \cdot y_1$, and output $(x, \vec{v}_1)$.

Figure 3.4: Construction of a PCG for the subfield vector-OLE correlation under the generalized syndrome decoding assumption

among the OT instances can be broken using a hash function satisfying a suitable security notion, called *correlation-robustness.*

3.8.2.1 Reversing the roles of the parties

Fix $p = 2$ and $r = \lambda$, so that $\mathbb{F} = \mathbb{F}_{2^\lambda}$ is an extension field of $\mathbb{F}_2$. When instantiating the construction of Figure 3.4 with these fields, the sender receives $(\vec{u}, \vec{v}_0) \in \mathbb{F}_2^m \times \mathbb{F}_2^m$ and the receiver receives $(x, \vec{v}_1 = \vec{u} \cdot x + \vec{v}_0) \in \mathbb{F}_2^m \times \mathbb{F}_2^m$. Since $\vec{u} \in \mathbb{F}_2^m$, we can alternatively view this construction the other way around: reversing the roles of the receiver and the sender, observe that for any $i \leq m$, it holds that $v_{0,i} = v_{1,i}$ if $u_i = 0$, and $v_{0,i} = v_{1,i} + x$ otherwise. Hence, we can view this construction as distributing m *correlated* oblivious transfers between a sender with correlated pairs of inputs $(v_{1,i}, v_{1,i} - x)$, where the $v_{1,i}$ are pseudorandom, and a receiver with pseudorandom selection bits u_i.

3.8.2.2 Breaking correlations with a correlation-robust hash

Let $\mathsf{H} : \mathbb{F}_{2^\lambda} \mapsto \{0,1\}^\lambda$ be a hash function. In the last step of the transformation, the sender computes $r^i \leftarrow \mathsf{H}(v_{0,i})$ and the receiver computes $(s_0^i, s_1^i) \leftarrow (\mathsf{H}(v_{1,i}), \mathsf{H}(v_{1,i} + x))$ for $i = 1$ to m. The tuples (u_i, r^i) and (s_0^i, s_1^i) satisfy the OT correlation, and by the security of the subfield VOLE, the u_i's are pseudorandom. To finish the proof, it remains to argue that even knowing the values $v_{0,i}$ for $i = 1$ to m, the values $\mathsf{H}(v_{0,i} + x)$ should all look *simultaneously* *random* to the receiver, where x is a truly random element of $\mathbb{F}_{2^\lambda}$. This is precisely what the notion of correlation-robust hashing captures:

Definition 3.8.2 (Correlation-Robust Hash [IKNP03])**.** *A hash function* $\mathsf{H} : \mathbb{F}_{2^\lambda} \mapsto \{0,1\}^\lambda$ *is* m*-correlation-robust if for* every *sequence* $(v^1, \cdots, v^m)$ *of distinct elements of* $\mathbb{F}_{2^\lambda}$*, the following distributions are indistinguishable:*

$$\mathcal{D}_0 = \{\mathsf{H}(v^1 + x), \cdots, \mathsf{H}(v^m + x) \mid x \xleftarrow{\$} \mathbb{F}_{2^\lambda}\}, \qquad \textit{and} \qquad \mathcal{D}_1 = \mathcal{U}_\lambda^n,$$

where $\mathcal{U}_\lambda$ *denotes the uniform distribution over* $\{0,1\}^\lambda$*.*

In our application to OT from sVOLE, the requirement that the v^i's are all distinct holds with overwhelming probability $\approx m \cdot 2^{-\lambda}$. Plugging the construction of Figure 3.4 into this transformation, we get:

Theorem 3.8.3. *Fix a security parameter* λ*. Let* m *be the target number of pseudorandom OTs, and* $n = m + k$*. Let* t *be a noise parameter. Let* $\mathcal{M} = \mathcal{M}_{n,m}(\mathbb{F}_2)$ *be a distribution over* $\mathbb{F}_2^{m\times n}$*. Let* $\mathcal{N} = \mathcal{N}_{n,t}(\mathbb{F}_2)$ *denote the distribution of random weight-*t *vectors over* $\mathbb{F}_2^n$ *Assuming the hardness of the* $(\mathcal{M}, \mathcal{N})$*-*$\mathsf{dGSD}(\mathbb{F}_2)$ *problem and using the following tools:*

- *a pseudorandom generator* $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda+2}$*,*
- *an* m*-correlation-robust hash function* $\mathsf{H} : \mathbb{F}_{2^\lambda} \mapsto \{0,1\}^\lambda$*,*

there exists a PCG for the OT correlation (with λ*-bit sender messages) with key size* $O(\lambda \cdot t \cdot \log n)$ *and where the* Expand *algorithm is dominated by* $2 \cdot t \cdot n$ *calls to* PRG*, one computation of the mapping* $\vec{v} \to H \cdot \vec{v}$ *(where* $H \in \mathbb{F}_2^{m\times n}$ *denotes the parity-check matrix sampled from* $\mathcal{M}$*), and* m *calls to* H *for the receiver (resp.* $2m$ *for the sender).*

We note that unlike our generic construction of PCG for low-degree additive correlations, the construction captured by Theorem 3.8.3 is really efficient: if an appropriate parity-check matrix H is chosen (such that there is a linear-time algorithm computing the mapping $\vec{v} \to H \cdot \vec{v}$), the computation boils down to $O(m)$ arithmetic operations, $2n$ calls to a PRG, and $\approx 1.5m$ calls to a correlation-robust hash function. Over modern hardware, a standard practice is to instantiate both the PRG and the hash function using the AES block cipher (with a fixed key), to benefit from the Intel AES-NI set of hardware instructions for AES. These are insanely fast: about 20 CPU cycles to encrypt 128 bits [MSY21]. With such hardware support, the construction enables the generation of hundreds of thousands of OTs per second on one core of a standard machine, and the key size for 2^{30} OTs is below 40kB. While these are impressive numbers, we can do even better! In the next two sections, we cover two standard ways to improve the construction of PCG for OTs, which have been described respectively in [BCGI18] and [BCG+19a].

3.8.3 Improvement I: using a regular noise distribution

Here, we consider a variant of the construction where the noise distribution is changed, from random t-sparse vectors to random *regular* vectors. For simplicity, assume that t divides n. We let $\mathcal{N} = \mathcal{N}_{n,t}(\mathbb{F}_2)$ denote the following distribution: $\mathcal{N}$ samples t uniformly random unit (*i.e.* weight-1) vectors over $\mathbb{F}_2^{n/t}$, and outputs their concatenation. Compared to random t-sparse vectors, this corresponds to dividing the noise vector $\vec{e}$ into t equal-length blocks and adding a single random noisy coordinate to each block.

3.8.3.1 Security considerations

The first and most important aspect to deal with is whether this change hurts security. Changing the noise distribution to regular noise amounts to changing the syndrome decoding assumption; the variant of syndrome decoding with regular noise is called *regular syndrome decoding*. I will not cover extensively the impact of using this noise distribution in the cryptanalysis of syndrome decoding. But let me just mention a few important points:

- Regular syndrome decoding is not a new assumption: it is a well-established variant of syndrome decoding whose introduction dates back to 2003, where it was introduced in [AFS03] as the assumption underlying the FSB candidate to the NIST hash function competition. It was subsequently analyzed in [FGS07; MDCY11; BLPS11; HOSS18], among others (of course, it was also used and analyzed in many works on silent secure computation, but they are posterior to the first use of regular noise for PCGs).

- Regularity is a two-edged sword for cryptanalysis: it reduces the search space (which improves syndrome decoding algorithms), but puts constraints on the shape of the solution (which makes it more difficult to satisfy).[12] As of today, for the range of parameters of interest in PCG applications, there are no known syndrome decoding algorithms that perform better against regular syndrome decoding compared to "traditional" syndrome decoding.[13]

- Eventually, it is easy to check that the regular noise distribution satisfies the criteria outlined in Lemma 3.7.4: for every large enough subset S of coordinates, with very high probability over the choice of $\vec{e} \stackrel{\$}{\leftarrow} \mathcal{N}$, at least one of the coordinates in S of $\vec{e}$ will be nonzero. A quick calculation shows that this probability is asymptotically *identical* for regular noise and the more standard t-sparse noise: from the viewpoint of linear tests, the regular noise distribution offers the same security as the random t-sparse noise.

3.8.3.2 Efficiency considerations

Let us now switch to *why* we want to use regular noise instead of random t-sparse noise. Remember that to succinctly share the noise vector $\vec{e}$, we wrote $\vec{e}$ as a sum of t unit vectors of length m and interpreted each of these unit vectors as the truth table of a point function with domain $[n]$, which we can share with a DPF of size $O(\lambda \cdot \log m)$. Switching to regular noise makes things much easier: now, $\vec{e}$ is directly a *concatenation* of t unit vectors of length m/t each, and each of the blocks of $\vec{e}$ can therefore be shared with a DPF of size $O(\lambda \cdot \log(m/t))$.

This already yields a slight improvement in the key length (from $O(t \cdot \lambda \cdot \log m)$ to $O(t \cdot \lambda \cdot \log(m/t)))$, but the most important effect is on *computation*. When representing $\vec{e}$ as a sum of t unit vector, the PCG.Expand algorithm had to run FullEval t times in parallel on an m-sized domain, requiring about $2tn$ calls to the PRG. Instead, when $\vec{e}$ is a concatenation

[12] This statement holds when solving the *search* RSD problem, but not necessarily when solving its decisional counterpart. However, as of today the fastest known attacks on decisional RSD rely on information-set decoding [ES23] and actually solve the search variant.

[13] There is a very recent exception in the work of [BØ23] which tailors algebraic cryptanalysis techniques to the regular syndrome decoding problem. However, they have a very moderate impact on their security and only apply to very specific parameter settings used in so-called "primal" PCG constructions, which I did not cover at all in this manuscript and are typically less efficient than the approach I cover here.

of t unit vectors, the $\mathsf{PCG.Expand}$ algorithm runs $\mathsf{FullEval}$ t times in parallel on an n/t-sized domain, which requires $2t(n/t) = 2n$ calls to the PRG: a t-fold reduction of the number of calls to our PRG! When using a flavor of syndrome decoding where the mapping $\vec{v} \to H \cdot \vec{v}$ is very efficient (as modern constructions do), calling the PRG $2tn$ times is by far the dominant cost of the computation, and using a regular noise yields one to two orders of magnitude of improvement.

3.8.4 Improvement II: using a puncturable pseudorandom function

Upon closer inspection, using FSS to compress shares of $x \cdot \vec{e}$ in our PCG construction for sVOLE is an overkill. Indeed, FSS guarantees that $x \cdot \vec{e}$ remains hidden to both parties, but the construction reveals x to the receiver and $\vec{e}$ to the sender anyway. It would suffice to rely on a variant of DPF for point functions $f_{\alpha,\beta}$ where the *position* α is revealed to one party (the sender, since it corresponds to a position of a nonzero entry in $\vec{e}$) and the *value* β is revealed to the other party (the receiver, since it corresponds to x). It turns out that this functionality can be achieved using a primitive known as a puncturable pseudorandom function.

3.8.4.1 Definition of PPRFs

Pseudorandom functions (PRF), introduced in [GGM86], are keyed functions that are indistinguishable from truly random functions. More formally,

Definition 3.8.4. *A pseudorandom function (PRF) with key space* $\{0,1\}^\lambda$*, domain* $[n]$*, and range* $\mathbb{F}_{2^\lambda}$ *is a keyed family of functions* $\{F_K\}_{K \in \{0,1\}^\lambda}$ *such that for every polynomial-time adversary* $\mathcal{A}$*, it holds that*

$$\left| \Pr_{F \xleftarrow{\$} \mathcal{F}}[\mathcal{A}^F(1^\lambda) = 1] - \Pr_{K \xleftarrow{\$} \{0,1\}^\lambda}[\mathcal{A}^{F_K}(1^\lambda) = 1] \right| \leq \mathsf{negl}(\lambda),$$

where $\mathcal{F}$ *denotes the space of all functions from* $[n]$ *to* $\mathbb{F}_{2^\lambda}$*, and* $\mathcal{A}^F$ *indicates that* $\mathcal{A}$ *is given oracle access to* F*. If the above holds for adversaries* $\mathcal{A}$ *restricted to querying only uniformly random inputs* $x \xleftarrow{\$} [n]$ *to the oracle, we say that* F_K *is a* weak *pseudorandom function (WPRF).*

A puncturable pseudorandom function (PPRF) 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 and does not reveal any information about the value $F.\mathsf{Eval}(k, x)$. PPRFs have been introduced in [KPTZ13; BW13; BGI14]. Below, we directly define PPRFs for the range of parameters of interest in our construction of PCG for OTs.

Definition 3.8.5 (t-Puncturable Pseudorandom Function)**.** *A puncturable pseudorandom function (PPRF) with key space* $\{0,1\}^\lambda$*, domain* $[n]$*, and range* $\mathbb{F}_{2^\lambda}$*, is a pseudorandom function* F *with three probabilistic polynomial-time algorithms* $(F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ *such that*

- $F.\mathsf{KeyGen}(1^\lambda)$ *outputs a random key* $K \in \{0,1\}^\lambda$*,*
- $F.\mathsf{Puncture}(K, \{S\})$*, on input a ley* $K \in \{0,1\}^\lambda$*, and a subset* $S \subset [n]$ *of size* t*, outputs a punctured key* $K\{S\}$*,*

Experiment $\mathsf{Exp\text{-}PPRF}$

Setup Phase. The adversary $\mathcal{A}$ sends a size-t subset $S^* \in [n]$ to the challenger. When it receives S^*, the challenger picks $K \xleftarrow{\$} F.\mathsf{KeyGen}(1^\lambda)$ and a random bit $b \xleftarrow{\$} \{0,1\}$.

Challenge Phase. The challenger sends $K\{S^*\} \leftarrow F.\mathsf{Puncture}(K, S^*)$ to $\mathcal{A}$. If $b = 0$, the challenger additionally sends $(F(K,x))_{x\in S^*}$ to $\mathcal{A}$; otherwise, if $b = 1$, the challenger picks t random values ($y_x \xleftarrow{\$} \mathbb{F}_{2^\lambda}$ for every $x \in S^*$) and sends them to $\mathcal{A}$.

Figure 3.5: Selective security game for puncturable pseudorandom functions. At the end of the experiment, $\mathcal{A}$ sends a guess b' and wins if $b' = b$.

- $F.\mathsf{Eval}(K\{S\}, x)$, *on input a key* $K\{S\}$ *punctured at all points in* S, *and a point* x, *outputs* $F(K,x)$ *if* $x \notin S$, *and* $\perp$ *otherwise,*

such that no probabilistic polynomial-time adversary wins the experiment $\mathsf{Exp\text{-}PPRF}$ *represented in Figure 3.5 with a non-negligible advantage over the random guess.*

3.8.4.2 Puncturable PRFs from length-doubling PRGs

Here, we recall how a PPRF can be constructed from any length-doubling pseudorandom generator G, using the GGM tree-based construction [GGM86; KPTZ13; BW13; BGI14]. For simplicity, assume that n is a power of two; se write $n = 2^d$. The construction proceeds as follows: On input a key $K \in \{0,1\}^\lambda$ and a point $x \in [n]$ (viewed as a bitstring $x_1 \cdots x_d \in \{0,1\}^d$), set $K^{(0)} \leftarrow K$ and perform the following iterative evaluation procedure: for $i = 1$ to $\ell \leftarrow d$, compute $(K_0^{(i)}, K_1^{(i)}) \leftarrow G(K^{(i-1)})$, and set $K^{(i)} \leftarrow K_{x_i}^{(i)}$. Output $K^{(\ell)}$. This procedure creates a complete binary tree with edges labeled by keys and 2^d leaves. The output of the PRF on an input x is the key labeling the leaf at the end of the path defined by x from the root of the tree.

- $F.\mathsf{KeyGen}(1^\lambda)$: output a random seed for G.
- $F.\mathsf{Puncture}(K, z)$: on input a key $K \in \{0,1\}^\lambda$ and a point $x = x_1 \cdots x_d$, for $i = 1$ to $\ell \leftarrow d$, compute $(K_0^{(i)}, K_1^{(i)}) \leftarrow G(K^{(i-1)})$, and set $K^{(i)} \leftarrow K_{x_i}^{(i)}$. Return $K\{x\} = (K_{1-x_1}^{(1)}, \ldots, K_{1-x_\ell}^{(\ell)})$.
- $F.\mathsf{Eval}(K\{x\}, x')$, on input a punctured key $K\{x\}$ and a point x, if $x = x'$, output $\perp$. Otherwise, parse $K\{x\}$ as $(K_{1-x_1}^{(1)}, \ldots, K_{1-x_\ell}^{(\ell)})$ and start the iterative evaluation procedure from the first $K_{1-x_i}^{(i)}$ such that $x'_i = 1 - x_i$.

To obtain a t-puncturable PRF with input domain $[n]$, one can simply run t instances of the above PPRF and set the output of the PRF to be the bitwise xor of the output of each instance; when the t punctured points are in distinct n/t-sized blocks (as in our construction based on regular syndrome decoding), it suffices instead to use a separate PPRF on domain $[n/t]$ for each block. With this construction, the length of a key punctured at t points is $t\lambda \log n$ (resp. $t\lambda \log(n/t)$), where λ is the seed size of the PRG.

3.8.4.3 Sharing the noise vector using a PPRF

Let $F = (F.\mathsf{KeyGen}, F.\mathsf{Puncture}, F.\mathsf{Eval})$ be a PPRF with domain $[n/t]$. When evaluating t instances of this PPRF on its full domain (with keys $K_1, \cdots, K_t$), the receiver gets a t-fold concatenation of length-n/t pseudorandom vectors. Furthermore, denoting $i_1, \cdots, i_t \in [n/t]$ the nonzero entries of $\vec{e}$, we can reveal the entire pseudorandom vector *except for the t positions indexed by the i_j's* to the sender. For the remaining t positions, we simply give to the sender the values $F(K_j, i_j) + x$ for $j = 1$ to t, and the sender places each value at position i_j in the j-th block of the pseudorandom vector.

The two vectors obtained this way form pseudorandom shares of a length-m vector over $\mathbb{F}_{2^\lambda}$ which is equal to 0 everywhere, except the entry i_j of each block j, where it is equal to x. This is the same as saying that the vector form shares of $x \cdot \vec{e}$. Hence, we achieved the same result as with FSS for point functions but using a much simpler construction, leveraging the fact that the sender knows the positions i_j of the nonzero entries of $\vec{e}$. Beyond the simplicity of the construction, it has two main advantages:

- A smaller receiver key. In this construction, the key of the receiver is compressed to $t \cdot \lambda$ bits (to store the PPRF keys $(K_1, \cdots, K_t)$), which can be further compressed to just λ bits by generating them from a short seed using a pseudorandom generator. This reduces the size of the entire receiver key to be as small as 128 bits (the sender key is also slightly reduced, but of size comparable to its size in the FSS-based construction).

- A better key distribution protocol. This touches on an aspect that we have not discussed so far, and won't have the space to discuss much, but briefly: when using PCGs to do secure computation in the silent preprocessing model, the parties need to distributively and securely generate the keys of the PCG using an interactive protocol (see Figure 3.2.2). It turns out that the PPRF-based construction of PCG for OT has a much better key distribution protocol, which uses only two rounds of interaction and $t \cdot \log(n/t)$ OT instances. The low number of rounds is a major advantage over high latency networks, but also has the independent advantage of enabling various useful forms of *non-interactive secure computation*, an area which I won't discuss any further here – but the interested reader should check [BCG+19a] to read more about it!

3.8.5 Concrete efficiency

Using all the improvements discussed above, we get:

Theorem 3.8.6. *Fix a security parameter λ. Let m be the target number of pseudorandom OTs, and $n = m + k$. Let t be a noise parameter. Let $\mathcal{M} = \mathcal{M}_{n,m}(\mathbb{F}_2)$ be a distribution over $\mathbb{F}_2^{m\times n}$. Let $\mathcal{N} = \mathcal{N}_{n,t}(\mathbb{F}_2)$ denote the distribution of random t-regular vectors over $\mathbb{F}_2^n$. Then, assuming the hardness of the $(\mathcal{M}, \mathcal{N})$-$\mathsf{dGSD}(\mathbb{F}_2)$ problem and the following tools:*

- *a pseudorandom generator* $\mathsf{PRG} : \{0,1\}^\lambda \mapsto \{0,1\}^{2\lambda+2}$,
- *an m-correlation-robust hash function* $\mathsf{H} : \mathbb{F}_{2^\lambda} \mapsto \{0,1\}^\lambda$,

there exists a PCG for the OT correlation (with λ-bit sender messages) with sender key size $\lambda \cdot t \cdot \log(n/t)$, receiver key size λ, and where the cost of the Expand *algorithm is dominated by $2 \cdot n$ calls to* PRG*, one computation of the mapping $\vec{v} \to H \cdot \vec{v}$ (where $H \in \mathbb{F}_2^{m\times n}$ denotes the parity-check matrix sampled from $\mathcal{M}$), and m calls to* H *for the receiver (resp. $2m$ for the sender).*

To give some perspective on this theorem, let us derive a few concrete numbers. The exact choice of t depends on the concrete cryptanalysis of the syndrome decoding assumption with the parameters used in the construction. Among many works on the subject, the recent work of [LWYY22] provides precise security estimations tailored to the parameters used in PCG constructions. Concretely, to generate $m = 2^{22}$ instances of λ-bit OTs and setting $n = 4m$, they estimate that using a value as low as $t = 34$ suffices to get 128 bits of security (see Table 5 in their work). With these numbers, setting $\lambda = 128$ yields a sender key size of about 10kB (and 16 Bytes for the receiver key).

As for computation, instantiating the PRG and the hash using the AES block cipher, it boils down to $\approx 2^{25}$ calls to AES[14] and one computation of the mapping $\vec{v} \to H \cdot \vec{v}$. Concretely, modern implementations can generate about 10 million pseudorandom λ-bit OTs per second with these approaches, on one core of a standard laptop (using more cores multiplies the number of OTs since everything parallelizes optimally).

3.9 Beyond Oblivious Transfers: PCGs for Complex Correlations

The construction of PCG for OT described in Section 3.8 circumvents the (impractical) $\Omega(n^2)$ computational cost of the generic PCG construction (for degree-2 additive correlations) using a two-step construction that first produces a subfield VOLE correlation, and then uses a correlation-robust hash function to turn it into an OT correlation. Unfortunately, this approach is strictly tailored to the OT correlation. Even though OT is essentially just an OLE over $\mathbb{F}_2$, the approach does not extend to OLEs over larger fields. More precisely, the subfield VOLE part of the construction extends easily (after all, we directly described over general fields), but the hashing technique does not generalize. To make it clear, let us look at the case of $\mathbb{F}_3$: here, the receiver would hold some $u_i \in \mathbb{F}_3$, and one of $(v_i, v_i + x, v_i - x) = (v_i, z_i, t_i)$. However, hashing these values does not yield an OLE over $\mathbb{F}_3$: it yields a 1-out-of-3 OT. Getting an OLE would require preserving the property that $z_i + t_i = 2v_i$ – but of course, hashing breaks this correlation as well!

As of today, we do not know of a construction for OLEs with performances as impressive as those we have for OTs. The case of OLEs is known to be much harder to handle – we have had efficient OT extension protocols for more than two decades [IKNP03], and no efficient OLE extension protocols (while OLE extensions have been demonstrated in [ADI+17] in the semi-honest setting, their concrete efficiency remains unclear). However, we can still do much better than paying the unaffordable $\Omega(n^2)$ cost of the general solution, by relying on a variant of the syndrome decoding problem with considerably more algebraic structure.

3.9.1 High level intuition

The key insight of the construction is that some structured degree-2 correlations can be obtained without computing the full tensor product of $H \cdot \vec{e}$ with itself. Instead, the idea will be to use a structured matrix H to embed $H \cdot \vec{e}$ into a polynomial ring. Over this ring, the tensor product can be replaced by a polynomial product, which can be computed in time $O(n \cdot \log^2 n)$, or even just $O(n \cdot \log n)$ when using cyclotomic rings. Then, the construction leverages the fact that if the polynomial splits entirely, the product of two polynomials can

[14] Recent works [BCG+22; GYW+23] show how to further halve this cost, but I won't cover these recent techniques here.

be projected back to the component-wise product of the two pseudorandom vectors they encode, which will exactly yield the OLE correlation.

3.9.2 A PCG for ring-OLE

We start by explaining how to compress the generation of a *single* OLE correlation over a large polynomial ring $\mathcal{R} = \mathbb{F}_p[X]/(P)$, where P is some degree-n split polynomial (*i.e.*, a polynomial that splits into n linear factors), and $\mathbb{F}_p$ is a finite field. Let us call ring-OLE this correlation:

Definition 3.9.1. *In a ring-OLE correlation each party P_σ receives $(x_\sigma, y_\sigma) \in \mathcal{R}^2$ for $\sigma = 0, 1$, which are random conditioned on $x_0 + x_1 = y_0 \cdot y_1$. Given an integer t, a t-sparse ring-OLE correlation is a correlation in which each party P_σ receives $(x_\sigma, y_\sigma) \in \mathcal{R}^2$ for $\sigma = 0, 1$, where y_0, y_1 are random t-sparse polynomials over $\mathcal{R}$, and x_0, x_1 random conditioned on $x_0 + x_1 = y_0 \cdot y_1$.*

Looking ahead, the degree n of the polynomial will translate afterwards to the number of OLEs produced over the base field $\mathbb{F}_p$. The construction follows our FSS-based template and proceeds in two steps: we show how to use FSS to obtain a PCG for t-sparse ring-OLE correlations, and then use a syndrome decoding assumption to turn it into a PCG for the ring-OLE correlation.

3.9.2.1 Distributing sparse ring-OLE correlations

First, using an FSS for multipoint functions allows to succinctly distribute a *sparse* ring-OLE correlation: sample two random t-sparse polynomials y_0, y_1 (*i.e.* polynomials with t nonzero coefficients in the standard basis), and define f to be the t^2-point function whose truth table is the coefficients of $y_0 \cdot y_1$ (over $\mathbb{F}_p[X]$ – *i.e.*, without reduction modulo P). Each party P_σ receives $\mathsf{k}_\sigma = (y_\sigma, f_\sigma)$, where $(f_0, f_1) = \mathsf{FSS.Gen}(f)$. With the construction of multipoint FSS discussed in Section 3.5.3, the size of k_σ is $O(t^2 \cdot \lambda \cdot \log n)$.

3.9.2.2 Distributing ring-OLE correlations

Then, to turn this sparse ring-OLE correlation into a pseudorandom ring-OLE correlation, we will compress two instances of the correlation using the mapping $(u, v) \to (u + a \cdot v)$, where $a \overset{\$}{\leftarrow} \mathcal{R}$ is a random (fixed) polynomial. Looking ahead, this will correspond to relying on the security of a variant of the syndrome decoding assumption over a polynomial ring, with a random systematic-form parity-check matrix $(1, a)$. Let $(x^0_\sigma, y^0_\sigma)_{\sigma\in\{0,1\}}$ and $(x^1_\sigma, y^1_\sigma)_{\sigma\in\{0,1\}}$ be two instances of a sparse ring-OLE correlation. Fix a random element $a \overset{\$}{\leftarrow} \mathcal{R}$. Each party P_σ defines

$$y_\sigma \leftarrow (1, a) \cdot (y^0_\sigma, y^1_\sigma)^\intercal = y^0_\sigma + a \cdot y^1_\sigma \bmod P(X).$$

The assumption that y_σ is indistinguishable from random is the *ring syndrome decoding* assumption. Using FFT, the mapping can be computed in $O(n \cdot \log^2 n)$ $\mathbb{F}_p$-operations over general rings (or even $O(n \cdot \log n)$ over cyclotomic rings), where n is the degree of the polynomials. Then, observe that we have

$$y_0 y_1 = (y^0_0 + a \cdot y^1_0) \cdot (y^0_1 + a \cdot y^1_1) = y^0_0 \cdot y^0_1 + a \cdot (y^0_1 \cdot y^1_0 + y^1_0 \cdot y^1_1) + a^2 \cdot (y^1_0 \cdot y^1_1),$$

where the polynomials $y_0^0 \cdot y_1^0, y_1^0 \cdot y_0^1, y_0^1 \cdot y_1^1$, and $y_0^1 \cdot y_1^1$ are all t^2-sparse. Hence, each of these four polynomials can be succinctly shared using FSS for a t^2-point function. Therefore, shares of $y_0 y_1$ can be reconstructed using a local linear combination of shares of sparse polynomials with public coefficients $(1, a, a^2)$, and the shares of the sparse polynomials can be distributed succinctly using FSS for multipoint functions.

Wrapping up. The final PCG looks as follows: each party P_σ gets (y_σ^0, y_σ^1) together with four FSS shares of t^2-point functions whose domain corresponds to these four terms. The PCG key size scales as $O(\lambda t^2 \log n)$ overall, where n is the degree of the polynomials over $\mathcal{R}$. Expanding the keys amounts to locally computing the shares of the sparse polynomial products (four evaluations of the FSS on their entire domain, using $O(t \cdot n)$ calls to a pseudorandom generator, or even $O(n)$ when relying on the regular ring syndrome decoding assumption – see Section 3.8.3 for the discussions on regular noise) and a few polynomial multiplications (computed using $\tilde{O}(n)$ arithmetic operations) with a and a^2 (which are public parameters).

3.9.3 From ring-OLE to OLEs

When P splits into n linear factors over $\mathbb{F}_p[X]$ (which is, in particular, the case over a suitable cyclotomic ring), a single pseudorandom ring-OLE correlation as above can be locally transformed into n instances of pseudorandom OLEs over $\mathbb{F}_p$, by evaluating the polynomials on n fixed distinct points. Furthermore, this entire evaluation can be performed in time $O(n \cdot \log n)$ using fast multipoint polynomial evaluation. This yields a PCG for generating n instances of the OLE correlation, with keys of size $O(\lambda t^2 \log n)$, and runtime $O(n \cdot \log n)$ when assuming the regular syndrome decoding assumption over a cyclotomic ring where P splits entirely.

This construction is significantly less efficient than our PCG for OT: its key size has a t^2 factor instead of a factor t, and the computation scales as $O(n \cdot \log n)$ instead of n. In practice, this translates to a construction with a key size in the $2 \sim 10$ Megabytes range for generating 2^{20} to 2^{35} OLEs. As for runtime, implementations using FFTs over cyclotomic rings can generate about a million OLEs in 10 seconds over one core of a standard laptop. This is quite efficient, but still two orders of magnitude slower than the runtime achieved by fast PCGs for OTs.

3.9.4 From OLEs to other correlations

Using the same reduction as in Section 2.7.2.2, a PCG for OLEs immediately yields a PCG for Beaver triples (with a key size twice larger). Furthermore, given any element $\Delta \in \mathbb{F}$, the protocol from the previous section can be easily modified to yield shares of a Beaver triple $(\vec{a}, \vec{b}, \vec{a} \odot \vec{b})$ together with *authenticated* shares $(\Delta \cdot \vec{a}, \Delta \cdot \vec{b}, \Delta \cdot \vec{a} \odot \vec{b})$ of this Beaver triple. This variant is particularly useful because, while Beaver triple is the natural resource to use in the semi-honest secure computation of arithmetic circuits in the correlated randomness model, their authenticated counterpart is the natural resource that enables efficient malicious secure computation of arithmetic circuits in the correlated randomness model. We refer the reader to [BCG+20b] for more details on these generalizations.

3.9.5 Historical notes

The study of the ring syndrome decoding assumption dates back to [HKL+12] and has received some (limited) attention from the cryptography community [BL12; DP12; LP15; GJL15]. For an appropriate choice of the polynomial P, it is also equivalent to the well-studied quasi-cyclic syndrome decoding assumption, used in NIST submissions such as BIKE [ABB+17] and HQC [MBD+18].

The construction sketched in this section was introduced in [BCG+20b]. Because it requires P to split entirely over $\mathbb{F}$, it is restricted to OLEs over a large field $\mathbb{F}$ of size $|\mathbb{F}| \geq n$. We finally circumvented this limitation in a recent work [BCCD23], achieving PCGs for OLEs (with comparable efficiency features) for all fields of size $|\mathbb{F}| > 2$. The construction is more mathematically involved, using quasi-abelian codes defined over group algebras, but follows a similar template (and additionally benefits from a heuristic security argument in the linear test framework).

One could think that this is the full picture, as OLEs over $\mathbb{F}_2$ are just OTs, for which we already have efficient constructions. However, that is not entirely true: our constructions of OLEs satisfy an additional and very useful feature called "programmability", which is fundamental for silent secure computation either in the malicious setting or when more than two parties are involved. Known constructions of PCG for OT do not satisfy programmability. Extending [BCCD23] to work over $\mathbb{F}_2$ would yield efficient *programmable* PCGs for OTs, which remains an intriguing and fascinating open question as of today.

3.10 Beyond PCGs: Pseudorandom Correlation Functions

In Section 3.2.1.2 and 3.2.1.3, I draw a parallel between secure computation in the correlated randomness model and secure communication via the one-time pad: in both cases, pre-distributed random strings satisfying some appropriate relations enable an information-theoretic realization of the target functionality. Then, I explained how pseudorandom generators (or stream ciphers) can be used to compress the one-time pad, yielding efficient secure communication from short keys, to motivate the search for a similar primitive targetting secure computation, which led us to pseudorandom correlation generators.

A few sections later, here we are, equipped with super-efficient PCGs for the OT correlation, and with relatively efficient PCGs for the OLE correlation (and more). However, in a sense that I will make precise very soon, these constructions do not fully mimic the features of stream-cipher-based protocols for secure communication. A useful feature of secure communication protocols is that, after securely exchanging a short key, two parties can *continuously* communicate. Concretely, this means that at any time, the two parties can exchange messages, without having to decide ahead of time on a bound on the amount of communication or to store long keys in advance for later use.

In practice, this can done by replacing the pseudorandom generator in the construction of Section 3.2.1.2 by a pseudorandom function F_K: following their one-time generation of a shared random key $K \in \{0,1\}^\lambda$, the two parties can publicly agree on a unique nonce x for the session and use the sequence $F_K(x), F_K(x+1), F_K(x+2), \cdots$ of pseudorandom strings to mask messages of arbitrary length. This allows the parties, given only K, to generate on-the-fly and upon need arbitrarily long shared pseudorandom strings to use in the one-time pad protocol. Alternatively, this can also be achieved using a simple PRG in *stream cipher mode*: after using $G(K) = (k_0, \cdots, k_n)$ to generate n λ-bit pseudorandom strings (assuming

G is a PRG from $\{0,1\}^\lambda$ to $\{0,1\}^{(n+1)\cdot\lambda}$), the parties can use $(k_1, \cdots, k_n)$ to mask an $n \cdot \lambda$-bit message M, and keep k_0 as a new "fresh" PRG seed to be used in the next communication session.

Switching back to PCGs, it turns out that there are no obvious incremental approaches to silent secure computation using any of the constructions we described so far. A natural approach, similar to the re-keying approach for secure communication that we just sketched, would be to use a portion of the pseudorandom correlations generated so far to run a secure computation protocol for re-generating fresh PCG seeds. However, unlike with secure communication, this approach requires the participants to interact online before they can compute their fresh PCG seeds. In practice, this means that whenever two participants execute a secure computation, they would have to anticipate a bound on the amount of correlations they will need in their next interaction (of course, they can always pick a very large bound, but this comes at a strong cost as we will see next). A much better solution would allow them to decide the amount of correlated randomness they need on-the-fly, without having to interact first.

unfortunately, all the PCGs we constructed come with a major limitation: The expansion of the correlated seeds is an "all-or-nothing" procedure, where the target correlation is produced *all at once* without enabling fast random access to the long output. Furthermore, existing PCG constructions do not even support the type of (stateful) incremental evaluation enabled by standard PRGs in stream-cipher mode, because the correlation of the PCG outputs is distinct from the correlation between the PCG keys (while for PRG, the correlation is preserved: equal seeds yield equal pseudorandom strings). This limits the use of PCGs to a monolithic form of silent preprocessing that requires parties to generate one for all and to store big amounts of correlated randomness that they might want to use in the future, or to re-interact to generate fresh PCG seeds whenever they anticipate that they might need to run a secure protocol.

3.10.1 Defining pseudorandom correlation functions

Without further ado, we introduce the notion of pseudorandom correlation functions to capture the ability to perform silent secure computation protocols where, after a one-time short interaction, the parties can on-demand and on-the-fly generate an a-priori arbitrary amount of pseudorandom correlated strings. A PCF is a pair $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ of algorithms where

- $\mathsf{PCF.Gen}$ outputs a pair of short correlated keys $(\mathsf{k}_0, \mathsf{k}_1)$,
- $\mathsf{PCF.Eval}(\mathsf{k}_\sigma, x)$ is a deterministic polynomial-time algorithm.

We require the following properties of a PCF: First, the joint distribution of outputs of Eval has to be computationally indistinguishable from truly random outputs satisfying the target correlation. Second, we require that for $\sigma = 0, 1$, when given the key $\mathsf{k}_{1-\sigma}$, the outputs of $\mathsf{PCF.Eval}(\mathsf{k}_\sigma, \cdot)$ are indistinguishable from random outputs of the target correlation when sampled conditioned on the partial outputs from $\mathsf{PCF.Eval}(\mathsf{k}_{1-\sigma}, x)$.

Preliminaries. To formally define a PCF, we use the notion of reverse-sampleable correlation generator given in Definition 3.3.2. However, we will use a slightly different semantic: when building PCGs, we were typically considering the correlation generator $\mathcal{C}$ as generating directly, on input 1^n, an "n-fold" instance of a target correlation (e.g., n copies of an OLE).

Here, we will think of our correlation generator $\mathcal{Y}$ as outputting "single-instances" of a target correlation (e.g., a single OLE), and we give it a different name to clarify the semantic. We will also assume a slightly generalized syntax for correlation generators compared with Definition 3.3.1:

Definition 3.10.1 (Reverse-samplable correlation generators, modified)**.** *We define a reverse-samplable correlation generator $\mathcal{Y}$ as in Definitions 3.3.1 and 3.3.2, except that we assume that $\mathcal{Y}(1^\lambda)$ returns a pair of string in $\{0,1\}^{\tau_0(\lambda)} \times \{0,1\}^{\tau_1(\lambda)}$ (rather than $\{0,1\}^\lambda \times \{0,1\}^\lambda$), where τ_0, τ_1 are output-length functions, and pass 1^λ as additional argument to* $\mathsf{RSample}$ *(to cover the case where $\tau_\sigma(\lambda)$ is much smaller than λ).*

Note also that the definition of reverse-samplable correlation generators does not capture correlations with a global parameter which remains the same across samples, as in the sVOLE correlation. This is captured by the notion of reverse sampleable correlation with setup, which allows all algorithms to depend on a fixed global secret, ensuring consistency across different invocations, formally defined in [BCG+20a] (we omit it there to avoid overcomplicating an already cumbersome definition).

Definition. We introduce the formal definition of PCFs for a reverse-sampleable correlation generator $\mathcal{Y}$ in Definition 3.10.2. The experiments for the two properties of a PCF, pseudorandom $\mathcal{Y}$-correlated outputs and internal security, are given on Figure 3.6 and Figure 3.7 respectively. In these experiments, N denotes a bound on the number of evaluations of the PCF made by the adversary $\mathscr{A}$, and the bit b distinguishes between the real-world experiment ($b = 0$) and the simulated experiment ($b = 1$). The definition below captures a notion of weak pseudorandom functions, which are secure when all queries to the function are uniformly random.

Definition 3.10.2 (Pseudorandom correlation function (PCF))**.** *Let $\mathcal{Y}$ be a reverse-sampleable correlation with output length functions $\tau_0(\lambda), \tau_1(\lambda)$ and let $\lambda \leq n(\lambda) \leq \mathrm{poly}(\lambda)$ be an input length function. A PCF* ($\mathsf{PCF.Gen}, \mathsf{PCF.Eval}$) *is a pair of algorithms with the following syntax:*

- $\mathsf{PCF.Gen}(1^\lambda)$ *is a probabilistic polynomial time algorithm that on input 1^λ, outputs a pair of keys* $(\mathsf{k}_0, \mathsf{k}_1)$*; we assume that λ can be inferred from the keys.*

- $\mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x)$ *is a deterministic polynomial-time algorithm that on input $\sigma \in \{0,1\}$, key k_σ and input value $x \in \{0,1\}^{n(\lambda)}$, outputs a value $y_\sigma \in \{0,1\}^{\tau_\sigma(\lambda)}$.*

Let N denote a bound on the number of PCF evaluations, B denote a bound on the size of the adversaries, and ε a bound on their success probability. We say ($\mathsf{PCF.Gen}, \mathsf{PCF.Eval}$) *is a (weak) (N, B, ε)*-secure pseudorandom correlation function (PCF) for $\mathcal{Y}$, *if the following conditions hold:*

- ***Pseudorandom $\mathcal{Y}$-correlated outputs.*** *For every $\sigma \in \{0,1\}$ and non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$, it holds*

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

  for all sufficiently large λ, where $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,b}(\lambda)$ for $b \in \{0,1\}$ is as defined in Figure 3.6. In particular, the adversary is given access to $N(\lambda)$ samples.

```
$\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda)$ :
for $i = 1$ to $N(\lambda)$:
    $x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
    $(y_0^{(i)}, y_1^{(i)}) \leftarrow \mathcal{Y}(1^\lambda)$
$b \leftarrow \mathcal{A}(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i\in[N(\lambda)]})$
return $b$
```

```
$\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda)$ :
$(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
for $i = 1$ to $N(\lambda)$:
    $x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
    for $\sigma \in \{0,1\}$:
        $y_\sigma^{(i)} \leftarrow \mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x^{(i)})$
$b \leftarrow \mathcal{A}(1^\lambda, (x^{(i)}, y_0^{(i)}, y_1^{(i)})_{i\in[N(\lambda)]})$
return $b$
```

Figure 3.6: Pseudorandom $\mathcal{Y}$-correlated outputs of a PCF.

```
$\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda)$ :
$(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
for $i = 1$ to $N(\lambda)$:
    $x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
    $y_{1-\sigma}^{(i)} \leftarrow \mathsf{PCF.Eval}(1-\sigma, \mathsf{k}_{1-\sigma}, x^{(i)})$
$b \leftarrow \mathcal{A}(1^\lambda, \sigma, k_\sigma, (x^{(i)}, y_{1-\sigma}^{(i)})_{i\in[N(\lambda)]})$
return $b$
```

```
$\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda)$ :
$(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCF.Gen}(1^\lambda)$
for $i = 1$ to $N(\lambda)$:
    $x^{(i)} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
    $y_\sigma^{(i)} \leftarrow \mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x^{(i)})$
    $y_{1-\sigma}^{(i)} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_\sigma^{(i)})$
$b \leftarrow \mathcal{A}(1^\lambda, \sigma, k_\sigma, (x^{(i)}, y_{1-\sigma}^{(i)})_{i\in[N(\lambda)]})$
return $b$
```

Figure 3.7: Internal security of a PCF. Here, $\mathsf{RSample}$ is the algorithm for reverse sampling $\mathcal{Y}$ as in Definition 3.3.2.

- ***Internal security.*** *For each $\sigma \in \{0,1\}$ and non-uniform adversary $\mathcal{A}$ of size $B(\lambda)$, it holds*

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

  for all sufficiently large λ, where $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,b}(\lambda)$ for $b \in \{0,1\}$ is as defined in Figure 3.7 (again, with $N(\lambda)$ samples).

We say that $(\mathsf{PCF.Gen}, \mathsf{PCF.Eval})$ is a PCF for $\mathcal{Y}$ *if it is a $(p, 1/p, p)$-secure PCF for $\mathcal{Y}$ for every polynomial p. If $B = N$, we will write (B, ε)-secure PCF for short.*

The definition above can be strengthened to the notion of strong PCFs. Informally, a strong PCF is defined as in Definition 3.10.2, except that in the experiments for pseudorandom $\mathcal{Y}$-correlated outputs and internal security (given on Figure 3.6 and Figure 3.7), the PCF inputs $x^{(i)}$ are not sampled uniformly from $\{0,1\}^{n(\lambda)}$. Instead, they are adaptively chosen by the adversary $\mathscr{A}$ which is given oracle access to $\mathsf{PCF.Eval}(1-\sigma, \mathsf{k}_{1-\sigma}, \cdot)$ for up to N queries. We focus here on weak PCFs for two reasons:

- The constructions we will describe in this write-up are only weak PCFs. This is somewhat inherent to the limitations of our template, we will elaborate on this later on.
- Any weak PCF can be heuristically transformed into a strong PCF by hashing the input before feeding it to the function. This heuristic transformation is provably secure if the hash function is modeled as a (programmable) random oracle.

Let me clarify of the distinction between weak and strong PCFs plays a role in secure computation. When using (weak) PCFs to generate N instances of a pseudorandom correlation for a secure computation protocol, the two parties must (without interaction) agree on the N random inputs $x^{(1)}, \cdots, x^{(N)}$. Concretely, this amounts to assuming access to a very long common reference string. A strong PCF, on the other hand, allows to generate the correlations using an arbitrary sequence of distinct inputs (typically a counter, setting $x^{(i)} = i$ for $i = 1, \cdots N$), which does not require any setup assumption. Then, the general transformation of a weak PCF into a strong PCF via a hash function modeled as a random oracle becomes equivalent to the well-known fact that a random oracle H can be used to setup a long common random string $(x^{(1)}, \cdots, x^{(N)})$ by setting $x^{(i)} \leftarrow \mathsf{H}(i)$ for $i = 1$ to N.

Equipped with a proper definition of PCFs, we now turn our attention to the question of constructing them. So, let us go back to our template, shall we?

3.10.2 A Template for pseudorandom correlation functions

Earlier in this manuscript (in Section 3.4.2.3), we introduced a template for PCGs for an additive correlation C which combines a pseudorandom generator G with a function secret sharing for the class of functions $C \circ G_k$, where $G_k(i)$ returns the i-th component of $G(k)$. Adapting this template to PCFs is straightforward: to get a PCF for an additive correlation C, we combine a PRF $F = \{F_K\}_{K \in \{0,1\}^*}$ with an FSS scheme for the class of functions $\mathcal{F} = \{C \circ F_K\}_{K \in \{0,1\}^*}$. That's it!

Theorem 3.10.3. *For any $\lambda \in \mathbb{N}$, let $C_\lambda : \mathbb{F}_{2^\lambda} \mapsto \mathbb{G}_\lambda$ be a function whose range is an Abelian group $\mathbb{G}_\lambda$, and let $m = m(\lambda) \in \mathrm{poly}(\lambda)$. Let $F = \{F_{K_\lambda} : [m] \mapsto \mathbb{F}_{2^\lambda}\}_{\lambda \in \mathbb{N}, K_\lambda \in \{0,1\}^\lambda}$ be a PRF with key space $\{0,1\}^\lambda$. Let $\mathcal{F} = \{C_\lambda \circ F_{K_\lambda} : [m] \mapsto \mathbb{G}_\lambda\}_{\lambda \in \mathbb{N}, K_\lambda \in \{0,1\}^\lambda}$. Let $(\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ be a 2-party function secret sharing for the class of functions $\mathcal{F}$. Consider the following construction:*

- $\mathsf{PCF.Gen}(1^\lambda)$: *sample $K \xleftarrow{\$} \{0,1\}^\lambda$ and output $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{FSS.Gen}(1^\lambda, C_\lambda \circ F_K)$*
- $\mathsf{PCG.Eval}(\sigma, \mathsf{k}_\sigma, x)$: *on input $x \in [m]$, return $\mathsf{FSS.Eval}(\sigma, \mathsf{k}_\sigma, x)$.*

The construction $(\mathsf{PCG.Gen}, \mathsf{PCG.Eval})$ is a secure PCF for the two-party additive correlation of length 1 defined by C_λ.

In the above theorem, we view C and F as public information, and the PRF key K_λ as the secret description of $C_\lambda \circ F_{K_\lambda}$. We remind the reader than an additive correlation of length 1 is a correlation defined by a function C_λ that outputs additive shares of $C_\lambda(r)$ to the parties, where r is a random input (see Definition 3.4.1). The proof of security proceeds through a sequence of straightforward game hops. Fix a party index σ:

1. First, using the secrecy property of the FSS, replace the key k_σ in the experiment $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda)$ with a simulated key $\mathsf{k}_\sigma \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{Leak}(C \circ F_K))$, where here $\mathsf{Leak}(C_\lambda \circ F_{K_\lambda})$ returns (m, C_λ, F). Observe that from this game onward, k_σ does not depend on the PRF key K_λ anymore.

2. Using the pseudorandomness of the PRF F_{K_λ}, replace each evaluation of F_{K_λ} made in $\mathsf{PCF.Eval}(1-\sigma, \mathsf{k}_{1-\sigma}, x)$ by a call to a uniformly random function.

3. At this stage, the samples $y_{1-\sigma}^{(i)}$ from the correlation are truly random. Replace the sampling of $(y_0^{(i)}, y_1^{(i)})$ by a sampling of $y_\sigma^{(i)}$ followed by $y_{1-\sigma}^{(i)} \leftarrow \mathsf{RSample}(1^\lambda, \sigma, y_\sigma^{(i)})$. This last game corresponds exactly to the experiment $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda)$.

For a detailed formal proof, we refer the reader to [BCG+20a].

3.10.3 Instantiating the template: challenges

We know from Section 3.5 that, assuming only a length-doubling PRG, there is an FSS scheme for every function which we can represent as a linear combination of point functions. From there, we can pick one of two paths:

- (1) Finding an appropriate pseudorandom function F_K such that $C \circ F_K$ fits within this class, or
- (2) If the above does not work out, or does not yield sufficiently satisfying instances, further expanding the class of functions for which we have efficient FSS until it captures $C \circ F_K$.

Looking ahead, I am going to describe two constructions, each of them following one of the two paths above. But before we delve into the details of these two constructions, it is interesting to take a step back and understand why our construction of PCG from syndrome decoding does not directly work here.

3.10.3.1 What goes wrong with syndrome decoding

In short, we constructed PCGs as $\mathsf{FSS}(C \circ G_k)$, where k is (a short description of) a sparse vector $\vec{e}$, and $G : k \to H \cdot \vec{e}$ with a public compressive matrix H. Then, G_k computes $H \cdot \vec{e}$ and outputs the i-th entry of this vector. The key hurdle of this construction can be described in one sentence:

Even to evaluate G_k at a single point i, one must compute the entire mapping $\vec{v} \to H \cdot \vec{v}$ before outputting its i-th entry.

Here, the reader might be raising an eyebrow: aren't we computing $H \cdot \vec{e}$ for a very sparse vector $\vec{e}$, say, with $t \approx 50$ nonzero entries? And if so, does this not require summing t columns of H? This is true, but recall that in the PCG construction, the parties will not manipulate $\vec{e}$ directly: rather, using FSS for point functions, they will first construct pseudorandom shares $(\vec{e}_0, \vec{e}_1)$ of $\vec{e}$ (which, unlike $\vec{e}$, are dense vectors!), and compute $(H \cdot \vec{e}_0, H \cdot \vec{e}_1)$.

Anyway, for our application to PCGs, this was no trouble: the $\mathsf{PCG.Expand}$ algorithm will, in any case, evaluate the PRG on its entire domain. But when we switch to PCFs, the *entire point* is to enable an incremental evaluation of the primitive, where each output $\mathsf{FSS}(C \circ F_K(i))$ can be computed independently of the others. With our syndrome-decoding-based PRG, the cost of evaluating $G_k(i)$ scales (at best linearly if there is a very fast mapping $\vec{v} \to H \cdot \vec{v}$) with the size of the *entire domain* of G_k. But a pseudorandom function should look random even given arbitrary polynomial access to its evaluations, hence the entire domain of a pseudorandom function F_K must have superpolynomial size![15]

[15] While a PRF can be defined over a polynomial-size domain, in this work, we typically focus on PRFs and PCFs which allow an arbitrary polynomial number of queries since our goal is to let the parties generate an arbitrary (polynomial) number of correlations.

3.10.4 Instantiating the template I: variable-density syndrome decoding

If we want to follow a similar approach to build PCFs, we will need a few crucial tweaks:

1. The matrix H and noise vector $\vec{e}$ should have superpolynomially large dimensions, yet $\vec{e}$ and the rows of H should admit a compact (polynomial-size) representation. This is easy for $\vec{e}$, which can be represented with $t \cdot \log n$ bits, which remains polynomial even for exponentially large n, but highly nontrivial for H: none of the candidates used for PCG construction satisfy this.

2. There should be an efficiently computable algorithm which, given an index i and (the compact description of) H and $\vec{e}_\sigma$ (party P_σ's FSS share of $\vec{e}$), outputs the i-th entry of $H \cdot \vec{e}_\sigma$.

We stress that we need both conditions above to hold, while simultaneously having the property that $H \cdot \vec{e}$ looks pseudorandom (to any polynomial-time algorithm that gets arbitrary access to the entries of this superpolynomially-large vector).

3.10.4.1 Key idea: a sparse but variable-density H

A natural way to satisfy the two conditions above would be to set H to have exponentially sparse rows, i.e., rows of exponential length $n(\lambda)$ with only $\mathrm{poly}(\lambda)$ nonzero entries: this way, its rows could be described succinctly (or even sampled on the fly). Furthermore, computing the i-th entries of $H \cdot \vec{e}_\sigma$ could be done by computing only the $\mathrm{poly}(\lambda)$ entries of $\vec{e}_\sigma$ corresponding to the nonzero entries $(i_1, \cdots, i_{\mathsf{HW}(h_i)})$ in the i-th row h_i of H, which translates to evaluating $\mathsf{FSS.Eval}(\sigma, \mathsf{k}_\sigma, i_j)$ for $j = 1$ to $\mathsf{HW}(h_i)$ and summing the $\mathrm{poly}(\lambda)$ outputs.

However, there is an apparent contradiction with the property that $H \cdot \vec{e}$ looks pseudorandom: if each h_i is exponentially sparse, and $\vec{e}$ is exponentially sparse as well, how could the inner products $\langle h_i, \vec{e} \rangle$ look pseudorandom? This is where the second idea comes in: we choose h_i and $\vec{e}$ with a *varying density* of nonzero entries. Concretely, by setting h_i and $\vec{e}$ to be dense at the beginning, and more sparse toward the end, we ensure that $\Pr_{h_i}[\langle h_i, \vec{e} \rangle = 0] \approx 1/2$. This does of course not suffice in itself to guarantee security. Nevertheless, we show that, by carefully balancing the varying densities in the vectors, it is possible to simultaneously reduce their Hamming weight to $\mathrm{poly}(\lambda)$ (enabling compact representation and evaluation of $\langle h_i, \vec{e}_\sigma \rangle$) while obtaining a variant of the syndrome decoding problem that is provably secure against all attacks from the linear test framework, which we covered in Section 3.7.1. At a high level, the variable density we choose is a regular distribution where each row h_i is divided into D blocks. The j-th block $h_{i,j}$ is sampled as a concatenation of λ unit vectors of length 2^j (observe that each block $h_{i,j}$ has exactly λ nonzero entries and a density of ones equal to $1/2^j$). The noise vector $\vec{e}$ is sampled from the same distribution as each of the rows h_i. The total number of nonzero entries per row is exactly $\lambda \cdot D$, while the length of a row is $O(\lambda \cdot 2^{2D})$. The formal definition follows.

3.10.4.2 Variable-density syndrome decoding

Fix parameters $\mathsf{pp} = (\lambda, D, N = 2^D)$. Let $\mathcal{R}_{\lambda,i}$ be the distribution of random λ-regular vectors over $\mathbb{F}_2^{\lambda \cdot 2^i}$: that is, a sample from $\mathcal{R}_{\lambda,i}$ is obtained by concatenating λ independent length-2^i unit vectors. We let $\mathcal{M}^i_{\mathsf{vd}}(\mathsf{pp})$ denote the distribution over $N \times (\lambda \cdot 2^i)$ matrices over $\mathbb{F}_2$ where each row is sampled independently from $\mathcal{R}_{\lambda,i}$, and $\mathcal{M}_{\mathsf{vd}}(\mathsf{pp})$ denote the distribution over $\mathbb{F}_2^{N\times 2N}$ obtained by sampling $H_i \xleftarrow{\$} \mathcal{M}^i_{\mathsf{vd}}(\mathsf{pp})$ for $i = 1$ to D and outputting $H = H_1||\cdots||H_D$. Eventually, we denote by $\mathcal{N}_{\mathsf{vd}}(\mathsf{pp})$ the noise distribution obtained by sampling $\vec{e}_i^{\intercal} \xleftarrow{\$} \mathcal{R}_{\lambda,i}$ and outputting $\vec{e} \leftarrow (\vec{e}_1//\cdots//\vec{e}_D) \in \mathbb{F}_2^{2N}$ (that is, $\vec{e}^{\intercal}$ is distributed as a row of H).

Definition 3.10.4 ($\mathsf{VDSD}(\lambda, D, N)$)**.** *The* variable-density syndrome decoding *assumption with VDSD λ, D blocks, and number of samples N, denoted* $\mathsf{VDLPN}(\lambda, D, N)$*, states that*

$$\{(H, \vec{b}) \mid H \xleftarrow{\$} \mathcal{M}_{\mathsf{vd}}(\mathsf{pp}), \vec{e} \xleftarrow{\$} \mathcal{N}_{\mathsf{vd}}(\mathsf{pp}), \vec{b} \leftarrow H \cdot \vec{e}\} \overset{c}{\approx} \{(H, \vec{b}) \mid H \xleftarrow{\$} \mathcal{M}_{\mathsf{vd}}(\mathsf{pp}), \vec{b} \xleftarrow{\$} \mathbb{F}_2^N\}.$$

In other words, using the terminology of Definition 3.7.2, VDSD corresponds to the $(\mathcal{M}_{\mathsf{vd}}(\mathsf{pp}, \mathcal{N}_{\mathsf{vd}}(\mathsf{pp}))\text{-}\mathsf{dGSD}(\mathbb{F}_2)$ assumption. In [BCG+20a; CD23], it was shown that the VDSD assumption is secure against all attacks from the linear test framework (Definition 3.7.3):

Theorem 3.10.5 ([BCG+20a; CD23], informal)**.** *The* $\mathsf{VDSD}(\lambda, D, 2^D)$ *assumption with* $D = \Omega(\lambda)$ *is* $(2^{-\Omega(\lambda)}, 2^{-\Omega(\lambda)})$*-secure against linear tests.*

The VDSD assumption parametrized with any $D = \mathrm{poly}(\lambda)$ immediately yields a weak PRF F:

- The vector $\vec{e} \xleftarrow{\$} \mathcal{N}_{\mathsf{vd}}(\mathsf{pp})$ defines the secret key of F.
- Given a string $x \in \{0,1\}^{\lambda\cdot\sum_{i\leq D} i}$, parse x into D blocks x_i of $\lambda \cdot i$ bits, each divided into λ strings $x_{i,j} \in \{0,1\}^i$. Map each $x_{i,j}$ to the length-2^i unit vector which has a 1 at $x_{i,j}$. Let $\mathsf{map}(x)$ denote the concatenation of all these unit vectors. Output $F_{\vec{e}}(x) = \mathsf{map}(x)^{\intercal} \cdot \vec{e}$.

For a random x, by construction, $\mathsf{map}(x)$ is distributed as a uniformly random column of H. Therefore, breaking the security of the above weak PRF after receiving N samples is equivalent to breaking the VDSD assumption.

3.10.4.3 From VDSD to PCFs

Using FSS for point functions, two parties can compute shares of the evaluation of the weak PRF $F_K : x \to \mathsf{map}(x)^{\intercal} \cdot \mathsf{map}(K)$ by summing $\lambda \cdot D = \mathrm{poly}(\lambda)$ evaluations of distributed point functions with respective domains $[\lambda \cdot 2^i]$. To obtain a PCF for some target additive correlation C from there, we have two options:

- For general degree-d additive correlations, we can use the observation 3.5.5 which lets us extend the class of function supported by our FSS constructions to handle any degree-d multivariate polynomial evaluated on each of the entries of $H \cdot \vec{e}$. Here, since $H \cdot \vec{e}$ is exponentially large, the transformation works as long as the multivariate polynomial is limited to taking a polynomial number of entries of $H \cdot \vec{e}$ as input). For example, I gave a step-by-step explanation of how to apply this observation to the Beaver triple correlation in Section 3.6.4. In this case, we get PCFs for Beaver triples with key size $O(\lambda' \cdot (\lambda \cdot D)^2 \cdot \log D)$ (where λ' denotes a security parameter for the distributed point function – we use a different one because it can typically be smaller than λ). While theoretically interesting, this yields much larger keys than for PCGs: using parameters computed in [CD23], we can set $\lambda' = 128$, $\lambda = 350$, and $D = 40$ (recall that the construction allows for up to 2^D pseudorandom instances of the correlation), this yields PCF keys of size 240 Gigabytes: that's not a very practical construction!

- Alternatively, if we only want PCFs for subfield VOLE or for OT, we can follow the much more efficient path outline in Section 3.8, which applies equally well to PCFs. This yields a PCF for OT under the VDSD assumption with key size $O(\lambda' \cdot \lambda \cdot D \cdot \log D)$, which translates to a few Megabytes using concrete parameters.

3.10.4.4 Historical notes

The variable density syndrome decoding assumption has been introduced in [BCG+20a] under the name *variable-density learning parity with noise* (VDLPN). In the context of PCGs, LPN and syndrome decoding are two different names for the same assumption, which reflect the emergence of related assumptions from within different communities (the learning theory community for LPN, and the coding theory community for syndrome decoding). Because I make heavy use of the coding-theoretic notions and formalism in this write-up, I decided to present all assumptions as being flavors of a generalized syndrome decoding assumption dGSD.

We initially proved the security of VDSD against attacks from the linear test framework in [BCG+20a] but, together with my Ph.D. student Clément Ducros, we identified a flaw in the analysis which we repaired in a subsequent paper [CD23]. This follow-up work also describes a variant of the VDSD assumption which enjoys the same features as the original assumption, but also benefits from a much tighter proof of security against linear tests, allowing to derive concrete provably secure parameters (the analysis in [BCG+20a] was purely asymptotic).

3.10.5 Instantiating the template II: expand-accumulate syndrome decoding

The construction from variable-density syndrome decoding induces a much larger overhead compared to PCG constructions, mainly because of the $\lambda \cdot D$ factor in the key length (and number of FSS evaluations per output) which is directly inherited from the use of a variable-density distribution (together with the constraints imposed by the security in the linear test framework). In this section, I will briefly cover a more recent and much more efficient construction that circumvents the limitations of the previous construction. The key insight underlying this improved construction lies in using more advanced FSS constructions for predicates such as the greater-than predicates, and its generalization to multidimensional rectangles. That is, unlike all other constructions described in this manuscript so far, *this new PCF cannot be based on distributed point function in a black-box way.* Below, I sketch both the more advanced FSS schemes used in the construction and the new variant of syndrome decoding tailored to these advanced FSS. In the spirit of not making this manuscript much longer than it already is, I will stick to relatively high-level sketches of the ideas here. The construction outlined in this section was described in [BCG+22].

3.10.5.1 Intuition

As before, we want to build a suitable "FSS-friendly" PRF using a syndrome decoding variant, where the noise $\vec{e}$ will be specified by the PRF key, and a random input x will define a random row from the parity-check matrix H. The goal is to define H and $\vec{e}$ such that (1) succinct shares of $\vec{e}$ can be efficiently distributed using and efficient function secret sharing scheme, and (2) obtaining shares of $\langle \vec{h}, \vec{e}_\sigma \rangle$ for a random row $\vec{h}$ of H and a (pseudo)random shares of $\vec{e}$ (computed via FSS) can be done efficiently (i.e., in polynomial time), even when the vectors $\vec{h}, \vec{e}$ have length exponential in λ.

The key idea underlying the construction is to replace the sparse noise $\vec{e}$ by an *interval noise*. A t-interval noise is filled with $t/2$ randomly placed all-1 strings. More precisely, $\vec{e}$ is sampled as follows: first, sample a random t-sparse $\vec{e}'$. Next, *accumulate* $\vec{e}'$ into $\vec{e}$: set $e_1 \leftarrow e'_1$, $e_2 \leftarrow e'_1 \oplus e'_2$, $e_3 \leftarrow e'_1 \oplus e'_2 \oplus e'_3$, and so on. The resulting distribution has the same entropy as the distribution of t-sparse vectors (since accumulating is a bijective mapping), but (crucially) it produces vectors whose proportion of zeroes and ones is balanced. Then, the expand-accumulate syndrome decoding assumption, which we introduce afterwards, essentially says that $(M, M \cdot \vec{e})$ is indistinguishable from $(M, \vec{b})$, where $\vec{b}$ is a random vector, M is a random *sparse* matrix, and $\vec{e}$ is a random t-interval noise. Because $\vec{e}$ is not sparse, this assumption can hold even when H is very sparse, and we actually provide heuristic support for its security using the linear test framework. The second core observation underlying the result from [BCG+22] is that t-interval noise can be written as a sum of t vectors of the form $(11 \cdots 1||00 \cdots 0)$, and such vectors can be succinctly shared using function secret sharing for *interval functions* $f_{<\alpha}$ which map every $x < \alpha$ to 1, and every $x \geq \alpha$ to 0.

The presentation below follows the outline of the intuition above, with the following modifications:

- The rate of the code is fixed to 1/5 (that is, H is an N-by-$5N$ matrix) for concreteness. This choice strikes a reasonable balance between efficiency and achieving good security bounds against linear tests.

- Observe that one can write a t-interval noise $\vec{e}$ as $\Delta_N \cdot \vec{e}'$, where $\vec{e}'$ is a t-sparse noise, and Δ_N is a $5N$-by-$5N$ lower triangular matrix filled with ones. In the definition below, we absorb the accumulation into the parity check matrix: that is, we view $H = M \cdot \Delta_N$ as the parity-check matrix, and the t-sparse vector $\vec{e}'$ as the noise (the two views are equivalent because Δ_N is a public invertible matrix). This presentation is more in line with describing expand-accumulate syndrome decoding as a variant of syndrome decoding over a standard noise distribution (but our description with t-interval noise matches more closely the intuition that motivated the approach).

- Eventually, we let $\vec{e}'$ be a *regular* t-sparse noise as in previous constructions, since it yields much more efficient constructions and provides the same security guarantees against linear tests (and does heuristically not seem to reduce security).

3.10.5.2 Expand-accumulate syndrome decoding

Fix parameters $\mathsf{pp} = (t, c, N)$. $N = N(\lambda)$ is the number of samples (typically exponential in λ, $c = c(\lambda)$ is a matrix sparsity parameter (typically $c = \Theta(\log N)$ or $\omega(\log N)$), and $t = t(\lambda)$ is the Hamming weight of the noise (typically $t = \Omega(\lambda)$). Let Δ_N denote a $5N$-by-$5N$ lower triangular matrix filled with ones. We let $\mathcal{M}_{\mathsf{ea}}(\mathsf{pp})$ denote the distribution obtained by sampling an N-by-$5N$ matrix M whose entries are independent Bernoulli sample equal to 1 with probability $c/2N$, and outputting $H = M \cdot \Delta_N$. Let $\mathsf{Ber}^{5N}_{c/2N}$ denote the distribution of the rows of M (that is, the rows are sampled as $5N$ independent Bernoulli samples). We denote by $\mathcal{N}_{\mathsf{ea}}(\mathsf{pp})$ the distribution obtained by concatenating t random unit vectors of length $5N/t$.

Definition 3.10.6 ($\mathsf{EASD}(t, c, N)$)**.** *The* expand-accumulate syndrome *assumption with noise weight t, matrix sparsity c, and number of samples N, denoted* $\mathsf{EASD}(t, c, N)$*, states that*

$$\{(H, \vec{b}) \mid H \xleftarrow{\$} \mathcal{M}_{\mathsf{ea}}(\mathsf{pp}), \vec{e} \xleftarrow{\$} \mathcal{N}_{\mathsf{ea}}(\mathsf{pp}), \vec{b} \leftarrow H \cdot \vec{e}\} \overset{c}{\approx} \{(H, \vec{b}) \mid H \xleftarrow{\$} \mathcal{M}_{\mathsf{ea}}(\mathsf{pp}), \vec{b} \xleftarrow{\$} \mathbb{F}_2^N\}.$$

The security of EASD against attacks from the linear test framework is directly stated as a theorem about the minimum distance of the code spanned by H in [BCG+22]:

Lemma 3.10.7 ([BCG+22], Theorem 3.10)**.** *Fix a parameter $c = \omega(\log N)$. The code generated by the rows of $H = M \cdot \Delta_N$ has minimum distance at least $\Omega(N)$, with probability at least $1 - N^{-\omega(1)}$ over the choice of H.*

We now explain how to get a weak PRF from the EASD assumption. Let $\vec{e}_1, \cdots, \vec{e}_t$ denote the t unit blocks of $\vec{e}$, and let $i_1, \cdots, i_t$ denote the indices of their nonzero entries. Observe that $\vec{v} \leftarrow \Delta_N \cdot \vec{e}$ can be decomposed into t blocks $\vec{v}_1, \cdots, \vec{v}_t$ such that each block $\vec{v}_j$ for j odd is of the form

$$(0, \cdots, 0, 1, \cdots, 1),$$

where the index of the first 1 is i_j. When j is even, the block is similar, but in reverse order: $(1, \cdots, 1, 0, \cdots, 0)$, with a first 0 at position i_j. This is more easily observed by noting that the operator $\vec{u} \to \Delta_N \cdot \vec{u}$ is the *accumulator* operator, which outputs a vector $\vec{v}$ whose j-th entries is the XOR of all entries of $\vec{u}$ from 1 to j. Now, each of these blocks can be seen as the truth table of a greater-than predicate, either $f^{\geq}_{i_j}(\ell)$ which outputs 1 iff $\ell \geq i_j$ (for odd j), or $f^{\leq}_{i_j}(\ell)$ which outputs 1 iff $\ell \leq i_j$ (for even j). This observation is what allows to compute the i-th entry of $M \cdot \Delta_N \cdot \vec{e}$ without computing the full vector, by evaluating $\approx c$ greater-than predicates on inputs defined by the i-th row of H (whose Hamming weight is close to c with high probability).

- As for VDSD, the vector $\vec{e} \xleftarrow{\$} \mathcal{N}_{\mathsf{ea}}(\mathsf{pp})$ defines the secret key of F. Let K denotes its compressed description ($K \in \{0,1\}^{t \cdot \log(5N/t)}$).
- On input a random x,
  - use x as the randomness to sample a length-$5N$ vector $h \leftarrow \mathsf{Ber}^{5N}_{c/2N}(x)$ from the distribution $\mathsf{Ber}^{5N}_{c/2N}$.
  - Let c' be the Hamming weight of h and let $(\ell_1, \cdots, \ell_{c'})$ denote the positions of the nonzero entries in h. Divide h into t equal-length blocks. For $j = 1$ to c', retrieve the index j' of the block where ℓ_j falls, and compute y_j as $f^{\geq}_{i_{j'}}(\ell_j)$ (if j' is odd) or as $f^{\leq}_{i_{j'}}(\ell_j)$ (if j' is even).
  - Output $F_K(x) = \bigoplus_{j=1}^{t} y_j$.

The function is indeed efficient, as it requires sampling a vector of average density c and XORing the outputs of t greater-than predicates. I will let the reader check that the value $F_K(x)$ corresponds exactly to $h \cdot \Delta_N \cdot \vec{e}$, where $h = \mathsf{Ber}^{5N}_{c/2N}(x)$ is sampled as a uniformly random row of H. Therefore, distinguishing F_K from a random function using at most N call entails distinguishing $H \cdot \Delta_N \cdot \vec{e}$ from random, which contradicts the EASD assumption.

3.10.5.3 Function secret sharing for intervals

As we just saw, EASD also yields a weak PRF. But where the PRF obtained from VDSD could be written as an XOR of point functions, that is not the case anymore: rather, we have a weak PRF which can be written as an XOR of *greater-than predicates*. Fortunately, it turns out that there also exist efficient constructions FSS for greater-than predicates (called *distributed comparison functions*, or DCFs [BGI16b; BCG+21a]), assuming only a length-doubling pseudorandom generator. The construction, while more involved, follows an approach similar to the construction of distributed point functions. Informally, a DCF allows one to efficiently share a comparison function $f^{\beta}_{<\alpha} : [N] \to \mathbb{F}$ which maps every $x < \alpha$ to β and every $x \geq \alpha$ to 0.

3.10.5.4 From EASD to PCFs

Using FSS for the greater-than predicate, two parties can compute shares of the evaluation of the weak PRF F_K defined above. To obtain a PCF for some target additive correlation C from there, we have again two options:

General construction for degree-d correlations. For general degree-d additive correlations, use the observation 3.5.5. But there is a catch: while the product of point functions is still a point function (with quadratically larger domain), the product of two greater-than predicates is not a greater-than predicate anymore, but rather a predicate defined by a "two-dimensional rectangle". Fortunately, the constructions described in the work of [BGI16b] extend to predicates defined by multi-dimensional rectangles, as a particular case of the general construction of PRG-based FSS for predicates defined by an arbitrary decision tree. Concretely, if we target the Beaver triple correlation (a degree-2 correlation), we must rely on the function class of two-dimensional $5N/t \times 5N/t$ intervals over a ring $\mathcal{R}$, where each function in the class is specified by a pair of indices $\alpha_1, \alpha_2 \in [5N/t]$ and $\beta \in \mathcal{R}$, and defined by

$$\begin{aligned} f_{\alpha_1,\alpha_2,\beta} &: [5N/t] \times [5N/t] \to \mathcal{R} \\ (x_1, x_2) &\mapsto \begin{cases} \beta & \text{if } x_1 < \alpha_1 \text{ and } x_2 < \alpha_2 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

The FSS construction from [BGI16b] has a key size bounded by $2|V|(\lambda + 1)$ bits, where V is the set of nodes in the binary decision tree. A $5N/t \times 5N/t$ interval function can be expressed as a decision tree with $2\log(5N/t)(\log(5N/t) + 1) + 1$ nodes, giving a key size of roughly $4\log^2 5N/t\lambda$ bits for the FSS scheme. Plugging this FSS in our construction for Beaver triples yields a PCF with a key size of roughly $4 \cdot (t \cdot \log(5N/t))^2\lambda$ bits. Using parameters estimations from [BCG+22], this translates to keys of size about 200 Megabytes.

Specialized construction for OT correlations. Alternatively, if we only want PCFs for subfield VOLE or for OT, we can again follow the much more efficient path outline in Section 3.8. In fact, for the same reason that we could replace the DPF with a PPRF when tailoring the template to the OT correlation, it turns out that a relaxed notion of distributed comparison function suffices, and that this notion can be realized more efficiently. Let me skip the details at this stage and refer the reader to [BCG+22] to discover in more detail what relaxed DCFs are, and how they can be constructed. I will just note that using this approach, the EASD assumption yields a very efficient PCF, with keys in the 200-Kilobyte range, and the ability to generate around 10^5 pseudorandom OTs per second on one core of a standard laptop (e.g., any modern laptop with a processor that has the AES-NI instruction set).

3.11 Beyond Pseudorandom Correlation Functions

We are now reaching the end of this manuscript, which I hope conveys some useful intuitions about pseudorandom correlation generators and their variants: how to build them, how efficient they are, and what challenges lie on the road to better PCGs and PCFs. Before I leave you, let me briefly mention a few more thrilling directions and topics, some of which I briefly mentioned in passing in this write-up.

3.11.1 Distributed setup

Even though PCGs and PCFs are the cornerstone of silent secure computation, they do not suffice alone to yield silent protocols: to use them, one also needs to interactively and securely execute the Gen algorithm of the PCG/PCF. While in theory, the fact that Gen outputs short keys suffices to guarantee the existence of a low-communication protocol to securely distribute them, it might not translate to a concretely efficient protocol *in practice*.

Designing efficient protocols for distributing PCG keys has been an important research topic with nontrivial challenges on its own, especially in the setting of security against malicious adversaries. What makes this challenge particularly well motivated is that (1) we have very efficient MPC protocols in the correlated randomness model with security against malicious adversaries, and (2) a nice byproduct of the silent generation of pseudorandom correlations is that, when using a maliciously-secure MPC protocol in the correlated randomness model, it suffices to use a maliciously-secure protocol to distribute the PCG keys for the resulting silent protocol to be maliciously secure! In theory, this permits to achieve malicious security at a minimal overhead – and an important research effort has been devoted to turning this nice theoretical feature into a reality. In particular, the use of PPRFs (or relaxed DCFs) has been key to achieving more efficient malicious distributed setup protocols.

3.11.2 Multiparty PCGs

I touched on this very briefly in Section 3.9.5. In short, all constructions I have described throughout this entire manuscript are restricted to the two-party setting, and as such, they only enable two-party silent secure computation. The more general setting of n-party silent MPC requires a generalization of PCGs to n-party PCGs. And the reason I have not introduced the notion is... We do not know how to construct it yet!

This is not entirely true. It is known that n-party PCGs follow from some high-end cryptographic primitives, such as indistinguishability obfuscation [BGI15] or spooky encryption [DHRW16] (the latter being known from the LWE assumption). However, none of these constructions are anywhere near practical, leaving a wide gap between the two-party setting, where we can base concretely efficient on coding-theoretic assumptions, and the n-party setting for $n > 2$ where only feasibility results using "heavy hammer" cryptography are known. The issue lies not in the template itself, which extends immediately to the n party setting, but in the FSS component: in fact, the two works I mentioned above [BGI15; DHRW16] construct multiparty FSS, and the application to n-party PCGs follows by plugging these FSS into the template.

3.11.2.1 The challenge of multiparty DPFs

Now, to instantiate the template from coding-theoretic assumptions, when targetting low-degree additive correlations, it would suffice to have n-party FSS for point functions only. Unfortunately, while two-party FSS for point functions with domain size m is known with key size $O(\lambda \cdot \log m)$, the best known 3-party constructions have key size $O(\lambda \cdot \sqrt{m})$. This construction is achieved using a slight tweak on the construction which I described in Section 3.5.2.1:

- To deal "compressed shares of zero" to three parties, the dealer distributes unordered pairs of PRG seeds $\{a,b\}, \{a,c\}$, and $\{b,c\}$ to the three parties, who expend them by XORing the outputs of the PRG on each of them. This works, because $(\mathsf{PRG}(a) \oplus \mathsf{PRG}(b)) \oplus (\mathsf{PRG}(a) \oplus \mathsf{PRG}(c)) \oplus (\mathsf{PRG}(b) \oplus \mathsf{PRG}(c)) = 0$.

- For the α_0-th row, the dealer distributes instead unordered pairs of the form $\{a,b\}, \{a,c\}$, and $\{a,d\}$. Observe now that $(\mathsf{PRG}(a) \oplus \mathsf{PRG}(b)) \oplus (\mathsf{PRG}(a) \oplus \mathsf{PRG}(c)) \oplus (\mathsf{PRG}(a) \oplus \mathsf{PRG}(d))$ is pseudorandom from the viewpoint of any pair of parties (for example, it looks pseudorandom to the first two parties because of the $\mathsf{PRG}(d)$ term). Yet, no pair of parties can distinguish their seeds from three-party compressed shares of zeroes: they always just see a pair of unordered sets with a single common element.

Unfortunately, unlike the two-party construction, there is no known variant of this technique that opens itself to recursive composition, hence the method is stuck at $O(\sqrt{m})$-sized keys (ignoring terms depending on λ). This is still a nontrivial key size, but when using the template for degree-d additive correlation, there is a power-of-d blowup in the seed size, which becomes already linear in m for $d = 2$!

When targetting only three-party Beaver triples, this limitation can be circumvented by using the more efficient PCG for OLEs which we covered in Section 3.9, since it avoids this quadratic overhead. This approach yields three-party PCG for Beaver triples from the ring syndrome decoding assumption with nontrivial key size $O(t^2 \cdot \lambda \cdot \sqrt{m})$, which have found some interesting applications in recent works [AS22] but are still considerably less efficient than their two-party counterpart.

3.11.2.2 Nesting to the rescue

To circumvent the limitations of the PRG-based construction of FSS for point functions, one possible direction is to rely on a nesting technique. At a high level, the idea of nesting is to combine an external FSS scheme within an internal FSS scheme. Concretely, suppose you have a two-party FSS scheme $\mathsf{FSS}_{\mathsf{in}} = (\mathsf{FSS}_{\mathsf{in}}.\mathsf{Gen}, \mathsf{FSS}_{\mathsf{in}}.\mathsf{Eval})$ for some class of functions $\mathcal{F}_{\mathsf{in}}$, and an outer two-party FSS scheme $\mathsf{FSS}_{\mathsf{out}} = (\mathsf{FSS}_{\mathsf{out}}.\mathsf{Gen}, \mathsf{FSS}_{\mathsf{out}}.\mathsf{Eval})$ for the class of functions $\mathcal{F}_{\mathsf{out}} = \{x \to \mathsf{FSS}_{\mathsf{in}}.\mathsf{Eval}(\mathsf{k}, f(x)) \mid f \in \mathcal{F}_{\mathsf{in}}\}$ (that is, the outer scheme supports a class of function that evaluates $\mathsf{FSS}_{\mathsf{in}}.\mathsf{Eval}$ on top of functions from $\mathcal{F}_{\mathsf{in}}$). Then one directly gets a 4-party FSS scheme for the class $\mathsf{FSS}_{\mathsf{in}}$:

- Generate 4 keys by running $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{FSS}_{\mathsf{in}}.\mathsf{Gen}(f)$, and then

$$(\mathsf{k}_{\sigma,0}, \mathsf{k}_{\sigma,1}) \leftarrow \mathsf{FSS}_{\mathsf{out}}.\mathsf{Gen}(\mathsf{k}_\sigma, f)$$

for $\sigma = 0, 1$.

- To evaluate shares of $f(x)$, each pair of parties $(P_{\sigma,0}, P_{\sigma,1})$ runs $\mathsf{FSS}_{\mathsf{out}}.\mathsf{Eval}(\mathsf{k}_{\sigma,b}, x)$ for $\sigma, b = 0, 1$ to obtain additive shares of $\mathsf{FSS}_{\mathsf{in}}.\mathsf{Eval}((\mathsf{k}_\sigma, f), x)$. The terms $\mathsf{FSS}_{\mathsf{in}}.\mathsf{Eval}((\mathsf{k}_\sigma, f), x)$ for $\sigma = 0, 1$ form themselves additive shares of $f(x)$, as expected.

In a recent work [BCM23], we used this nesting technique to obtain new results on low-communication secure computation with more than two parties. Because at each application of the nesting, the complexity of the function class blows up, it seems to be limited to a very small number of nesting operations (typically one). Compared to constructions based on iO or LWE, it is therefore limited to fewer parties but can be instantiated from a wider variety of assumptions. And as the reader can guess, it is also not very efficient, we are still in the realm of theoretical feasibility results.

3.11.2.3 Programmability to the rescue

A much more practical solution is offered by the notion of *programmable* PCGs. At a high level, a programmable PCG has the feature that multiple instances of the 2-party PCG can be combined into a single instance of an n-party PCG. This is easier to see if the reader remembers the construction of n-party Beaver triples from 2-party Beaver triples from Section 2.7.2.1: one can view an n-party Beaver triple as a linear combination of $O(n^2)$ “correlated” two-party Beaver triples. Programmable PCGs for Beaver triples allow precisely to correlate two instances in the following way: given a pair of keys $(\mathsf{k}_0, \mathsf{k}_1)$ generating shares of pseudorandom vectors $(\vec{a}, \vec{b})$ and of their component-wise product $\vec{a} \odot \vec{b}$, one can generate a related pair of keys $(\mathsf{k}_0, \mathsf{k}_1)$ generating shares of pseudorandom vectors $(\vec{a}, \vec{c})$ and their component-wise product $\vec{a} \odot \vec{c}$, with a new pseudorandom vector $\vec{c}$, but the same $\vec{a}$ as before.

The notion of programmable PCGs has been introduced in [BCG+19b], and a programmable PCG for Beaver triples was described in [BCG+20b]. It offers a practical way to achieve n-party PCGs, with two main caveats:

- It is not compatible with the technique that lets one generate PCG keys for *authenticated* Beaver triples, which we mentioned in Section 3.9.4. Concretely, this means that we do not have an efficient strategy, as of today, to generate (silently) the type of n-party correlated randomness used in malicious secure computation.

- The cost of sharing an n-party correlation scales as n^2, while a “direct” FSS-based construction (obtained by plugging an n-party FSS in the template) would have keys of size scaling linearly with n. This makes the construction efficient only for a relatively small number of parties.

3.11.3 Public-key PCFs

Eventually, the notion of PCF which we covered in Section 3.10 is still not the perfect counterpart to the efficient protocols we have for the easier setting of secure communication. The reader might remember that we described PCF as a means to emulate a property similar to what one gets from stream ciphers, but in the secure computation setting: a way to continuously and on-demand generate the appropriate shared material to be used in a two-party secure computation protocol.

Now, PCFs like stream ciphers require a distributed setup phase, where the two parties will interact to generate short keys. Over a large network, say, with n users, this means that if n parties want to be able to do pairwise secure computations at any point in the future, they will need to run $\Omega(n^2)$ pairwise distributed setup protocols among themselves, to let each pair of parties obtain shared PCF keys. For secure communication, however, this can be circumvented using public key cryptography: in modern secure communication protocols, each participant simply uploads a short public key online and stores a short associated secret key. Now, whenever two parties P_i, P_j want to communicate, they can directly derive a shared key K from $(\mathsf{pk}_j, \mathsf{sk}_i)$ and $(\mathsf{pk}_i, \mathsf{sk}_j)$ respectively. This means that only $O(n)$ communications are required to enable secure communication between every pair of nodes in the network – a fundamental property over a very large network such as the Web.

A strengthening of the notion of PCF, dubbed *public-key PCF*, was described in [OSY21] to emulate precisely these features. Informally, a public-key PCF enables all parties to publish online a short public key and store a short secret key, such that each pair (P_i, P_j) of parties can later run a key derivation algorithm (without further communication) $\mathsf{k}_i \leftarrow \mathsf{KeyDer}(\mathsf{pk}_j, \mathsf{sk}_i, i)$ and $\mathsf{k}_j \leftarrow \mathsf{KeyDer}(\mathsf{pk}_i, \mathsf{sk}_j, j)$ so that the key pair $(\mathsf{k}_i, \mathsf{k}_j)$ is a PCF key pair (from which the parties can later derive an arbitrary amount of correlated pseudorandomness). None of the constructions described in this manuscript achieves the stronger notion of public-key PCF. The work of [OSY21] described the first "reasonably efficient" construction using clever and elegant ideas, but their construction is still about five orders of magnitude slower than, for example, the PCF based on Expand-Accumulate codes. Achieving concretely efficient public-key PCFs, which I view as being the holy grail of silent secure computation, remains a thrilling open problem, which is the subject of ongoing work with some of my students.

Bibliography

[ABB+17] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. "Bike: Bit flipping key encapsulation". In: (2017) (cit. on p. 91).

[ABB+20] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, and Santosh Ghosh. *BIKE*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions. National Institute of Standards and Technology, 2020 (cit. on p. 80).

[ABG+14] Adi Akavia, Andrej Bogdanov, Siyao Guo, Akshay Kamath, and Alon Rosen. "Candidate weak pseudorandom functions in AC^0 o MOD_2". In: *ITCS 2014*. Ed. by Moni Naor. ACM, Jan. 2014, pp. 251–260. DOI: 10.1145/2554797.2554821 (cit. on p. 77).

[ACH20] Thomas Agrikola, Geoffroy Couteau, and Dennis Hofheinz. "The Usefulness of Sparsifiable Inputs: How to Avoid Subexponential iO". In: *PKC 2020, Part I*. Ed. by Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas. Vol. 12110. LNCS. Springer, Heidelberg, May 2020, pp. 187–219. DOI: 10.1007/978-3-030-45374-9_7 (cit. on p. 9).

[ACI+20] Thomas Agrikola, Geoffroy Couteau, Yuval Ishai, Stanislaw Jarecki, and Amit Sahai. "On Pseudorandom Encodings". In: *TCC 2020, Part III*. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12552. LNCS. Springer, Heidelberg, Nov. 2020, pp. 639–669. DOI: 10.1007/978-3-030-64381-2_23 (cit. on p. 9).

[ACM22] Thomas Agrikola, Geoffroy Couteau, and Sven Maier. "Anonymous Whistleblowing over Authenticated Channels". In: *TCC 2022, Part II*. Ed. by Eike Kiltz and Vinod Vaikuntanathan. Vol. 13748. LNCS. Springer, Heidelberg, Nov. 2022, pp. 685–714. DOI: 10.1007/978-3-031-22365-5_24 (cit. on p. 8).

[ACMS23] Abtin Afshar, Geoffroy Couteau, Mohammad Mahmoody, and Elahe Sadeghi. "Fine-Grained Non-interactive Key-Exchange: Constructions and Lower Bounds". In: *EUROCRYPT 2023, Part I*. LNCS. Springer, Heidelberg, June 2023, pp. 55–85. DOI: 10.1007/978-3-031-30545-0_3 (cit. on p. 8).

[ADI+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. "Secure Arithmetic Computation with Constant Computational Overhead". In: *CRYPTO 2017, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 223–254. DOI: 10.1007/978-3-319-63688-7_8 (cit. on pp. 45, 76, 88).

[AFS03] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. *A Fast Provably Secure Cryptographic Hash Function.* Cryptology ePrint Archive, Report 2003/230. https://eprint.iacr.org/2003/230. 2003 (cit. on p. 84).

[AG11] Sanjeev Arora and Rong Ge. “New Algorithms for Learning in Presence of Errors”. In: *ICALP 2011.* Earlier version: Learning Parities with Structured Noise, ECCC 2010. 2011, pp. 403–415 (cit. on p. 79).

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. “On pseudorandom generators with linear stretch in NC0”. In: *International Workshop on Approximation Algorithms for Combinatorial Optimization.* Springer. 2006, pp. 260–271 (cit. on p. 72).

[AIK09] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. “Cryptography with Constant Input Locality”. In: *Journal of Cryptology* 22.4 (Oct. 2009), pp. 429–469. DOI: 10.1007/s00145-009-9039-0 (cit. on p. 73).

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. “Priced Oblivious Transfer: How to Sell Digital Goods”. In: *EUROCRYPT 2001.* Ed. by Birgit Pfitzmann. Vol. 2045. LNCS. Springer, Heidelberg, May 2001, pp. 119–135. DOI: 10.1007/3-540-44987-6_8 (cit. on p. 29).

[AK23] Benny Applebaum and Niv Konstantini. “Actively Secure Arithmetic Computation and VOLE with Constant Computational Overhead”. In: *EUROCRYPT 2023, Part II.* LNCS. Springer, Heidelberg, June 2023, pp. 190–219. DOI: 10.1007/978-3-031-30617-4_7 (cit. on p. 45).

[Al 01] Abdulrahman Al Jabri. “A statistical decoding algorithm for general linear block codes”. In: 2001 (cit. on pp. 72, 77).

[AL17] Gilad Asharov and Yehuda Lindell. “A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation”. In: *Journal of Cryptology* 30.1 (Jan. 2017), pp. 58–151. DOI: 10.1007/s00145-015-9214-4 (cit. on pp. 23, 24).

[Ale03] Michael Alekhnovich. “More on Average Case vs Approximation Complexity”. In: *44th FOCS.* IEEE Computer Society Press, Oct. 2003, pp. 298–307. DOI: 10.1109/SFCS.2003.1238204 (cit. on pp. 29, 72, 79).

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. “More efficient oblivious transfer and extensions for faster secure computation”. In: *ACM CCS 2013.* Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 535–548. DOI: 10.1145/2508859.2516738 (cit. on p. 37).

[ALSZ15] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. “More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries”. In: *EUROCRYPT 2015, Part I.* Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 673–701. DOI: 10.1007/978-3-662-46800-5_26 (cit. on p. 37).

[AS22] Damiano Abram and Peter Scholl. “Low-Communication Multiparty Triple Generation for SPDZ from Ring-LPN”. In: *PKC 2022, Part I.* Ed. by Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe. Vol. 13177. LNCS. Springer, Heidelberg, Mar. 2022, pp. 221–251. DOI: 10.1007/978-3-030-97121-2_9 (cit. on p. 104).

[BBCS92] Charles H. Bennett, Gilles Brassard, Claude Crépeau, and Marie-Hélène Skubiszewska. “Practical Quantum Oblivious Transfer”. In: *CRYPTO’91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 351–366. DOI: 10.1007/3-540-46766-1_29 (cit. on p. 29).

[BC15] Olivier Blazy and Céline Chevalier. “Generic Construction of UC-Secure Oblivious Transfer”. In: *ACNS 15*. Ed. by Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis. Vol. 9092. LNCS. Springer, Heidelberg, June 2015, pp. 65–86. DOI: 10.1007/978-3-319-28166-7_4 (cit. on p. 29).

[BC22] Chris Brzuska and Geoffroy Couteau. “On Building Fine-Grained One-Way Functions from Strong Average-Case Hardness”. In: *EUROCRYPT 2022, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. LNCS. Springer, Heidelberg, May 2022, pp. 584–613. DOI: 10.1007/978-3-031-07085-3_20 (cit. on p. 8).

[BC23] Dung Bui and Geoffroy Couteau. “Improved Private Set Intersection for Sets with Small Entries”. In: *PKC 2023, Part II*. LNCS. Springer, Heidelberg, May 2023, pp. 190–220. DOI: 10.1007/978-3-031-31371-4_7 (cit. on p. 5).

[BCCD23] Maxime Bombar, Geoffroy Couteau, Alain Couvreur, and Clément Ducros. “Correlated Pseudorandomness from the Hardness of Quasi-Abelian Decoding”. In: *Annual International Cryptology Conference*. Springer. 2023 (cit. on pp. 53, 61, 77, 80, 91).

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. “Homomorphic Secret Sharing: Optimizations and Applications”. In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 2105–2122. DOI: 10.1145/3133956.3134107 (cit. on pp. 53, 61).

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. “Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation”. In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 291–308. DOI: 10.1145/3319535.3354255 (cit. on pp. 53, 61, 80, 83, 87).

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. “Efficient Pseudorandom Correlation Generators: Silent OT Extension and More”. In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 489–518. DOI: 10.1007/978-3-030-26954-8_16 (cit. on pp. 6, 7, 53, 58–61, 80, 105).

[BCG+20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. “Correlated Pseudorandom Functions from Variable-Density LPN”. In: *61st FOCS*. IEEE Computer Society Press, Nov. 2020, pp. 1069–1080. DOI: 10.1109/FOCS46700.2020.00103 (cit. on pp. 6, 7, 9, 53, 61, 77, 93, 96, 98, 99).

[BCG+20b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. “Efficient Pseudorandom Correlation Generators from Ring-LPN”. In: *CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. LNCS. Springer, Heidelberg, Aug. 2020, pp. 387–416. DOI: 10.1007/978-3-030-56880-1_14 (cit. on pp. 53, 61, 80, 90, 91, 105).

[BCG+21a] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. “Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation”. In: *EUROCRYPT 2021, Part II*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. LNCS. Springer, Heidelberg, Oct. 2021, pp. 871–900. DOI: 10.1007/978-3-030-77886-6_30 (cit. on p. 102).

[BCG+21b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. “Low-Complexity Weak Pseudorandom Functions in $\mathtt{AC0}[\mathtt{MOD2}]$”. In: *CRYPTO 2021, Part IV*. Ed. by Tal Malkin and Chris Peikert. Vol. 12828. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 487–516. DOI: 10.1007/978-3-030-84259-8_17 (cit. on pp. 9, 53).

[BCG+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. “Correlated Pseudorandomness from Expand-Accumulate Codes”. In: *CRYPTO 2022, Part II*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13508. LNCS. Springer, Heidelberg, Aug. 2022, pp. 603–633. DOI: 10.1007/978-3-031-15979-4_21 (cit. on pp. 6, 7, 53, 61, 77, 80, 88, 99–102).

[BCG+23] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. “Oblivious Transfer with Constant Computational Overhead”. In: *EUROCRYPT 2023, Part I*. LNCS. Springer, Heidelberg, June 2023, pp. 271–302. DOI: 10.1007/978-3-031-30545-0_10.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. “Compressing Vector OLE”. In: *ACM CCS 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM Press, Oct. 2018, pp. 896–912. DOI: 10.1145/3243734.3243868 (cit. on pp. 6, 7, 53, 59–61, 79–81, 83).

[BCKR21] Chris Brzuska, Geoffroy Couteau, Pihla Karanko, and Felix Rohrbach. “On Derandomizing Yao’s Weak-to-Strong OWF Construction”. In: *TCC 2021, Part II*. Ed. by Kobbi Nissim and Brent Waters. Vol. 13043. LNCS. Springer, Heidelberg, Nov. 2021, pp. 429–456. DOI: 10.1007/978-3-030-90453-1_15 (cit. on p. 9).

[BCM22] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. “Sublinear Secure Computation from New Assumptions”. In: *TCC 2022, Part II*. Ed. by Eike Kiltz and Vinod Vaikuntanathan. Vol. 13748. LNCS. Springer, Heidelberg, Nov. 2022, pp. 121–150. DOI: 10.1007/978-3-031-22365-5_5 (cit. on p. 4).

[BCM23] Elette Boyle, Geoffroy Couteau, and Pierre Meyer. “Sublinear-Communication Secure Multiparty Computation Does Not Require FHE”. In: *EUROCRYPT 2023, Part II*. LNCS. Springer, Heidelberg, June 2023, pp. 159–189. DOI: 10.1007/978-3-031-30617-4_6 (cit. on pp. 4, 105).

[BCPW15] Fabrice Benhamouda, Geoffroy Couteau, David Pointcheval, and Hoeteck Wee. "Implicit Zero-Knowledge Arguments and Applications to the Malicious Setting". In: *CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. LNCS. Springer, Heidelberg, Aug. 2015, pp. 107–129. DOI: 10.1007/978-3-662-48000-7_6 (cit. on pp. 6, 7).

[BDO23] Lennart Braun, Ivan Damgård, and Claudio Orlandi. "Secure multiparty computation from threshold encryption based on class groups". In: *Annual International Cryptology Conference*. Springer. 2023, pp. 613–645 (cit. on p. 49).

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. "Semi-homomorphic Encryption and Multiparty Computation". In: *EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. LNCS. Springer, Heidelberg, May 2011, pp. 169–188. DOI: 10.1007/978-3-642-20465-4_11 (cit. on p. 49).

[Bea91] Donald Beaver. "Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority". In: *Journal of Cryptology* 4.2 (Jan. 1991), pp. 75–122. DOI: 10.1007/BF00196771 (cit. on p. 24).

[Bea92a] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 420–432. DOI: 10.1007/3-540-46766-1_34 (cit. on p. 41).

[Bea92b] Donald Beaver. "Foundations of Secure Interactive Computing". In: *CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 377–391. DOI: 10.1007/3-540-46766-1_31 (cit. on p. 24).

[Bea95] Donald Beaver. "Precomputing Oblivious Transfer". In: *CRYPTO'95*. Ed. by Don Coppersmith. Vol. 963. LNCS. Springer, Heidelberg, Aug. 1995, pp. 97–109. DOI: 10.1007/3-540-44750-4_8 (cit. on pp. 38, 41).

[Bea96] Donald Beaver. "Correlated Pseudorandomness and the Complexity of Private Computations". In: *28th ACM STOC*. ACM Press, May 1996, pp. 479–488. DOI: 10.1145/237814.237996 (cit. on pp. 36, 61).

[Bea98] Donald Beaver. "Adaptively Secure Oblivious Transfer". In: *ASIACRYPT'98*. Ed. by Kazuo Ohta and Dingyi Pei. Vol. 1514. LNCS. Springer, Heidelberg, Oct. 1998, pp. 300–314. DOI: 10.1007/3-540-49649-1_24 (cit. on p. 29).

[Ber06] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *PKC 2006*. Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Vol. 3958. LNCS. Springer, Heidelberg, Apr. 2006, pp. 207–228. DOI: 10.1007/11745853_14 (cit. on p. 30).

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. "Cryptographic Primitives Based on Hard Learning Problems". In: *CRYPTO'93*. Ed. by Douglas R. Stinson. Vol. 773. LNCS. Springer, Heidelberg, Aug. 1994, pp. 278–291. DOI: 10.1007/3-540-48329-2_24 (cit. on pp. 72, 73).

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. "Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)". In: *20th ACM STOC*. ACM Press, May 1988, pp. 103–112. DOI: 10.1145/62212.62222 (cit. on p. 5).

[BGI08] Eli Biham, Yaron J. Goren, and Yuval Ishai. “Basing Weak Public-Key Cryptography on Strong One-Way Functions”. In: *TCC 2008*. Ed. by Ran Canetti. Vol. 4948. LNCS. Springer, Heidelberg, Mar. 2008, pp. 55–72. DOI: 10.1007/978-3-540-78524-8_4 (cit. on p. 8).

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. “Functional Signatures and Pseudorandom Functions”. In: *PKC 2014*. Ed. by Hugo Krawczyk. Vol. 8383. LNCS. Springer, Heidelberg, Mar. 2014, pp. 501–519. DOI: 10.1007/978-3-642-54631-0_29 (cit. on pp. 85, 86).

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. “Function Secret Sharing”. In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 337–367. DOI: 10.1007/978-3-662-46803-6_12 (cit. on pp. 64, 65, 68, 103).

[BGI16a] Elette Boyle, Niv Gilboa, and Yuval Ishai. “Breaking the Circuit Size Barrier for Secure Computation Under DDH”. In: *CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 509–539. DOI: 10.1007/978-3-662-53018-4_19 (cit. on p. 4).

[BGI16b] Elette Boyle, Niv Gilboa, and Yuval Ishai. “Function Secret Sharing: Improvements and Extensions”. In: *ACM CCS 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, Oct. 2016, pp. 1292–1303. DOI: 10.1145/2976749.2978429 (cit. on pp. 64, 68, 102).

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. “Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)”. In: *20th ACM STOC*. ACM Press, May 1988, pp. 1–10. DOI: 10.1145/62212.62213 (cit. on p. 20).

[BHK+11] Gilles Brassard, Peter Høyer, Kassem Kalach, Marc Kaplan, Sophie Laplante, and Louis Salvail. “Merkle Puzzles in a Quantum World”. In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 391–410. DOI: 10.1007/978-3-642-22792-9_22 (cit. on p. 8).

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. “Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding”. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 520–536. DOI: 10.1007/978-3-642-29011-4_31 (cit. on pp. 72, 77).

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. “Noise-tolerant learning, the parity problem, and the statistical query model”. In: *32nd ACM STOC*. ACM Press, May 2000, pp. 435–440. DOI: 10.1145/335305.335355 (cit. on pp. 72, 77).

[BL12] Daniel J Bernstein and Tanja Lange. “Never trust a bunny”. In: *International Workshop on Radio Frequency Identification: Security and Privacy Issues*. Springer. 2012, pp. 137–148 (cit. on p. 91).

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. "Smaller Decoding Exponents: Ball-Collision Decoding". In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 743–760. DOI: 10.1007/978-3-642-22792-9_42 (cit. on pp. 72, 77).

[BLPS11] Daniel J. Bernstein, Tanja Lange, Christiane Peters, and Peter Schwabe. "Really Fast Syndrome-Based Hashing". In: *AFRICACRYPT 11*. Ed. by Abderrahmane Nitaj and David Pointcheval. Vol. 6737. LNCS. Springer, Heidelberg, July 2011, pp. 134–152 (cit. on p. 84).

[BM09] Boaz Barak and Mohammad Mahmoody-Ghidary. "Merkle Puzzles Are Optimal - An $O(n^2)$-Query Attack on Any Key Exchange from a Random Oracle". In: *CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. LNCS. Springer, Heidelberg, Aug. 2009, pp. 374–390. DOI: 10.1007/978-3-642-03356-8_22 (cit. on p. 8).

[BM18] Leif Both and Alexander May. "Decoding Linear Codes with High Error Rate and Its Impact for LPN Security". In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*. Ed. by Tanja Lange and Rainer Steinwandt. Springer, Heidelberg, 2018, pp. 25–46. DOI: 10.1007/978-3-319-79063-3_2 (cit. on pp. 72, 77).

[BM90] Mihir Bellare and Silvio Micali. "Non-Interactive Oblivious Transfer and Applications". In: *CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 547–557. DOI: 10.1007/0-387-34805-0_48 (cit. on p. 29).

[BM97] Mihir Bellare and Daniele Micciancio. "A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost". In: *EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. LNCS. Springer, Heidelberg, May 1997, pp. 163–192. DOI: 10.1007/3-540-69053-0_13 (cit. on p. 77).

[BØ23] Pierre Briaud and Morten Øygarden. "A New Algebraic Approach to the Regular Syndrome Decoding Problem and Implications for PCG Constructions". In: LNCS. Springer, Heidelberg, June 2023, pp. 391–422. DOI: 10.1007/978-3-031-30589-4_14 (cit. on p. 84).

[BPT85] Richard Berger, René Peralta, and Tom Tedrick. "A Provably Secure Oblivious Transfer Protocol". In: *EUROCRYPT'84*. Ed. by Thomas Beth, Norbert Cot, and Ingemar Ingemarsson. Vol. 209. LNCS. Springer, Heidelberg, Apr. 1985, pp. 379–386. DOI: 10.1007/3-540-39757-4_26 (cit. on p. 29).

[BR17] Andrej Bogdanov and Alon Rosen. *Pseudorandom Functions: Three Decades Later*. Cryptology ePrint Archive, Report 2017/652. https://eprint.iacr.org/2017/652. 2017 (cit. on p. 77).

[BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. "Average-case fine-grained hardness". In: *49th ACM STOC*. Ed. by Hamed Hatami, Pierre McKenzie, and Valerie King. ACM Press, June 2017, pp. 483–496. DOI: 10.1145/3055399.3055466 (cit. on p. 8).

[BRSV18] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. "Proofs of Work From Worst-Case Assumptions". In: *CRYPTO 2018, Part I.* Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. LNCS. Springer, Heidelberg, Aug. 2018, pp. 789–819. DOI: 10.1007/978-3-319-96884-1_26 (cit. on p. 8).

[BTV16] Sonia Bogos, Florian Tramer, and Serge Vaudenay. "On solving LPN using BKW and variants". In: (2016) (cit. on p. 77).

[BV16] Sonia Bogos and Serge Vaudenay. "Optimization of LPN Solving Algorithms". In: *ASIACRYPT 2016, Part I.* Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 703–728. DOI: 10.1007/978-3-662-53887-6_26 (cit. on p. 77).

[BW13] Dan Boneh and Brent Waters. "Constrained Pseudorandom Functions and Their Applications". In: *ASIACRYPT 2013, Part II.* Ed. by Kazue Sako and Palash Sarkar. Vol. 8270. LNCS. Springer, Heidelberg, Dec. 2013, pp. 280–300. DOI: 10.1007/978-3-642-42045-0_15 (cit. on pp. 85, 86).

[Can00] Ran Canetti. "Security and Composition of Multiparty Cryptographic Protocols". In: *Journal of Cryptology* 13.1 (Jan. 2000), pp. 143–202. DOI: 10.1007/s001459910006 (cit. on p. 24).

[CC18] Pyrros Chaidos and Geoffroy Couteau. "Efficient Designated-Verifier Non-interactive Zero-Knowledge Proofs of Knowledge". In: *EUROCRYPT 2018, Part III.* Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, Apr. 2018, pp. 193–221. DOI: 10.1007/978-3-319-78372-7_7 (cit. on pp. 6, 7).

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. "Multiparty Unconditionally Secure Protocols (Extended Abstract)". In: *20th ACM STOC.* ACM Press, May 1988, pp. 11–19. DOI: 10.1145/62212.62214 (cit. on p. 20).

[CCG+21] Arka Rai Choudhuri, Michele Ciampi, Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. "Oblivious Transfer from Trapdoor Permutations in Minimal Rounds". In: *TCC 2021, Part II.* Ed. by Kobbi Nissim and Brent Waters. Vol. 13043. LNCS. Springer, Heidelberg, Nov. 2021, pp. 518–549. DOI: 10.1007/978-3-030-90453-1_18 (cit. on p. 29).

[CCJ23] Eliana Carozza, Geoffroy Couteau, and Antoine Joux. "Short Signatures from Regular Syndrome Decoding in the Head". In: LNCS. Springer, Heidelberg, June 2023, pp. 532–563. DOI: 10.1007/978-3-031-30589-4_19 (cit. on p. 7).

[CCM98] Christian Cachin, Claude Crépeau, and Julien Marcil. "Oblivious Transfer with a Memory-Bounded Receiver". In: *39th FOCS.* IEEE Computer Society Press, Nov. 1998, pp. 493–502. DOI: 10.1109/SFCS.1998.743500 (cit. on p. 29).

[CD23] Geoffroy Couteau and Clément Ducros. "Pseudorandom Correlation Functions from Variable-Density LPN, Revisited". In: *PKC 2023, Part II.* LNCS. Springer, Heidelberg, May 2023, pp. 221–250. DOI: 10.1007/978-3-031-31371-4_8 (cit. on pp. 53, 77, 98, 99).

[CDLR16] Ignacio Cascudo, Ivan Damgård, Felipe Lacerda, and Samuel Ranellucci. "Oblivious Transfer from Any Non-trivial Elastic Noisy Channel via Secret Key Agreement". In: *TCC 2016-B, Part I*. Ed. by Martin Hirt and Adam D. Smith. Vol. 9985. LNCS. Springer, Heidelberg, Oct. 2016, pp. 204–234. DOI: 10.1007/978-3-662-53641-4_9 (cit. on p. 29).

[CDM+18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. "On the Concrete Security of Goldreich's Pseudorandom Generator". In: *ASIACRYPT 2018, Part II*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. LNCS. Springer, Heidelberg, Dec. 2018, pp. 96–124. DOI: 10.1007/978-3-030-03329-3_4.

[CDMT22] Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. "Statistical Decoding 2.0: Reducing Decoding to LPN". In: *ASIACRYPT 2022, Part IV*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13794. LNCS. Springer, Heidelberg, Dec. 2022, pp. 477–507. DOI: 10.1007/978-3-031-22972-5_17 (cit. on p. 72).

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. "Multiparty Computation from Threshold Homomorphic Encryption". In: *EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Vol. 2045. LNCS. Springer, Heidelberg, May 2001, pp. 280–299. DOI: 10.1007/3-540-44987-6_18 (cit. on pp. 46, 49).

[CFM21] Geoffroy Couteau, Pooya Farshim, and Mohammad Mahmoody. "Black-Box Uselessness: Composing Separations in Cryptography". In: *ITCS 2021*. Ed. by James R. Lee. Vol. 185. LIPIcs, Jan. 2021, 47:1–47:20. DOI: 10.4230/LIPIcs.ITCS.2021.47 (cit. on p. 9).

[CG18] Matteo Campanelli and Rosario Gennaro. "Fine-Grained Secure Computation". In: *TCC 2018, Part II*. Ed. by Amos Beimel and Stefan Dziembowski. Vol. 11240. LNCS. Springer, Heidelberg, Nov. 2018, pp. 66–97. DOI: 10.1007/978-3-030-03810-6_3 (cit. on p. 8).

[CGKR22] Geoffroy Couteau, Dahmun Goudarzi, Michael Klooß, and Michael Reichle. "Sharp: Short Relaxed Range Proofs". In: *ACM CCS 2022*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. ACM Press, Nov. 2022, pp. 609–622. DOI: 10.1145/3548606.3560628 (cit. on p. 7).

[CH19] Geoffroy Couteau and Dennis Hofheinz. "Designated-Verifier Pseudorandom Generators, and Their Applications". In: *EUROCRYPT 2019, Part II*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 562–592. DOI: 10.1007/978-3-030-17656-3_20 (cit. on p. 6).

[CH20] Geoffroy Couteau and Dominik Hartmann. "Shorter Non-interactive Zero-Knowledge Arguments and ZAPs for Algebraic Languages". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 768–798. DOI: 10.1007/978-3-030-56877-1_27 (cit. on pp. 6, 7).

[Cha90] David Chaum. "Showing Credentials without Identification Transferring Signatures between Unconditionally Unlinkable Pseudonyms". In: *AUSCRYPT'90*. Ed. by Jennifer Seberry and Josef Pieprzyk. Vol. 453. LNCS. Springer, Heidelberg, Jan. 1990, pp. 246–264. DOI: 10.1007/BFb0030366 (cit. on p. 7).

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. "Compact E-Cash". In: *EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. LNCS. Springer, Heidelberg, May 2005, pp. 302–321. DOI: 10.1007/11426639_18 (cit. on p. 7).

[CJJQ23] Geoffroy Couteau, Abhishek Jain, Zhengzhong Jin, and Willy Quach. "A Note on Non-Interactive Zero-Knowledge from CDH". In: *Annual International Cryptology Conference*. Springer. 2023 (cit. on p. 6).

[CK88] Claude Crépeau and Joe Kilian. "Achieving Oblivious Transfer Using Weakened Security Assumptions (Extended Abstract)". In: *29th FOCS*. IEEE Computer Society Press, Oct. 1988, pp. 42–52. DOI: 10.1109/SFCS.1988.21920 (cit. on p. 29).

[CK89] Benny Chor and Eyal Kushilevitz. "A Zero-One Law for Boolean Privacy (extended abstract)". In: *21st ACM STOC*. ACM Press, May 1989, pp. 62–72. DOI: 10.1145/73007.73013 (cit. on p. 20).

[CKLR21] Geoffroy Couteau, Michael Klooß, Huang Lin, and Michael Reichle. "Efficient Range Proofs with Transparent Setup from Bounded Integer Commitments". In: *EUROCRYPT 2021, Part III*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12698. LNCS. Springer, Heidelberg, Oct. 2021, pp. 247–277. DOI: 10.1007/978-3-030-77883-5_9 (cit. on p. 7).

[CKS+14] Seung Geol Choi, Jonathan Katz, Dominique Schröder, Arkady Yerukhimovich, and Hong-Sheng Zhou. "(Efficient) Universally Composable Oblivious Transfer Using a Minimal Number of Stateless Tokens". In: *TCC 2014*. Ed. by Yehuda Lindell. Vol. 8349. LNCS. Springer, Heidelberg, Feb. 2014, pp. 638–662. DOI: 10.1007/978-3-642-54242-8_27 (cit. on p. 29).

[CKSU21] Geoffroy Couteau, Shuichi Katsumata, Elahe Sadeghi, and Bogdan Ursu. "Statistical ZAPs from Group-Based Assumptions". In: *TCC 2021, Part I*. Ed. by Kobbi Nissim and Brent Waters. Vol. 13042. LNCS. Springer, Heidelberg, Nov. 2021, pp. 466–498. DOI: 10.1007/978-3-030-90459-3_16 (cit. on p. 6).

[CKU20] Geoffroy Couteau, Shuichi Katsumata, and Bogdan Ursu. "Non-interactive Zero-Knowledge in Pairing-Free Groups from Weaker Assumptions". In: *EUROCRYPT 2020, Part III*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12107. LNCS. Springer, Heidelberg, May 2020, pp. 442–471. DOI: 10.1007/978-3-030-45727-3_15 (cit. on p. 6).

[CKWZ13] Seung Geol Choi, Jonathan Katz, Hoeteck Wee, and Hong-Sheng Zhou. "Efficient, Adaptively Secure, and Composable Oblivious Transfer with a Single, Global CRS". In: *PKC 2013*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Vol. 7778. LNCS. Springer, Heidelberg, Feb. 2013, pp. 73–88. DOI: 10.1007/978-3-642-36362-7_6 (cit. on p. 29).

[CLPØ21] Geoffroy Couteau, Helger Lipmaa, Roberto Parisella, and Arne Tobias Ødegaard. "Efficient NIZKs for Algebraic Sets". In: *ASIACRYPT 2021, Part III*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13092. LNCS. Springer, Heidelberg, Dec. 2021, pp. 128–158. DOI: 10.1007/978-3-030-92078-4_5 (cit. on pp. 6, 7).

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. “Obfuscation of Probabilistic Circuits and Applications”. In: *TCC 2015, Part II.* Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 468–497. DOI: 10.1007/978-3-662-46497-7_19 (cit. on p. 3).

[CM21] Geoffroy Couteau and Pierre Meyer. “Breaking the Circuit Size Barrier for Secure Computation Under Quasi-Polynomial LPN”. In: *EUROCRYPT 2021, Part II.* Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. LNCS. Springer, Heidelberg, Oct. 2021, pp. 842–870. DOI: 10.1007/978-3-030-77886-6_29 (cit. on pp. 4, 53).

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. “Constrained Pseudorandom Functions from Homomorphic Secret Sharing”. In: *EUROCRYPT 2023, Part III.* LNCS. Springer, Heidelberg, June 2023, pp. 194–224. DOI: 10.1007/978-3-031-30620-4_7 (cit. on pp. 4, 9, 53).

[CNs07] Jan Camenisch, Gregory Neven, and abhi shelat. “Simulatable Adaptive Oblivious Transfer”. In: *EUROCRYPT 2007.* Ed. by Moni Naor. Vol. 4515. LNCS. Springer, Heidelberg, May 2007, pp. 573–590. DOI: 10.1007/978-3-540-72540-4_33 (cit. on p. 29).

[CO15] Tung Chou and Claudio Orlandi. “The Simplest Protocol for Oblivious Transfer”. In: *LATINCRYPT 2015.* Ed. by Kristin E. Lauter and Francisco Rodríguez-Henríquez. Vol. 9230. LNCS. Springer, Heidelberg, Aug. 2015, pp. 40–58. DOI: 10.1007/978-3-319-22174-8_3 (cit. on p. 29).

[Cou18] Geoffroy Couteau. “New Protocols for Secure Equality Test and Comparison”. In: *ACNS 18.* Ed. by Bart Preneel and Frederik Vercauteren. Vol. 10892. LNCS. Springer, Heidelberg, July 2018, pp. 303–320. DOI: 10.1007/978-3-319-93387-0_16 (cit. on p. 5).

[Cou19] Geoffroy Couteau. “A Note on the Communication Complexity of Multiparty Computation in the Correlated Randomness Model”. In: *EUROCRYPT 2019, Part II.* Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11477. LNCS. Springer, Heidelberg, May 2019, pp. 473–503. DOI: 10.1007/978-3-030-17656-3_17 (cit. on p. 4).

[CPP16] Geoffroy Couteau, Thomas Peters, and David Pointcheval. “Encryption Switching Protocols”. In: *CRYPTO 2016, Part I.* Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 308–338. DOI: 10.1007/978-3-662-53018-4_12 (cit. on p. 5).

[CPP17] Geoffroy Couteau, Thomas Peters, and David Pointcheval. “Removing the Strong RSA Assumption from Arguments over the Integers”. In: *EUROCRYPT 2017, Part II.* Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. LNCS. Springer, Heidelberg, Apr. 2017, pp. 321–350. DOI: 10.1007/978-3-319-56614-6_11 (cit. on p. 7).

[CR19] Geoffroy Couteau and Michael Reichle. “Non-interactive Keyed-Verification Anonymous Credentials”. In: *PKC 2019, Part I.* Ed. by Dongdai Lin and Kazue Sako. Vol. 11442. LNCS. Springer, Heidelberg, Apr. 2019, pp. 66–96. DOI: 10.1007/978-3-030-17253-4_3 (cit. on p. 7).

[CR22] Geoffroy Couteau and Adi Rosén. "Random Sources in Private Computation". In: *ASIACRYPT 2022, Part I*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13791. LNCS. Springer, Heidelberg, Dec. 2022, pp. 443–473. DOI: 10.1007/978-3-031-22963-3_15 (cit. on p. 9).

[CRR21a] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. "Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes". In: *CRYPTO 2021, Part III*. Ed. by Tal Malkin and Chris Peikert. Vol. 12827. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 502–534. DOI: 10.1007/978-3-030-84252-9_17 (cit. on pp. 53, 61, 77, 80).

[CRR21b] Geoffroy Couteau, A. W. Roscoe, and Peter Y. A. Ryan. "Partially-Fair Computation from Timed-Release Encryption and Oblivious Transfer". In: *ACISP 21*. Ed. by Joonsang Baek and Sushmita Ruj. Vol. 13083. LNCS. Springer, Heidelberg, Dec. 2021, pp. 330–349. DOI: 10.1007/978-3-030-90567-5_17 (cit. on p. 5).

[CSW20] Ran Canetti, Pratik Sarkar, and Xiao Wang. "Efficient and Round-Optimal Oblivious Transfer and Commitment with Adaptive Security". In: *ASIACRYPT 2020, Part III*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12493. LNCS. Springer, Heidelberg, Dec. 2020, pp. 277–308. DOI: 10.1007/978-3-030-64840-4_10 (cit. on p. 29).

[CZ22] Geoffroy Couteau and Maryam Zarezadeh. "Non-interactive Secure Computation of Inner-Product from LPN and LWE". In: *ASIACRYPT 2022, Part I*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13791. LNCS. Springer, Heidelberg, Dec. 2022, pp. 474–503. DOI: 10.1007/978-3-031-22963-3_16 (cit. on p. 5).

[DFMS04] Ivan Damgård, Serge Fehr, Kirill Morozov, and Louis Salvail. "Unfair Noisy Channels and Oblivious Transfer". In: *TCC 2004*. Ed. by Moni Naor. Vol. 2951. LNCS. Springer, Heidelberg, Feb. 2004, pp. 355–373. DOI: 10.1007/978-3-540-24638-1_20 (cit. on p. 29).

[DGH+20] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. "Two-Round Oblivious Transfer from CDH or LPN". In: *EUROCRYPT 2020, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. LNCS. Springer, Heidelberg, May 2020, pp. 768–797. DOI: 10.1007/978-3-030-45724-2_26 (cit. on p. 29).

[DH21] Itai Dinur and Ben Hasson. "Distributed Merkle's Puzzles". In: *TCC 2021, Part II*. Ed. by Kobbi Nissim and Brent Waters. Vol. 13043. LNCS. Springer, Heidelberg, Nov. 2021, pp. 310–332. DOI: 10.1007/978-3-030-90453-1_11 (cit. on p. 8).

[DH76] Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638 (cit. on p. 33).

[DHRS04] Yan Zong Ding, Danny Harnik, Alon Rosen, and Ronen Shaltiel. "Constant-Round Oblivious Transfer in the Bounded Storage Model". In: *TCC 2004*. Ed. by Moni Naor. Vol. 2951. LNCS. Springer, Heidelberg, Feb. 2004, pp. 446–472. DOI: 10.1007/978-3-540-24638-1_25 (cit. on p. 29).

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. “Spooky Encryption and Its Applications”. In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 93–122. DOI: 10.1007/978-3-662-53015-3_4 (cit. on p. 103).

[DIJL23] 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*. Ed. by Helena Handschuh and Anna Lysyanskaya. Springer Nature Switzerland, 2023, pp. 315–348. ISBN: 978-3-031-38545-2 (cit. on p. 4).

[Din01] Yan Zong Ding. “Oblivious Transfer in the Bounded Storage Model”. In: *CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. LNCS. Springer, Heidelberg, Aug. 2001, pp. 155–170. DOI: 10.1007/3-540-44647-8_9 (cit. on p. 29).

[DP12] Ivan Damgård and Sunoo Park. *How Practical is Public-Key Encryption Based on LPN and Ring-LPN?* Cryptology ePrint Archive, Report 2012/699. https://eprint.iacr.org/2012/699. 2012 (cit. on p. 91).

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. “Multiparty Computation from Somewhat Homomorphic Encryption”. In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38 (cit. on p. 49).

[DT17] Thomas Debris-Alazard and Jean-Pierre Tillich. “Statistical decoding”. In: 2017 (cit. on pp. 72, 77).

[DVV16] Akshay Degwekar, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. “Fine-Grained Cryptography”. In: *CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 533–562. DOI: 10.1007/978-3-662-53015-3_19 (cit. on p. 8).

[EGL82] Shimon Even, Oded Goldreich, and Abraham Lempel. “A Randomized Protocol for Signing Contracts”. In: *CRYPTO’82*. Ed. by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Plenum Press, New York, USA, 1982, pp. 205–210 (cit. on p. 28).

[EKM17] Andre Esser, Robert Kübler, and Alexander May. “LPN Decoded”. In: *CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. LNCS. Springer, Heidelberg, Aug. 2017, pp. 486–514. DOI: 10.1007/978-3-319-63715-0_17 (cit. on pp. 72, 77).

[ES23] Andre Esser and Paolo Santini. “Not just regular decoding: Asymptotics and improvements of regular syndrome decoding attacks”. In: *Cryptology ePrint Archive* (2023) (cit. on p. 84).

[EWT21] Shohei Egashira, Yuyu Wang, and Keisuke Tanaka. “Fine-Grained Cryptography Revisited”. In: *Journal of Cryptology* 34.3 (July 2021), p. 23. DOI: 10.1007/s00145-021-09390-3 (cit. on p. 8).

[Fei02] Uriel Feige. “Relations between average case complexity and approximation complexity”. In: *34th ACM STOC*. ACM Press, May 2002, pp. 534–543. DOI: 10.1145/509907.509985 (cit. on p. 72).

[FGS07] Matthieu Finiasz, Philippe Gaborit, and Nicolas Sendrier. “Improved fast syndrome based cryptographic hash functions”. In: *Proceedings of ECRYPT Hash Workshop*. Vol. 2007. Citeseer. 2007, p. 155 (cit. on p. 84).

[FKI06] Marc PC Fossorier, Kazukuni Kobara, and Hideki Imai. “Modeling bit flipping decoding based on nonorthogonal check sums with application to iterative decoding attack of McEliece cryptosystem”. In: *IEEE Transactions on Information Theory* 53.1 (2006), pp. 402–411 (cit. on pp. 72, 77).

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. “Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String (Extended Abstract)”. In: *31st FOCS*. IEEE Computer Society Press, Oct. 1990, pp. 308–317. DOI: 10.1109/FSCS.1990.89549 (cit. on p. 5).

[FMV19] Daniele Friolo, Daniel Masny, and Daniele Venturi. “A Black-Box Construction of Fully-Simulatable, Round-Optimal Oblivious Transfer from Strongly Uniform Key Agreement”. In: *TCC 2019, Part I*. Ed. by Dennis Hofheinz and Alon Rosen. Vol. 11891. LNCS. Springer, Heidelberg, Dec. 2019, pp. 111–130. DOI: 10.1007/978-3-030-36030-6_5 (cit. on p. 29).

[FS09] Matthieu Finiasz and Nicolas Sendrier. “Security Bounds for the Design of Code-Based Cryptosystems”. In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 88–105. DOI: 10.1007/978-3-642-10366-7_6 (cit. on pp. 72, 77).

[Gen09] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *41st ACM STOC*. Ed. by Michael Mitzenmacher. ACM Press, May 2009, pp. 169–178. DOI: 10.1145/1536414.1536440 (cit. on pp. 3, 25, 31).

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. “How to Construct Random Functions”. In: *Journal of the ACM* 33.4 (Oct. 1986), pp. 792–807 (cit. on pp. 85, 86).

[GH08] Matthew Green and Susan Hohenberger. “Universally Composable Adaptive Oblivious Transfer”. In: *ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Vol. 5350. LNCS. Springer, Heidelberg, Dec. 2008, pp. 179–197. DOI: 10.1007/978-3-540-89255-7_12 (cit. on p. 29).

[GH11] Matthew Green and Susan Hohenberger. “Practical Adaptive Oblivious Transfer from Simple Assumptions”. In: *TCC 2011*. Ed. by Yuval Ishai. Vol. 6597. LNCS. Springer, Heidelberg, Mar. 2011, pp. 347–363. DOI: 10.1007/978-3-642-19571-6_21 (cit. on p. 29).

[GI14] Niv Gilboa and Yuval Ishai. “Distributed Point Functions and Their Applications”. In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 640–658. DOI: 10.1007/978-3-642-55220-5_35 (cit. on p. 68).

[GI99] Niv Gilboa and Yuval Ishai. “Compressing Cryptographic Resources”. In: *CRYPTO’99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 591–608. DOI: 10.1007/3-540-48405-1_37 (cit. on p. 61).

[Gil99] Niv Gilboa. “Two Party RSA Key Generation”. In: *CRYPTO’99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 116–129. DOI: 10.1007/3-540-48405-1_8 (cit. on p. 45).

[GJJM20] Vipul Goyal, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. "Statistical Zaps and New Oblivious Transfer Protocols". In: *EUROCRYPT 2020, Part III*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12107. LNCS. Springer, Heidelberg, May 2020, pp. 668–699. DOI: 10.1007/978-3-030-45727-3_23 (cit. on p. 29).

[GJL15] Qian Guo, Thomas Johansson, and Carl Löndahl. "A new algorithm for solving Ring-LPN with a reducible polynomial". In: *IEEE Transactions on Information Theory* 61.11 (2015), pp. 6204–6212 (cit. on p. 91).

[GJL20] Qian Guo, Thomas Johansson, and Carl Löndahl. "Solving LPN Using Covering Codes". In: *Journal of Cryptology* 33.1 (Jan. 2020), pp. 1–33. DOI: 10.1007/s00145-019-09338-8 (cit. on p. 77).

[GKM+00] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, and Mahesh Viswanathan. "The Relationship between Public Key Encryption and Oblivious Transfer". In: *41st FOCS*. IEEE Computer Society Press, Nov. 2000, pp. 325–335. DOI: 10.1109/SFCS.2000.892121 (cit. on p. 29).

[GLM22] S. Dov Gordon, Phi Hung Le, and Daniel McVicker. *Linear Communication in Malicious Majority MPC*. Cryptology ePrint Archive, Report 2022/781. https://eprint.iacr.org/2022/781. 2022 (cit. on p. 49).

[GLSV21] Alex B. Grilo, Huijia Lin, Fang Song, and Vinod Vaikuntanathan. "Oblivious Transfer Is in MiniQCrypt". In: *EUROCRYPT 2021, Part II*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. LNCS. Springer, Heidelberg, Oct. 2021, pp. 531–561. DOI: 10.1007/978-3-030-77886-6_18 (cit. on p. 29).

[GM00] Juan A. Garay and Philip D. MacKenzie. "Concurrent Oblivious Transfer". In: *41st FOCS*. IEEE Computer Society Press, Nov. 2000, pp. 314–324. DOI: 10.1109/SFCS.2000.892120 (cit. on p. 29).

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208 (cit. on p. 5).

[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 ACM STOC*. Ed. by Alfred Aho. ACM Press, May 1987, pp. 218–229. DOI: 10.1145/28395.28420 (cit. on p. 40).

[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: *CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Heidelberg, Aug. 1987, pp. 171–185. DOI: 10.1007/3-540-47721-7_11 (cit. on pp. 19, 24, 25).

[Gol00] Oded Goldreich. *Candidate One-Way Functions Based on Expander Graphs*. Cryptology ePrint Archive, Report 2000/063. https://eprint.iacr.org/2000/063. 2000 (cit. on p. 33).

[Gol09] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009 (cit. on p. 24).

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. “Perfect Non-interactive Zero Knowledge for NP”. In: *EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, May 2006, pp. 339–358. DOI: 10.1007/11761679_21 (cit. on p. 5).

[Gro05] Jens Groth. “Non-interactive Zero-Knowledge Arguments for Voting”. In: *ACNS 05*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Vol. 3531. LNCS. Springer, Heidelberg, June 2005, pp. 467–482. DOI: 10.1007/11496137_32 (cit. on p. 7).

[GS08] Jens Groth and Amit Sahai. “Efficient Non-interactive Proof Systems for Bilinear Groups”. In: *EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. LNCS. Springer, Heidelberg, Apr. 2008, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24 (cit. on p. 5).

[GV88] Oded Goldreich and Ronen Vainish. “How to Solve any Protocol Problem - An Efficiency Improvement”. In: *CRYPTO’87*. Ed. by Carl Pomerance. Vol. 293. LNCS. Springer, Heidelberg, Aug. 1988, pp. 73–86. DOI: 10.1007/3-540-48184-2_6 (cit. on p. 28).

[GWZ09] Juan A. Garay, Daniel Wichs, and Hong-Sheng Zhou. “Somewhat Non-committing Encryption and Efficient Adaptively Secure Oblivious Transfer”. In: *CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. LNCS. Springer, Heidelberg, Aug. 2009, pp. 505–523. DOI: 10.1007/978-3-642-03356-8_30 (cit. on p. 29).

[GYW+23] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. “Half-Tree: Halving the Cost of Tree Expansion in COT and DPF”. In: *EUROCRYPT 2023, Part I*. LNCS. Springer, Heidelberg, June 2023, pp. 330–362. DOI: 10.1007/978-3-031-30545-0_12 (cit. on p. 88).

[Hai04] Iftach Haitner. “Implementing Oblivious Transfer Using Collection of Dense Trapdoor Permutations”. In: *TCC 2004*. Ed. by Moni Naor. Vol. 2951. LNCS. Springer, Heidelberg, Feb. 2004, pp. 394–409. DOI: 10.1007/978-3-540-24638-1_22 (cit. on p. 29).

[Hai08] Iftach Haitner. “Semi-honest to Malicious Oblivious Transfer - The Black-Box Way”. In: *TCC 2008*. Ed. by Ran Canetti. Vol. 4948. LNCS. Springer, Heidelberg, Mar. 2008, pp. 412–426. DOI: 10.1007/978-3-540-78524-8_23 (cit. on p. 29).

[HCR02] Dowon Hong, Ku-Young Chang, and Heuisu Ryu. “Efficient Oblivious Transfer in the Bounded-Storage Model”. In: *ASIACRYPT 2002*. Ed. by Yuliang Zheng. Vol. 2501. LNCS. Springer, Heidelberg, Dec. 2002, pp. 143–159. DOI: 10.1007/3-540-36178-2_9 (cit. on p. 29).

[HKL+12] Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. “Lapin: An Efficient Authentication Protocol Based on Ring-LPN”. In: *FSE 2012*. Ed. by Anne Canteaut. Vol. 7549. LNCS. Springer, Heidelberg, Mar. 2012, pp. 346–365. DOI: 10.1007/978-3-642-34047-5_20 (cit. on p. 91).

[HOSS18] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. "TinyKeys: A New Approach to Efficient Multi-Party Computation". In: *CRYPTO 2018, Part III*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10993. LNCS. Springer, Heidelberg, Aug. 2018, pp. 3–33. DOI: 10.1007/978-3-319-96878-0_1 (cit. on p. 84).

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. "Extending Oblivious Transfers Efficiently". In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 145–161. DOI: 10.1007/978-3-540-45146-4_9 (cit. on pp. 36, 81, 83, 88).

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and Jürg Wullschleger. "Constant-Rate Oblivious Transfer from Noisy Channels". In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 667–684. DOI: 10.1007/978-3-642-22792-9_38 (cit. on p. 29).

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. "Founding Cryptography on Oblivious Transfer - Efficiently". In: *CRYPTO 2008*. Ed. by David Wagner. Vol. 5157. LNCS. Springer, Heidelberg, Aug. 2008, pp. 572–591. DOI: 10.1007/978-3-540-85174-5_32 (cit. on p. 28).

[IR89] Russell Impagliazzo and Steven Rudich. "Limits on the Provable Consequences of One-Way Permutations". In: *21st ACM STOC*. ACM Press, May 1989, pp. 44–61. DOI: 10.1145/73007.73012 (cit. on p. 36).

[Kal05] Yael Tauman Kalai. "Smooth Projective Hashing and Two-Message Oblivious Transfer". In: *EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. LNCS. Springer, Heidelberg, May 2005, pp. 78–95. DOI: 10.1007/11426639_5 (cit. on p. 29).

[Kil88] Joe Kilian. "Founding Cryptography on Oblivious Transfer". In: *20th ACM STOC*. ACM Press, May 1988, pp. 20–31. DOI: 10.1145/62212.62215 (cit. on p. 28).

[Kir11] Paul Kirchner. *Improved Generalized Birthday Attack*. Cryptology ePrint Archive, Report 2011/377. https://eprint.iacr.org/2011/377. 2011 (cit. on pp. 72, 77).

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. "Improved OT Extension for Transferring Short Secrets". In: *CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. LNCS. Springer, Heidelberg, Aug. 2013, pp. 54–70. DOI: 10.1007/978-3-642-40084-1_4 (cit. on p. 37).

[KN09] Kaoru Kurosawa and Ryo Nojima. "Simple Adaptive Oblivious Transfer without Random Oracle". In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 334–346. DOI: 10.1007/978-3-642-10366-7_20 (cit. on p. 29).

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. "Actively Secure OT Extension with Optimal Overhead". In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 724–741. DOI: 10.1007/978-3-662-47989-6_35 (cit. on p. 37).

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. “Overdrive: Making SPDZ Great Again”. In: *EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, Apr. 2018, pp. 158–189. DOI: 10.1007/978-3-319-78372-7_6 (cit. on p. 49).

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. “Delegatable pseudorandom functions and applications”. In: *ACM CCS 2013*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 669–684. DOI: 10.1145/2508859.2516668 (cit. on pp. 85, 86).

[KSS10] Jonathan Katz, Ji Sun Shin, and Adam Smith. “Parallel and Concurrent Security of the HB and HB+ Protocols”. In: *Journal of Cryptology* 23.3 (July 2010), pp. 402–421. DOI: 10.1007/s00145-010-9061-2 (cit. on p. 73).

[KsS12] Benjamin Kreuter, abhi shelat, and Chih-Hao Shen. “Billion-Gate Secure Computation with Malicious Adversaries”. In: *USENIX Security 2012*. Ed. by Tadayoshi Kohno. USENIX Association, Aug. 2012, pp. 285–300 (cit. on p. 30).

[LF06] Éric Levieil and Pierre-Alain Fouque. “An Improved LPN Algorithm”. In: *SCN 06*. Ed. by Roberto De Prisco and Moti Yung. Vol. 4116. LNCS. Springer, Heidelberg, Sept. 2006, pp. 348–359. DOI: 10.1007/11832072_24 (cit. on p. 77).

[LGD21] Yi-Fu Lai, Steven D. Galbraith, and Cyprien Delpech de Saint Guilhem. “Compact, Efficient and UC-Secure Isogeny-Based Oblivious Transfer”. In: *EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. LNCS. Springer, Heidelberg, Oct. 2021, pp. 213–241. DOI: 10.1007/978-3-030-77870-5_8 (cit. on p. 29).

[Lip05] Helger Lipmaa. “An Oblivious Transfer Protocol with Log-Squared Communication”. In: *ISC 2005*. Ed. by Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao. Vol. 3650. LNCS. Springer, Heidelberg, Sept. 2005, pp. 314–328 (cit. on p. 29).

[LLW19] Rio LaVigne, Andrea Lincoln, and Virginia Vassilevska Williams. “Public-Key Cryptography in the Fine-Grained Setting”. In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 605–635. DOI: 10.1007/978-3-030-26954-8_20 (cit. on p. 8).

[LP15] Helger Lipmaa and Kateryna Pavlyk. “Analysis and Implementation of an Efficient Ring-LPN Based Commitment Scheme”. In: *CANS 15*. Ed. by Michael Reiter and David Naccache. LNCS. Springer, Heidelberg, Dec. 2015, pp. 160–175. DOI: 10.1007/978-3-319-26823-1_12 (cit. on p. 91).

[LWYY22] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. *The Hardness of LPN over Any Integer Ring and Field for PCG Applications*. Cryptology ePrint Archive, Report 2022/712. https://eprint.iacr.org/2022/712. 2022 (cit. on p. 88).

[Lyu05] Vadim Lyubashevsky. “The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem”. In: 2005 (cit. on pp. 72, 77).

[LZ09] Yehuda Lindell and Hila Zarosim. "Adaptive Zero-Knowledge Proofs and Adaptively Secure Oblivious Transfer". In: *TCC 2009*. Ed. by Omer Reingold. Vol. 5444. LNCS. Springer, Heidelberg, Mar. 2009, pp. 183–201. DOI: 10.1007/978-3-642-00457-5_12 (cit. on p. 29).

[LZ13] Yehuda Lindell and Hila Zarosim. "On the Feasibility of Extending Oblivious Transfer". In: *TCC 2013*. Ed. by Amit Sahai. Vol. 7785. LNCS. Springer, Heidelberg, Mar. 2013, pp. 519–538. DOI: 10.1007/978-3-642-36594-2_29 (cit. on p. 37).

[MBD+18] Carlos Aguilar Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. "Efficient Encryption From Random Quasi-Cyclic Codes". In: *IEEE Trans. Information Theory* 64.5 (2018), pp. 3927–3943. DOI: 10.1109/TIT.2018.2804444. URL: https://doi.org/10.1109/TIT.2018.2804444 (cit. on p. 91).

[MDCY11] Mohammed Meziani, Özgür Dagdelen, Pierre-Louis Cayrel, and Sidi Mohamed El Yousfi Alaoui. "S-FSB: An improved variant of the FSB hash family". In: *International Conference on Information Security and Assurance*. Springer. 2011, pp. 132–145 (cit. on p. 84).

[Mer74] R. Merkle. "C.S. 244 Project Proposal". In: *Facsimile available at http://www.merkle.com/1974*. 1974 (cit. on p. 8).

[Mer78] Ralph C Merkle. "Secure communications over insecure channels". In: *Communications of the ACM* 21.4 (1978), pp. 294–299 (cit. on p. 8).

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. "Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$". In: *ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 107–124. DOI: 10.1007/978-3-642-25385-0_6 (cit. on pp. 72, 77).

[MO15] Alexander May and Ilya Ozerov. "On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes". In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 203–228. DOI: 10.1007/978-3-662-46800-5_9 (cit. on pp. 72, 77).

[MR19] Daniel Masny and Peter Rindal. "Endemic Oblivious Transfer". In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 309–326. DOI: 10.1145/3319535.3354210 (cit. on p. 29).

[MR92] Silvio Micali and Phillip Rogaway. "Secure Computation (Abstract)". In: *CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 392–404. DOI: 10.1007/3-540-46766-1_32 (cit. on p. 24).

[MRR21] Ian McQuoid, Mike Rosulek, and Lawrence Roy. "Batching Base Oblivious Transfers". In: *ASIACRYPT 2021, Part III*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13092. LNCS. Springer, Heidelberg, Dec. 2021, pp. 281–310. DOI: 10.1007/978-3-030-92078-4_10 (cit. on p. 29).

[MS20] Daniele Micciancio and Jessica Sorrell. “Simpler Statistically Sender Private Oblivious Transfer from Ideals of Cyclotomic Integers”. In: *ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, Heidelberg, Dec. 2020, pp. 381–407. DOI: 10.1007/978-3-030-64834-3_13 (cit. on p. 29).

[MST03] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. “On e-Biased Generators in NC0”. In: *44th FOCS*. IEEE Computer Society Press, Oct. 2003, pp. 136–145. DOI: 10.1109/SFCS.2003.1238188 (cit. on p. 76).

[MSY21] Jean-Pierre Münch, Thomas Schneider, and Hossein Yalame. *VASA: Vector AES Instructions for Security Applications*. Cryptology ePrint Archive, Report 2021/1493. https://eprint.iacr.org/2021/1493. 2021 (cit. on p. 83).

[NN90] Joseph Naor and Moni Naor. “Small-bias Probability Spaces: Efficient Constructions and Applications”. In: *22nd ACM STOC*. ACM Press, May 1990, pp. 213–223. DOI: 10.1145/100216.100244 (cit. on p. 76).

[NP01] Moni Naor and Benny Pinkas. “Efficient Oblivious Transfer Protocols”. In: *12th SODA*. Ed. by S. Rao Kosaraju. ACM-SIAM, Jan. 2001, pp. 448–457 (cit. on pp. 29, 30).

[NP99a] Moni Naor and Benny Pinkas. “Oblivious Transfer and Polynomial Evaluation”. In: *31st ACM STOC*. ACM Press, May 1999, pp. 245–254. DOI: 10.1145/301250.301312 (cit. on p. 49).

[NP99b] Moni Naor and Benny Pinkas. “Oblivious Transfer with Adaptive Queries”. In: *CRYPTO’99*. Ed. by Michael J. Wiener. Vol. 1666. LNCS. Springer, Heidelberg, Aug. 1999, pp. 573–590. DOI: 10.1007/3-540-48405-1_36 (cit. on p. 29).

[NY89] Moni Naor and Moti Yung. “Universal One-Way Hash Functions and their Cryptographic Applications”. In: *21st ACM STOC*. ACM Press, May 1989, pp. 33–43. DOI: 10.1145/73007.73011 (cit. on p. 34).

[OOS17] Michele Orrù, Emmanuela Orsini, and Peter Scholl. “Actively Secure 1-out-of-N OT Extension with Application to Private Set Intersection”. In: *CT-RSA 2017*. Ed. by Helena Handschuh. Vol. 10159. LNCS. Springer, Heidelberg, Feb. 2017, pp. 381–396. DOI: 10.1007/978-3-319-52153-4_22 (cit. on p. 37).

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. “The Rise of Paillier: Homomorphic Secret Sharing and Public-Key Silent OT”. In: *EUROCRYPT 2021, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. LNCS. Springer, Heidelberg, Oct. 2021, pp. 678–708. DOI: 10.1007/978-3-030-77870-5_24 (cit. on pp. 4, 106).

[Ove06] Raphael Overbeck. “Statistical Decoding Revisited”. In: *ACISP 06*. Ed. by Lynn Margaret Batten and Reihaneh Safavi-Naini. Vol. 4058. LNCS. Springer, Heidelberg, July 2006, pp. 283–294 (cit. on pp. 72, 77).

[Pai99] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *EUROCRYPT’99*. Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16 (cit. on p. 48).

[Pra62] Eugene Prange. “The use of information sets in decoding cyclic codes”. In: (1962) (cit. on pp. 72, 77).

[PS19] Chris Peikert and Sina Shiehian. “Noninteractive Zero Knowledge for NP from (Plain) Learning with Errors”. In: *CRYPTO 2019, Part I*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. LNCS. Springer, Heidelberg, Aug. 2019, pp. 89–114. DOI: 10.1007/978-3-030-26948-7_4 (cit. on p. 5).

[PSS17] Arpita Patra, Pratik Sarkar, and Ajith Suresh. “Fast Actively Secure OT Extension for Short Secrets”. In: *NDSS 2017*. The Internet Society, Feb. 2017 (cit. on p. 37).

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. “A Framework for Efficient and Composable Oblivious Transfer”. In: *CRYPTO 2008*. Ed. by David Wagner. Vol. 5157. LNCS. Springer, Heidelberg, Aug. 2008, pp. 554–571. DOI: 10.1007/978-3-540-85174-5_31 (cit. on p. 29).

[Rab81] Michael Rabin. *How to exchange secrets with oblivious transfer*. Tech. rep. Technical Report TR-81. Aiken Computation Lab, Harvard University, 1981 (cit. on p. 28).

[Reg05] Oded Regev. “On lattices, learning with errors, random linear codes, and cryptography”. In: *37th ACM STOC*. Ed. by Harold N. Gabow and Ronald Fagin. ACM Press, May 2005, pp. 84–93. DOI: 10.1145/1060590.1060603 (cit. on p. 48).

[Rom90] John Rompel. “One-Way Functions are Necessary and Sufficient for Secure Signatures”. In: *22nd ACM STOC*. ACM Press, May 1990, pp. 387–394. DOI: 10.1145/100216.100269 (cit. on p. 34).

[RRT23] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. “Expand-Convolute Codes for Pseudorandom Correlation Generators from LPN”. In: *CRYPTO 2023*. Springer. 2023 (cit. on pp. 77, 80).

[RS21] Lawrence Roy and Jaspal Singh. “Large Message Homomorphic Secret Sharing from DCR and Applications”. In: *CRYPTO 2021, Part III*. Ed. by Tal Malkin and Chris Peikert. Vol. 12827. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 687–717. DOI: 10.1007/978-3-030-84252-9_23 (cit. on p. 4).

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Communications of the Association for Computing Machinery* 21.2 (Feb. 1978), pp. 120–126. DOI: 10.1145/359340.359342 (cit. on p. 33).

[Saa07] Markku-Juhani Olavi Saarinen. “Linearization Attacks Against Syndrome Based Hashes”. In: *INDOCRYPT 2007*. Ed. by K. Srinathan, C. Pandu Rangan, and Moti Yung. Vol. 4859. LNCS. Springer, Heidelberg, Dec. 2007, pp. 1–9 (cit. on p. 77).

[Ste88] Jacques Stern. “A method for finding codewords of small weight”. In: 1988 (cit. on pp. 72, 77).

[SW14] Amit Sahai and Brent Waters. “How to use indistinguishability obfuscation: deniable encryption, and more”. In: *46th ACM STOC*. Ed. by David B. Shmoys. ACM Press, May 2014, pp. 475–484. DOI: 10.1145/2591796.2591825 (cit. on p. 5).

[Wag02] David Wagner. “A Generalized Birthday Problem”. In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Heidelberg, Aug. 2002, pp. 288–303. DOI: 10.1007/3-540-45708-9_19 (cit. on pp. 72, 77).

[Wie83] Stephen Wiesner. “Conjugate coding”. In: *ACM Sigact News* 15.1 (1983), pp. 78–88 (cit. on p. 28).

[WP22] Yuyu Wang and Jiaxin Pan. “Non-Interactive Zero-Knowledge Proofs with Fine-Grained Security”. In: *EUROCRYPT 2022, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. LNCS. Springer, Heidelberg, May 2022, pp. 305–335. DOI: 10.1007/978-3-031-07085-3_11 (cit. on p. 8).

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. “Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation”. In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 21–37. DOI: 10.1145/3133956.3134053 (cit. on p. 25).

[Wul09] Jürg Wullschleger. “Oblivious Transfer from Weak Noisy Channels”. In: *TCC 2009*. Ed. by Omer Reingold. Vol. 5444. LNCS. Springer, Heidelberg, Mar. 2009, pp. 332–349. DOI: 10.1007/978-3-642-00457-5_20 (cit. on p. 29).

[Yao82] Andrew Chi-Chih Yao. “Protocols for Secure Computations (Extended Abstract)”. In: *23rd FOCS*. IEEE Computer Society Press, Nov. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38 (cit. on pp. 19, 24, 25).

[Zic17] Lior Zichron. *Locally Computable Arithmetic Pseudorandom Generators*. 2017 (cit. on p. 76).

[ZJW16] Bin Zhang, Lin Jiao, and Mingsheng Wang. “Faster Algorithms for Solving LPN”. In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 168–195. DOI: 10.1007/978-3-662-49890-3_7 (cit. on p. 77).