Unconditionally Secure Cryptographic Protocols from Coding-Theoretic Primitives

Proefschrift
ter verkrijging van
de graad van Doctor aan de Universiteit Leiden
op gezag van Rector Magnificus prof. mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 6 december 2017
klokke 12:30 uur

door

Gabriele Spini
geboren te Sondrio, Italië,
in 1989

Promotores:

Prof. dr. Ronald Cramer (CWI, Amsterdam & Universiteit Leiden)

Prof. dr. Gilles Zémor (Université de Bordeaux)

Copromotor:

Dr. Serge Fehr (CWI, Amsterdam)

Samenstelling van de promotiecommissie:

Prof. dr. Yuval Ishai (Technion, Haifa)

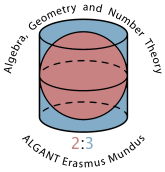
Dr. Emmanuela Orsini (University of Bristol)

Dr. Berry Schoenmakers (Technische Universiteit Eindhoven)

Prof. dr. Bart de Smit (Universiteit Leiden)

Prof. dr. Aad van der Vaart (Universiteit Leiden)

This work was funded by Erasmus Mundus Algant-Doc and was carried out at Universiteit Leiden, Université de Bordeaux and CWI Amsterdam.



Universiteit Leiden

université de BORDEAUX

CWI

Universiteit Leiden

THÈSE EN COTUTELLE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR

DE L'UNIVERSITÉ DE BORDEAUX

ET DE L'UNIVERSITÉ DE LEYDE

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET INFORMATIQUE

INSTITUT DES MATHÉMATIQUES DE L'UNIVERSITÉ DE LEYDE

SPÉCIALITÉ Mathématiques Pures

Par Gabriele SPINI

Protocoles avec Sécurité Inconditionnelle issus de Techniques de la Théorie des Codes

Sous la direction de Ronald CRAMER, Serge FEHR et Gilles ZÉMOR

Soutenue le 6 décembre 2017

Membres du jury :

Anne CANTEAUT	Directrice de recherche, Inria Paris	Examinatrice
Bart DE SMIT	Universiteit Leiden	Examinateur
Yuval ISHAI	Professeur, Technion, Haifa	Rapporteur
Berry SCHOENMAKERS	Universitair Hoofddocent, TU Eindhoven	Rapporteur
Aad VAN DER VAART	Professeur, Universiteit Leiden	Examinateur
Gilles ZÉMOR	Professeur, Université de Bordeaux	Directeur

Contents

Chapter 1

Introduction

In this thesis, we introduce new unconditionally secure cryptographic protocols, constructed and analyzed with techniques from Coding Theory. We present here the context of our work and summarize our contributions.

1.1 Context

Historically, Cryptography originated from the need for private communication: since ancient times, the transmission of written messages has been vulnerable to eavesdropping; for instance, enemy forces could intercept a courier carrying a message between two allied generals. Thus to protect sensitive information from such interception attacks, so-called *encryption* techniques were developed.

An encryption scheme allows a sender to turn a message into a *ciphertext,* which is transmitted to the intended receiver; the receiver then uses a secret *key* to recover the original message from the ciphertext. The goal of the scheme is to ensure that no information on the original message can be extracted from the ciphertext without knowing the secret key; thus any eavesdropper that intercepts the communication from sender to receiver would obtain no information about the actual message.

Historical examples of encryption schemes are abundant, but encryption was more empirical than scientific until the 1940s and the seminal work of Claude E. Shannon [60, 61]. Shannon's work provided a mathematical definition of what it means for an encryption scheme to be secure, and established bounds and possibility results for schemes that are secure according to his definition.

Shannon's notion of security is *information-theoretic,* meaning that it holds regardless of the computing power of the eavesdropper; Shannon showed that information-theoretic security can be achieved, but that it requires the secret key to be as long as the secret message.

The starting point of another major transition in Cryptography is the 1970s work of Diffie and Hellman, whose seminal paper [26] would inspire research on new areas of Cryptography. Diffie and Hellman solved the seemingly impossible problem of key-agreement without a secure channel: namely, they showed that it is possible for a sender and a receiver to agree on a secret key only by communicating via a channel exposed to eavesdropping, if the eavesdropper has limited computing power. Moreover, building upon earlier research by Merkle, Diffie and Hellman proposed the concept of *Public-Key Cryptography;*[1] two notable examples of this concept are Public-Key Encryption and Digital Signatures. In Public-Key Encryption, *two* keys are used, namely one for encryption and one for decryption; the idea is to keep the decryption key private, and to freely disseminate the encryption key, so that any user can encrypt a message, but only the owner of the decryption key can decrypt it. A public and a private key are also used in digital-signature schemes, where only the owner of the private key can produce a valid *signature* for a message, but anyone who knows the public key can check the validity of a signature; thus any user can verify that an acquired message originated from the owner of the private key and was not altered in the transmission.

The work of Diffie and Hellman attracted great interest, and research in Cryptography started to expand to several new areas. A relevant example for this dissertation is the study of more general private-communication scenarios, where sender and receiver exchange messages not by means of a single channel, but over a more complex communication infrastructure, e.g. a number of parallel channels or a communication network; furthermore, enhanced variants of encryption (e.g. *homomorphic encryption*) have been introduced and studied. Other important concepts within Cryptography, on the other hand, are fundamentally different from encryption: this is the case, for instance, of *zero-knowledge protocols,* that allow a *"prover"* to demonstrate to a *"verifier"* that he knows a solution to a certain problem, without revealing any information aside from the fact that he knows that solution. Another important example is *multi-party computation,* where several mutually distrusting parties wish to jointly perform some computation while keeping their inputs private.

From a more general perspective, encryption and its variants aim at achieving *unilateral security,* i.e. their goal is to protect a system or entity against an

[1] Allegedly, Ellis had already developed the idea in 1969, and Cocks designed an early public-key encryption scheme in the early 1970s; however, since both were working with the British intelligence agency GCHQ, their results remained classified until 1997.

external attacker. Zero-knowledge and multi-party computation, on the other hand, are examples of techniques for *multilateral security,* meaning that they aim at protecting the parties involved in the protocol from each other.

Another important distinction between cryptographic techniques is related to the formalization of their security. Many schemes are provably secure under the assumption that a computational problem is intractable; schemes of this type are thus called *computationally secure,* and cannot be broken by attackers with limited computing power (if the assumption they rely on is true). Conversely, schemes whose security does not rely on the assumed hardness of some computational problem are called *unconditionally* or *information-theoretically secure;* these schemes are further divided into *statistically-secure* ones, which allow for a (small) error probability, and *perfectly-secure* ones, which tolerate no error probability.

Cryptography is nowadays an inter-disciplinary field, intersecting mathematics, computer science and engineering. Notably, there has been a fruitful interplay between Cryptography and *Coding Theory,* a large and well-established field that studies how to transmit data efficiently and reliably; in particular, techniques for error-correction have found numerous applications in Cryptography, and are used as building blocks for the new protocols that we discuss in this dissertation.

We now give an overview of the areas of Cryptography that we contribute to.

Secret Sharing. An important topic within modern Cryptography is Secret Sharing, introduced independently by Shamir [58] and Blakley [9] in 1979. Most of the topics that we cover in this dissertation can be seen as generalizations and/or extensions of Secret Sharing.

In its basic form, a secret-sharing scheme takes as input a secret value s, and produces n *shares* via a *sharing* algorithm; the shares are computed in such a way that the secret s can be recovered from any large enough set of shares by means of a *reconstruction* algorithm, while small enough sets of shares yield no information at all on s – i.e., any secret is equally likely to have generated the shares in such a set.

A notable secret-sharing scheme was presented by Shamir in his 1979 seminal article, and is still a widely used ingredient for cryptographic constructions. Shamir's scheme is defined by fixing a threshold $t < n$; for a given secret (which is assumed to lie in a finite field $\mathbb{F}$), the sharing algorithm samples a uniformly random polynomial $P(x) \in \mathbb{F}[x]$ of degree at most t with the property that $P(0)$ is equal to the secret. The shares are then defined as $P(\alpha_1), \ldots, P(\alpha_n)$, for pairwise-distinct and non-zero $\alpha_i \in \mathbb{F}$. Since P is uniquely determined by

$t+1$ or more interpolation points, we have that sets of shares of size $t+1$ or more allow the secret to be reconstructed; moreover, by arguing with Lagrange interpolation theory, it is readily seen that sets of shares of size t or less yield no information at all on the secret.

Secret Sharing has several direct applications. For instance, it can be used for reliable and secure storage: if each share associated to a secret is stored in a separate device, then the secret can be reconstructed even if some of the devices are not available (as long as the remaining ones are in sufficient number), while at the same time an attacker that breaks into a limited number of devices would obtain no information on the secret. Or, a secret-sharing scheme can be used for one-way reliable and secure communication: by sending each of the n shares over a different channel to a receiver, it is guaranteed that the secret can be received even if some of the channels fail to deliver their message, while an attacker eavesdropping on some of the channels would obtain no information at all on the secret.

Furthermore, secret-sharing schemes are used as building blocks in many areas of Cryptography, such as multi-party computation [6, 13, 18], byzantine agreement [56], threshold cryptography [25], access control [54], attribute-based encryption [34, 72], and generalized oblivious transfer [59, 71].

In the vanilla version we discussed, Secret Sharing combines two properties, namely recovery from erasures and privacy. Some "enhanced" variants of this concept have been proposed and studied; in particular, we discuss in this thesis *Robust Secret Sharing,* that combines recovery from *errors* and privacy. More precisely, a secret-sharing scheme is called *robust* if its reconstruction algorithm can recover the secret even if the share string contains errors – i.e., even if some of the shares are incorrect; a small error probability in the reconstruction process is generally allowed.[2] Notice how Robust Secret Sharing has thus direct application to the above "reliable and secure communication setting": if again each share associated to a secret message is sent over a separate channel to a recipient, then the message can be received even if some of the channels deliver an incorrect share, while an attacker eavesdropping on some of the channels would obtain no information on the secret message.

We mention earlier in this introduction that Coding Theory has a record of fruitful interplay with Cryptography; an important example in this interplay involves Secret Sharing. Indeed, a central topic in Coding Theory is error- (and erasure-) correction, namely the study of codes that can recover data from a partially incorrect string of symbols (error correction) or from an incomplete string (erasure correction); thus Coding Theory and Secret Sharing have a

[2] A variation called *Verifiable Secret Sharing* aims at providing security against malicious execution of the sharing phase as well.

similar flavor, as they share this goal of error- or erasure-correction, while Secret Sharing also has a privacy requirement.

In fact, there is a one-to-one correspondence between secret-sharing schemes and error-correcting codes[3], first pointed out by Massey in 1995 [51]; thus any error-correcting code yields a secret-sharing scheme, where the privacy property of the latter can be expressed in terms of the underlying code. Subsequent work on the topic [14] has established new ways of constructing secret-sharing schemes from codes; by using algebraic-geometry codes, this has notably allowed to construct secret-sharing schemes with high ratio between the secret size and the share size.

Secure Message Transmission. Other areas of cryptography spawned as twists on the original cryptographic scenario of private communication; one such example was introduced by Dolev et al. [27] in 1992, and is called *Secure Message Transmission,* shortened to SMT. SMT models a generalization of the "reliable and secure communication" scenario that we discussed in the previous paragraph, where a sender Alice is connected to a receiver Bob by n parallel two-ways channels, and where an adversary Eve controls t of these channels, meaning that she acquires all data that is transmitted over the t channels and that she can overwrite it with data of her choice. The goal of an SMT protocol is to allow Alice to communicate a secret message to Bob with *privacy* and *reliability,* meaning that Eve should acquire no information on the message, while Bob should be able to recover it completely in spite of the errors introduced by Eve; in contrast to (Robust) Secret Sharing, two-ways communication is generally allowed in SMT.

We stress the fact that traditionally, SMT protocols are required to achieve *perfect* privacy, meaning that Eve should obtain no information at all on the message, while reliability can either allow for some small error probability or be perfect. In the latter case, we speak of *Perfectly* Secure Message Transmission or *PSMT.*

Two factors influence whether PSMT is possible and how difficult it is to achieve, namely the number t of channels controlled by Eve, and the number r of transmission rounds, where a transmission round is a phase involving only one-way communication (either from Alice to Bob, or from Bob to Alice).

For $r = 1$, i.e. when communication is only allowed from Alice to Bob, it was showed in Dolev et al.'s original paper [27] that PSMT is possible if and only if $t < n/3$; in fact, one-round SMT is nothing but a special instance of

[3]to be precise, the correspondence is between *linear* secret-sharing schemes and *linear* error-correcting codes. A formal discussion of linearity would be beyond the scope of this introduction; we point out, however, that linearity is an important property for many applications of Secret Sharing.

Robust Secret Sharing, and thus schemes for the latter (e.g. Shamir's scheme together with error correction) can be used to achieve security in this regime of parameters. It is also interesting to compare this setting to the classical scenario of private communication over a *single* channel: the availability of several channels enables to achieve perfect privacy and reliability without any assumption on the computational power of the adversary, and without any need for *a priori* agreement over a secret key.

When interaction is allowed, i.e. $r \geq 2$, it was also shown in [27] that PSMT can tolerate a greater number of corrupted channels, namely it is only required that $t < n/2$, although only an inefficient way to do this was proposed. Subsequently, several works on the topic [57, 1, 68, 46, 45] shared the goal of designing PSMT protocols with improved efficiency, trying to minimize the amount of computation performed by users as well as the amount of required communication; the most recent efforts focus on the hardest regime of parameters, namely allowing only $r = 2$ rounds of communication and setting $n = 2t + 1$.

Secure Network Coding. Techniques from Cryptography and Coding Theory also prove useful in more complex communication problems; a notable example is communication over *networks* that are exposed to external attacks. One can think of a network as a collection of wires and intermediate nodes that connect some source nodes (with no incoming wires) to some sink nodes (with no outbound wires); the source nodes produce some data and send it via the outbound wires, and the sink nodes read the data received via the incoming wires. In Network Routing, the intermediate nodes simply read the data received via the incoming wires and forward it over outbound wires; in contrast to this approach, Network *Coding* allows intermediate nodes to manipulate the received data (namely, perform linear operations on it) and send the result over the outbound wires. The advantage of these operations is that they increase the throughput of the network [2, 47, 43], reaching the best possible value.

From the early 2000's and the work of Cai and Yeung [10], researchers have studied external attacks in this context, first focusing on attackers that eavesdrop on some of the wires of the network [10, 30, 29, 62], and more recently considering the case where attackers can also inject errors on some of the wires [63]. An important point is that Secure Network Coding classically did not study interactive scenarios, and thus assumed that data could only be transmitted from the source nodes (i.e. from the senders) to the sink nodes (i.e., the receivers).

In this dissertation, we show how techniques from Secure Message Transmission, which in turn rely on Coding Theory, can be used to provide security for

network-coding scenarios where two-ways communication is allowed.

Multi-Party Computation. We conclude our survey by discussing a flourishing subfield of Cryptography called *Multi-Party Computation,* first introduced by Yao in 1982 [73] and shortened to MPC.

In MPC, n parties known as *players* hold private inputs $x_1, \ldots, x_n$ respectively and aim at computing the value $f(x_1, \ldots, x_n)$ of a function f on their inputs, while guaranteeing the correctness of the output and while keeping their inputs private.

If we imagine that players can appeal to a trusted mediator, then the problem can be easily solved: players would simply send their inputs to the mediator via secure channels, the mediator would compute the value of $f(x_1, \ldots, x_n)$ and communicate it back to all the players. The goal of an MPC protocol is to allows players to achieve the same outcome without any external mediator.

In MPC, the adversary is internal (and we thus speak of multi-lateral security), meaning that he gains control of some players who are then called *corrupted*. The adversary can be *passive,* meaning that he only reads the data acquired by corrupted parties, or *active,* meaning that he can decide the actions of corrupted players.

Yao's seminal paper [73] introduced the concept of MPC and presented a first two-party protocol; subsequently, several theoretical possibility and impossibility results were presented in the 1980s [32, 6, 13]. Informally stated, these works proved the following crowning result: that if the adversary can only corrupt a certain amount of the players, then every function (with finite domain and finite image) can be securely computed.

The exact number of corrupted players that can be tolerated depends on the capabilities of the adversary (active or passive) and on the required security type. More precisely, n players can compute any function with *computational* security as long as the adversary corrupts less than $n/2$ players, even if the adversary is active. If computational intractability assumptions are not made, then n players can compute any function with *unconditional* security as long as less than $n/2$ players are corrupted by a *passive* adversary; and similarly, n players can unconditionally-securely compute any function as long as less than $n/3$ players are corrupted by an *active* adversary.

The result from [32] is possible thanks to the use of *zero-knowledge,* a cryptographic primitive that has subsequently been widely used to build MPC protocols. Zero-knowledge protocols were first introduced by Goldwasser et al. [33], and, informally stated, allow a party (known as the *prover*) to demonstrate to another party (known as the *verifier*) that he knows a solution to a

certain problem, without revealing any information aside from the fact that he knows that solution. Typically, zero-knowledge protocols are used in MPC to enable (and force) players to prove that they are behaving honestly, i.e. they are following the instructions of the MPC protocol.

Another important step in the development of MPC protocols is the work of Cramer et al. [18], which introduced a paradigm for the construction of MPC protocol from secret-sharing schemes. This type of approach has been very successful, and secret-sharing schemes (or, more precisely, *linear* schemes) are nowadays one of the fundamental building blocks of Multi-Party Computation.

Since the possibility results mentioned above, a great effort has been made in the cryptographic community to design efficient (and thus implementable) MPC protocols. A promising example in this sense is the so-called "SPDZ" protocol by Damgård et al. [23, 22], which will be further discussed in this thesis.

1.2 Thesis Outline and Contributions

The contributions of this thesis to the fields presented above are based on the following publications:

- Ronald Cramer, Ivan Bjerre Damgård, Nico Döttling, Serge Fehr, and Gabriele Spini. Linear secret sharing schemes from error correcting codes and universal hash functions. In *Advances in Cryptology - EUROCRYPT 2015*.
- Gabriele Spini and Gilles Zémor. Perfectly secure message transmission in two rounds. In *Theory of Cryptography - TCC 2016-B*.
- Gabriele Spini and Gilles Zémor. Universally secure network coding with feedback. In *IEEE International Symposium on Information Theory - ISIT 2016*.
- Gabriele Spini and Serge Fehr. Cheater detection in SPDZ multiparty computation. In *Information Theoretic Security - 9th International Conference - ICITS 2016*.

Secret Sharing. Our first contribution to Secret Sharing establishes a new connection between Secret Sharing and Coding Theory. Previous work on this connection [51, 14] described both the secret and the shares in terms of an error-correcting code; this approach has the downside that the privacy

threshold of a secret-sharing scheme has to be estimated in terms of the *dual* of the underlying code. This can be a serious drawback, since to obtain good secret-sharing schemes one needs to design codes with good parameters and good duals, which might not be achievable; for instance, currently known codes with linear-time encoding and decoding have very bad dual codes, so that one cannot directly transpose desirable properties from Coding Theory to their analogues in Secret Sharing.

We circumvent this problem by establishing a new connection between secret-sharing schemes and codes, where the shares are still described in terms of a code, but the secret is now given by a *function* acting on the shares. This allows us to estimate the privacy parameter in terms of the code and the function, thus without analyzing the dual of the code. In this way, we can fully harness the potential of recent code constructions, such as efficient encoding and decoding or efficient list-decoding, to obtain improved schemes.

We present two applications of this connection. The first, more direct one is to construct schemes with linear-time sharing and reconstruction (and with "good" privacy and reconstruction threshold), which was an hitherto open problem. This application uses *universal hash functions,* which provide particularly good bounds for the privacy parameter in our connection; furthermore, linear-time computable universal hash functions have been recently described [28], so that we can combine them with linear-time encodable and decodable codes and obtain the desired schemes with linear-time sharing and reconstruction.

The second application addresses a classical goal of Robust Secret Sharing: in the non-trivial regime of parameters, robustness can be achieved only by allowing for a small error probability in the reconstruction process and by adding extra "check" data to the shares, and several efforts aimed at optimizing the trade-off between the error probability and the share size. Our construction improves on previous work, and in fact reaches the optimal share size for given error probability; furthermore, the sharing and robust-reconstruction of a secret can be efficiently performed, namely in time polynomial in the secret size and the number n of shares.

Our construction builds upon the above connection between secret-sharing schemes and codes, and makes use of two main ingredients. The first one consists of codes with good list-decodability properties, where loosely speaking, list decoding is a form of error correction where on input an incorrect string one tries to recover a list of possible original strings, and not a single one; the second component consists of *AMD* codes, which can detect certain types of manipulation on data. Intuitively, robust reconstruction is achieved by first list-decoding the share string to obtain a list of possible secrets, and then use the AMD code to detect and prune out incorrect secrets; by choosing recent

construction of list-decodable codes and AMD codes with good properties, we are able design two families of efficient robust-secret sharing schemes with optimal share size.

Perfectly Secure Message Transmission and Secure Network Coding. We design a protocol for Perfectly Secure Message Transmission that improves on previous work in two main aspects: first, it follows a conceptually simpler blueprint, making use of relatively straightforward coding-theoretic techniques; and second, it achieves better complexity.

To be more precise, the protocol we introduce has a communication complexity of $O\left(n^2 \log n\right)$ and a transmission rate of $O(n)$, where these two quantities are defined as the total number of bits transmitted to communicate a single-bit secret, and the ratio between the total number of bits transmitted and the bit-size of the secret (for a secret of growing size), respectively. We thus improve over the previous state-of-the-art protocol, due to Kurosawa and Suzuki [46, 45], which achieved a communication complexity of $O\left(n^3 \log n\right)$ bits and a transmission rate of $O(n)$; the latter value is optimal.

Just as importantly, our solution is conceptually simpler than previous protocols. Our core idea combines two main components, namely *"syndrome-decoding"* techniques from Coding Theory and some tools developed in previous work to identify the channels corrupted by Eve. Together, these two components form a straightforward structure, in contrast to previous work on PSMT which relied on early strategies that marked substantial progress at the time but are now surpassed by more effective techniques. It is also worth remarking that its simple core structure allows our protocol to be adapted to a Secure Network Coding scenario, as we will discuss in the following lines.

Classically, only one-way communication (from source to sink nodes) has been studied for Secure Network Coding; in this setting, security can be guaranteed as long as the number of wires controlled by the attacker is less than one third of the network connectivity, defined as the minimum number of wires whose removal disconnect the source node from any sink node. We contribute to the topic by presenting protocols where data can also be conveyed from sink nodes to source nodes; with this feature, security is guaranteed against a stronger adversary, namely the number of corrupted wires only needs to be smaller than one half of the connectivity.

We adapt our techniques for PSMT to this setting; this is possible thanks to the similarity between PSMT and Secure Network Coding. A full discussion of this similarity would be beyond the scope of this introduction, but we give here some intuition: focus for simplicity on a Secure Network Coding scenario with a single source node (that we imagine controlled by Alice) and a single

sink node (that we imagine controlled by Bob). If we then only look at the input/output behaviour, this scenario can be seen as a generalization of PSMT where the adversary Eve controls a "linear combination" of channels, instead of a subset of channels, in the following sense. Recall that in the presence of network coding, intermediate nodes in a network send linear combinations of the received data over the outbound wires. This means that wires always carry linear combinations of the input symbols, so that in particular the data eavesdropped by the attacker Eve has this "linear-combination" form; similarly, the errors injected by Eve on the wires under her control propagate in the same fashion through the network, so that Bob's received data will be affected by linear combination of the errors.

The techniques we introduced for PSMT can be easily adapted to this scenario with enhanced capabilities of the adversary. More precisely, we first show a direct adaptation of our PSMT protocol to the single-sender, single-receiver network coding scenario, obtaining a protocol which is secure for a greater number of corrupted wires compared to previous work, namely going from a third of the connectivity to a half. Furthermore, we introduce a three-round protocol, based on the same core ideas, that also works in a multi-cast scenario, i.e. with several receivers or "Bobs"; once again, two-ways communication allows us to achieve greater security compared to previous work, from a third to a half of the connectivity.

Secure Multi-Party Computation. Our work on MPC concerns the so-called SPDZ protocol (named after the initials of its authors) for Multi-Party Computation [23, 22], a recent scheme with very good efficiency and thus promising possibilities of becoming practical.

A drawback of the SPDZ scheme is that it does not provide any form of dishonest player identification: as soon as cheating is detected, the protocol simply aborts, leaving the honest players clueless about the identity of the cheater. Thus an adversary can force the scheme to abort without having to fear any consequences; this means that SPDZ is vulnerable to a denial-of-service attack.

We contribute to the topic by providing SPDZ with a form of cheater detection, without significantly increasing the running time of the scheme; the efficiency of SPDZ is in fact one of its most important features.

We present an "improved version" of SPDZ with identifiable abort; more precisely, we achieve the following. In case no cheating takes place, the protocol computes the right output value and has a complexity of the same magnitude as the original SPDZ. In case cheating does take place, several situations may occur, depending on the exact behaviour of the adversary. A first possible

outcome is that the protocol is able to handle the cheating, i.e., it successfully computes the output value; in this case, the protocol is slower by a factor 2 only compared to the original SPDZ. The other possible outcome is that the protocol aborts, but identifies a dishonest player; here we distinguish two further cases: *identification without agreement,* where each honest player identifies a dishonest one, but there may be no consensus on the identity of the dishonest players; and *identification with agreement,* where all honest players agree on the identity of at least a dishonest player. In the no-agreement case, our protocol is slower by a factor 2 only, while in the in-agreement case, the protocol may be significantly slower, but since a dishonest player will be publicly exposed, there is little incentive for the adversary to enforce this scenario.

Chapter 2

Preliminaries

This chapter presents the fundamental building blocks upon which this thesis is built. We introduce basic notation in Section 2.1; Section 2.2 gives an overview of Coding Theory, a central building block for this dissertation. In Section 2.3, we give a brief overview of probability theory, adapting the discussion to our needs; in Section 2.4, we discuss how to model algorithms and protocols. Finally, Section 2.5 discusses a key subject for this dissertation, Secret Sharing, while Section 2.6 recalls the connection between Secret Sharing and Coding Theory.

2.1 General Notation

We denote by $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ the set of non-negative integers, relative integers and real numbers, respectively; given two real number $a < b$, we denote by $[a, b]$ the closed interval $\{x \in \mathbb{R} : a \leq x \leq b\}$; A finite field with unspecified cardinality is denoted by $\mathbb{F}$; given a prime power q, we denote a finite field with q elements by $\mathbb{F}_q$.

Vectors are written in boldface; in expressions involving vectors and matrices, vectors are considered to be row vectors unless otherwise stated. The logarithm function log is assumed to be in base 2.

Given sets $\mathcal{S}_1, \ldots, \mathcal{S}_n$ and a subset of coordinates $\mathcal{A} \subseteq [n]$, we denote by $\mathcal{S}^{\mathcal{A}}$ the set $\{(s_i)_{i\in\mathcal{A}} : s_i \in \mathcal{S}_i\}$. Furthermore, elements $(s_i)_{i\in\mathcal{A}}$ are simply denoted by $s_{\mathcal{A}}$, and $\pi_{\mathcal{A}} : \mathcal{S}_1 \times \cdots \times \mathcal{S}_n \to \mathcal{S}^{\mathcal{A}}$ denotes the projection map $(\mathbf{s}_1, \ldots \mathbf{s}_n) \mapsto \mathbf{s}_{\mathcal{A}}$.

2.2 Error-Correcting Codes

Error-correcting codes are mathematical objects that are used to protect data from erasures or errors. We focus on *block* codes, which will be a fundamental building block for several parts of this dissertation; with a slight abuse of notation, we will refer to these codes simply as error-correcting codes, highlighting their purpose. The reader can refer to [49] for a detailed study of error correction.

2.2.1 Basic Definitions and Properties

Let F be a non-empty set, and $n > 0$ be an integer; given two elements $\mathbf{v}, \mathbf{v}' \in F^n$, we define their *Hamming distance* to be $d_{\mathrm{H}}(\mathbf{v}, \mathbf{v}') := |\{i : \mathbf{v}_i \neq \mathbf{v}'_i\}|$; if F has an abelian group structure (in additive notation), we can also define the *Hamming weight* of an element $\mathbf{v} \in F^n$ to be $w_{\mathrm{H}}(\mathbf{v}) := |\{i : \mathbf{v}_i \neq 0\}|$. Notice that in this case, $d_{\mathrm{H}}(\mathbf{v}, \mathbf{v}') = w_{\mathrm{H}}(\mathbf{v} - \mathbf{v}') = w_{\mathrm{H}}(\mathbf{v}' - \mathbf{v})$.

Definition 2.2.1 (**Error-correcting Code**). *Let F be a finite set, and let $n > 0$ be an integer; an* error-correcting code *of length n over F is a non-empty subset $\mathcal{C} \subseteq F^n$. F is called the* alphabet *of $\mathcal{C}$, n is called its* length*, and elements of $\mathcal{C}$ are called* codewords*; we will frequently call $\mathcal{C}$ simply a* code*, if no confusion can arise.*

The minimum distance *of $\mathcal{C}$ is $d_{\min}(\mathcal{C}) := \min\{d_{\mathrm{H}}(\mathbf{x}, \mathbf{x}') : \mathbf{x} \neq \mathbf{x}' \in \mathcal{C}\}$; for completeness, if $|\mathcal{C}| = 1$ we set $d_{\min}(\mathcal{C}) := n + 1$.*

The minimum distance of a code determines how may erasures or errors the code can correct, as we show in the following remark.

Remark 2.2.1. Let $\mathcal{C}$ be a code of length n and minimum distance d with alphabet F; we then have that $\mathcal{C}$ can correct $d - 1$ erasures and $\lfloor (d-1)/2 \rfloor$ errors, in the following sense.

Let $\mathbf{x} \in \mathcal{C}$ be an arbitrary codeword; let $\mathbf{y} \in (F \sqcup \{\perp\})^n$ be an n-tuple obtained from $\mathbf{x}$ by replacing up to $d - 1$ of its symbols with erasure marks $\perp$. Then $\mathbf{x}$ is uniquely determined by $\mathbf{y}$, namely it is the only element in the set $\{\mathbf{x}' \in \mathcal{C} : d_{\mathrm{H}}(\mathbf{x}', \mathbf{y}) \leq d - 1\}$. In other words, $n - d + 1$ symbols of any codeword of $\mathcal{C}$ uniquely determine it.

For what concerns error correction, let $\mathbf{x} \in \mathcal{C}$ be an arbitrary codeword and let $\mathbf{y} \in F^n$ be an n-tuple such that $d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) \leq \lfloor (d-1)/2 \rfloor$. Then again, $\mathbf{x}$ is uniquely determined by $\mathbf{y}$, namely it is the only element in the set $\{\mathbf{x}' \in \mathcal{C} : d_{\mathrm{H}}(\mathbf{x}', \mathbf{y}) \leq \lfloor (d-1)/2 \rfloor\}$.

Definition 2.2.2. *Let $\mathcal{C}$ be a code; if its alphabet F is equal to a finite field $\mathbb{F}_q$ and $\mathcal{C}$ is an $\mathbb{F}_q$-subspace of $\mathbb{F}_q^n$, we say that $\mathcal{C}$ is an* ($\mathbb{F}_q$-) linear *code.*

We then combine three key parameters of $\mathcal{C}$ into a triplet $[n, k, d]$, where

- *n is, as stated, the* length *of the code;*
- *k is the* dimension *of $\mathcal{C}$ as an $\mathbb{F}_q$-linear space;*
- *d is the* minimum distance *of $\mathcal{C}$; notice that $d = \min\{w_{\mathrm{H}}(\mathbf{x}) : \mathbf{x} \in \mathcal{C} \setminus \{\mathbf{0}\}\}$ (assuming $|C| \neq \{0\}$).*

We sometimes use a variant of this notation and write $[n, k, d]_q$, in order to have the alphabet size as well in a unique expression. The rate *of the code is defined to be $R := k/n$.*

Definition 2.2.3 (**Parity-Check Matrix**)**.** *Let $\mathcal{C}$ be a linear code of parameters $[n, k, d]_q$. A* parity-check matrix *of $\mathcal{C}$ is any matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ with the property that*

$$\mathcal{C} = \left\{\mathbf{x} \in \mathbb{F}_q^n : \mathbf{H}\mathbf{x}^T = \mathbf{0}\right\}.$$

It is immediately seen that a parity-check matrix exists for any linear code; furthermore, given a parity-check matrix $\mathbf{H}$ for $\mathcal{C}$, we define its associated syndrome map *$\sigma : \mathbb{F}_q^n \to \mathbb{F}_q^{(n-k)}$ where $\sigma(\mathbf{v}) := \mathbf{H}\mathbf{v}^T$. Thus by definition, $\mathcal{C} = \ker\sigma$.*

Let us now discuss in some more detail the properties of erasure/error correction of codes. Let $\mathcal{C}$ be a code; for simplicity, assume it is linear, with parameters $[n, k, d]_q$. As a direct consequence of Remark 2.2.1, there exists a function $\texttt{Decode} : (\mathbb{F}_q \sqcup \{\perp\})^n \to (\mathcal{C} \sqcup \{\perp\})$ that can correct from $d-1$ erasures and $\lfloor (d-1)/2 \rfloor$ errors; this means that given any $\mathbf{x} \in \mathcal{C}$, if $\mathbf{y} \in (F \sqcup \{\perp\})^n$ is obtained from $\mathbf{x}$ by replacing up to $d-1$ of its symbols with erasures marks $\perp$, then $\texttt{Decode}(\mathbf{y}) = \mathbf{x}$, and similarly for errors.

Such a function `Decode` is usually called a decoding *algorithm.* An important remark is that it is non-trivial to devise efficient decoding algorithms, namely given an arbitrary code $\mathcal{C}$ there is no general method to compute $\texttt{Decode}(\mathbf{y})$ for an arbitrary $\mathbf{y}$ *efficiently* (i.e., in time polynomial in the block length – see Section 2.4 for a discussion over complexity); it is thus a classical goal of coding theory to design codes with *efficient* associated decoding algorithm.

Furthermore, given a linear code $\mathcal{C}$ of dimension k over $\mathbb{F}_q$, we have that there exists an isomorphism $\texttt{Encode} : \mathbb{F}_q^k \to \mathcal{C}$; `Encode` is then said to be an *encoding*

algorithm. We stress the fact that the name "decoding function" or "decoding algorithm" sometimes denotes $\texttt{Encode}^{-1} \circ \texttt{Decode}$; it will always be clear in this dissertation if we are referring to this function or simply to `Decode`.

We now recall the concept of dual code, which will be used to describe the connection between codes and Secret Sharing.

Definition 2.2.4. *Let $\mathcal{C}$ be a linear code of parameters $[n, k, d]$ with alphabet $\mathbb{F}$. The* dual code *of $\mathcal{C}$ is the linear code of length n given by $\mathcal{C}^{\perp} := \{\mathbf{y} \in \mathbb{F}^n : \mathbf{y}\mathbf{x}^T = 0 \quad \forall\, \mathbf{x} \in \mathcal{C}\}$.*

The parameters of $\mathcal{C}^{\perp}$ are sometimes denoted by $[n, k^{\perp}, d^{\perp}]$; it can be proved that $k + k^{\perp} = n$.

We conclude this first part of the discussion on codes with the following definition:

Definition 2.2.5 (**Folded Codes**). *Let $\mathbb{F}_q$ be a finite field and $m > 0$ be an integer; then a code $\mathcal{C}$ with alphabet $\mathbb{F}_q^m$ is said to be* m-folded *if it is an $\mathbb{F}_q$-linear space. m is then called the* folding parameter *of $\mathcal{C}$.*

We associate to a folded code $\mathcal{C}$ the parameters $[n, k, d]_{q^m}$, where n and d are its block length and minimum distance, and k (unless otherwise specified) is its dimension over $\mathbb{F}_q$. The rate *of the code is defined to be $R := k/mn$.*

2.2.2 MDS Codes

We discuss the class of linear codes with maximum possible dimension and distance; the errors and erasures we focus on in this section are *adversarial,* meaning that we do not assume that they are sampled according to some probability distribution, rather we just suppose that they have bounded weight. Recall that as seen in the previous subsection, the amount of erasures or errors of this type that a code can correct are directly proportional to its minimum distance; it is thus desirable to design codes with distance as large as possible.

On the other hand, it is also desirable to design codes that contain many words, i.e., that have large dimension – a code that corrects from many errors but contain only a single word is of little use.

We recall the following fundamental fact, that establishes a trade-off between the minimum distance and the size of a code:

Theorem 2.2.1 (**Singleton Bound**). *Let $\mathcal{C}$ be a code of length n over the alphabet F; then it holds that $d_{\min}(\mathcal{C}) + \log_{|F|} |\mathcal{C}| \leq n + 1$.*

Proof. Let $d := d_{\min}(\mathcal{C})$; notice that $d \leq n$ unless $|\mathcal{C}| = 1$, in which case the statement holds. Thus assume that $d \leq n$, and define $\mathcal{C}'$ to be the punctured code obtained from $\mathcal{C}$ by removing its first $d-1$ symbols, i.e.

$$\mathcal{C}' := \left\{(\mathbf{x}_d, \ldots, \mathbf{x}_n) \in F^{n-d+1} \,:\, \exists\, \mathbf{x}_1, \ldots, \mathbf{x}_{d-1} \text{ s.t. } (\mathbf{x}_1, \ldots, \mathbf{x}_{d-1}, \mathbf{x}_d, \ldots, \mathbf{x}_n) \in \mathcal{C}\right\}.$$

By definition of minimum distance, we have that $|\mathcal{C}'| = |\mathcal{C}|$. We thus have the following inequalities:

$$|\mathcal{C}| = |\mathcal{C}'| \leq \left|F^{n-d+1}\right| = |F|^{n-d+1}.$$

The claim follows.

□

Definition 2.2.6 (**MDS Codes**). *A code $\mathcal{C}$ is said to be* Maximum Distance Separable *or* MDS *if it meets the Singleton bound.*

For simplicity, we will only consider linear *MDS codes in this dissertation; hence a linear code $\mathcal{C}$ of parameters $[n, k, d]$ is MDS if $d + k = n + 1$.*

We recall the following important result for MDS codes:

Proposition 2.2.2. *Let $\mathcal{C}$ be an MDS code of parameters $[n, k, n-k+1]_q$; let $\mathcal{B} \subseteq [n]$ be a subset with $|\mathcal{B}| = k$. Then $\mathcal{B}$ is an* information set *for $\mathcal{C}$, i.e. the projection map $\pi_{\mathcal{B}} : \mathcal{C} \to \mathbb{F}_q^{\mathcal{B}}$ is one-to-one.*

An important example of MDS code is given by the family of Reed-Solomon codes, defined as follows:

Definition 2.2.7 (**Reed-Solomon Codes**). *Let $\mathbb{F}_q$ be a finite field and n a positive integer with $n \leq q$; a* Reed-Solomon *code RS of length n over $\mathbb{F}_q$ is defined as follows.*

Fix a non-negative integer $t \leq n$ and n elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ with $\alpha_i \neq \alpha_j$ for any $i \neq j$ (notice that this is possible since $n \leq q$); then let

$$RS := \{(P(\alpha_1), \ldots, P(\alpha_n)) \,:\, P(x) \in \mathbb{F}_q[x], \deg P \leq t\}.$$

It is easily seen that RS is MDS with parameters $[n, t+1, n-t]$ (simply observe that the number of roots of a non-zero polynomial cannot exceed its degree).

An important property of Reed-Solomon codes is that they are decodable in time polynomial in n and q.

Also notice that the construction of Reed-Solomon codes implies that if $q \geq n$, then MDS codes of parameters $[n, k, n-k+1]_q$ exist for any k.

2.3 Probability Theory

2.3.1 Modeling Non-Determinism

When using mathematics to analyze real-life scenarios, one may be confronted with the task of modeling a statement which is inherently unpredictable, or in other words, non-deterministic: such a statement cannot be described by a "standard" mathematical statement, which is either true or false. Think of the toy example of rolling a die: a statement such as "the outcome is an even number" is not surely true or surely false, but rather can be either of them with a certain probability; similarly, the outcome of the die roll is not a fixed value, but may be one of several possible values that occur with certain probabilities.

Probability theory is the branch of mathematics that models such non-deterministic statements and values. It hence plays an important role in this dissertation, where non-determinism is extensively used to provide security in a cryptographic setting.

The standard formulation of probability theory is due to Kolmogorov. With this approach, non-determinism is modeled by selecting a suitable probability space that represents the source of randomness; non-deterministic statements and values are then modeled in terms of the probability space.

We will adopt a more abstract point of view on probability theory, inspired by the lecture notes of Terence Tao for his graduate course at UCLA [**?**]. With this approach, a non-deterministic statement or value is modeled by a more abstract object; in order to study the properties of this object, one can then select a suitable *representation* for it, expressed in Kolmogorov's formulation, and refer to the formalism of standard probability theory.

In fact, this approach is used very often in cryptographic publications, although it is only implicit; our aim is thus to give a firm mathematical ground to this widely used convention.

The advantages of this abstract approach are particularly evident in case there is no canonical representation for an abstract object, since the flexibility in selecting and changing representations becomes very useful. Just as importantly, defining the probabilistic models that we need becomes faster and more intuitive with the abstract approach. Indeed, defining the probability space to model a non-deterministic statement can be quite arbitrary and cumbersome, even if the high-level description of the statement is very intuitive. In this sense, our abstract approach does not introduce any unnecessary complexity when modeling non-determinism.

2.3.2 Kolmogorov's Probability Theory

We present here some basic notions of probability theory in the classical formulation due to Kolmogorov; we restrict ourselves to *finite* probability theory, as it is sufficient to cover the topics of this dissertation.

Definition 2.3.1. *A* (finite) probability space *consists of a pair* (Ω, p)*, where* Ω *is a finite, non-empty set called* sample space*, and* $p : \Omega \to [0, 1]$ *is a function called* probability measure *such that*

$$\sum_{\omega \in \Omega} p(\omega) = 1.$$

An event *is a subset* E *of the probability space* Ω*; we extend the probability measure* p *to events by setting*

$$p(E) := \sum_{\omega \in E} p(\omega),$$

where by convention, $p(\emptyset) := 0$*.*

In Kolmogorov's formulation, an event is used to model a non-deterministic statement; notice that standard (i.e., deterministic) statements can be seen as events with probability 1 if true, or 0 if false.

Similarly, non-deterministic values are modeled by random variables, defined as follows.

Definition 2.3.2. *Let* (Ω, p) *be a probability space; a* random variable *over* (Ω, p) *is defined to be a function* $X : \Omega \to \mathcal{X}$*, where we may assume* $\mathcal{X}$ *to be finite. With a slight abuse of notation, we will sometimes write* $X : (\Omega, p) \to \mathcal{X}$*.*

Notice that any deterministic value $x \in \mathcal{X}$ can be seen as the random variable $(\Omega, p) \to \mathcal{X}$, $\omega \mapsto x$.

We adopt the following convention. For $x \in \mathcal{X}$, we denote by $X = x$ the event $\{\omega \in \Omega : X(\omega) = x\}$; similarly, given a subset $\mathcal{A} \in \mathcal{X}$, we write $X \in \mathcal{A}$ to denote the event $\{\omega \in \Omega : X(\omega) \in \mathcal{A}\}$, and for two random variables $X : (\Omega, p) \to \mathcal{X}$ and $Y : (\Omega, p) \to \mathcal{Y}$, we write $X = Y$ to denote the event $\{\omega \in \Omega : X(\omega) = Y(\omega)\}$, etc.

We recall the following important concepts:

Definition 2.3.3. *Let* (Ω, p) *be a probability space, and let* E *and* E' *be events of* Ω*; we usually denote* $p(E \cap E')$ *by* $p(E, E')$*. If* $p(E') > 0$*, we define the* conditional *probability of* E *given* E' *to be*

$$p(E|E') := \frac{p(E, E')}{p(E')}.$$

Definition 2.3.4. *Given a probability space (Ω, p) and two events $E, E' \subseteq \Omega$, we say that E and E' are* independent *if $p(E, E') = p(E) \cdot p(E')$. Notice that in this case, if $p(E') > 0$ then $p(E|E') = p(E)$.*

Given two random variables $X : (\Omega, p) \to \mathcal{X}$ and $Y : (\Omega, p) \to \mathcal{Y}$, we say that X and Y are independent *if the events $X = x$ and $Y = y$ are independent for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. In this case, $p(X = x|Y = y) = p(X = x)$.*

Definition 2.3.5. *Let $X : (\Omega, p) \to \mathcal{X}$ be a random variable. The (probability)* distribution *of X is defined to be the function $D_X : \mathcal{X} \to [0, 1]$ given by*

$$D_X(x) := p(X = x).$$

We say that a random variable X is uniformly distributed *over $\mathcal{X}$ if $D_X(x) = 1/|\mathcal{X}|$ for any $x \in \mathcal{X}$.*

Let X and Y be two random variables defined over the same probability space (Ω, p), having range $\mathcal{X}$ and $\mathcal{Y}$ respectively. Then their cartesian product $XY : (\Omega, p) \to \mathcal{X} \times \mathcal{Y}, \omega \mapsto (X(\omega), Y(\omega))$ is a well-defined random variable; its distribution $D_{XY} : (x, y) \mapsto p(X = x, Y = y)$ is called the *joint* distribution of X and Y. D_X and D_Y are called *marginal* distributions in this setting; notice that marginal distributions can be obtained from the joint one, e.g. $D_X(x) = \sum_y D_{XY}(x, y)$.

These concepts and notations naturally extend to the case of several random variables.

Remark 2.3.1. Given a probability space (Ω, p), we have a one-to-one correspondence between events $E \subseteq \Omega$ and boolean random variables $X_E : \Omega \to \{0, 1\}$: simply define $X_E(\omega) = 1$ if $\omega \in E$, and 0 otherwise. This means that we can view events as a special type of random variables (namely, those with range equal to $\{0, 1\}$).

2.3.3 Abstract Probability Theory

We describe in this section a more abstract variant of probability theory; as discussed in Section 2.3.1, this has the advantage of not relying on a fixed probability space, while at the same time the properties of Kolmogorov's formalism can be applied by selecting a suitable representation.

Given a finite set $\mathcal{X}$, we would like to think of a non-deterministic value in $\mathcal{X}$ as an abstract object defined only in terms of its distribution. To this end,

consider the set of triplets (Ω, p, X) where (Ω, p) is a probability space and $X : (\Omega, p) \to \mathcal{X}$ is a random variable; we then have that the binary relation given by $(\Omega, p, X) \sim (\Omega', p', X')$ if $D_X = D_{X'}$ is an equivalence relation.

Definition 2.3.6. *Given a finite, non-empty set $\mathcal{X}$, we denote by $\mathcal{RV}(\mathcal{X})$ the set of equivalence classes under the relation $\sim$ described above; an* abstract random variable *over $\mathcal{X}$ is then an equivalence class $x \in \mathcal{RV}(\mathcal{X})$. We write $x \in_\$ \mathcal{X}$.*

A representation *of $x \in \mathcal{RV}(\mathcal{X})$ is then simply an element (Ω, p, X) of the equivalence class x. The* distribution *D_x of x is defined to be the distribution $D_X : \mathcal{X} \to [0, 1]$ of any representation (Ω, p, X) of x.*

Hence we have that an abstract random variable x over $\mathcal{X}$ is uniquely determined by its distribution D_x; conversely, it can be easily seen that any function $D : \mathcal{X} \to [0, 1]$ with $\sum_x D(x) = 1$ is the distribution of an abstract random variable over $\mathcal{X}$. This means that we can view an abstract random variable as an *abstract object* entirely defined by its distribution.

This implies that properties of random variables (in the Kolmogorov sense) that are determined by their distributions are naturally inherited by abstract random variables. For instance, an abstract random variable $x \in \mathcal{RV}(\mathcal{X})$ is naturally defined to be uniformly distributed if $D_x(\hat{x}) = 1/|\mathcal{X}|$ for any $\hat{x} \in \mathcal{X}$; or, given $x \in_\$ \mathcal{X}$ and a representation (Ω, p, X) of x, expressions like $p(X = \hat{x})$, $p(X \in \mathcal{A})$, etc. are well-defined and independent of the choice of X. Hence we will typically write $p(x = \hat{x})$, $p(x \in \mathcal{A})$, etc. instead, implicitly assuming that a suitable representation (Ω, p, X) has been selected; this also means that the letter p appearing in these formulas does not necessarily refer to a specific probability measure.

Remark 2.3.2. In line with Remark 2.3.1, non-deterministic statements can be modeled as abstract random variables over $\{0, 1\}$.

These notions extend to the case of two or more random variables: given two non-empty finite sets $\mathcal{X}, \mathcal{Y}$, an *abstract pair* of random variables over $\mathcal{X} \times \mathcal{Y}$ is naturally defined as an element $(x, y) \in \mathcal{RV}(\mathcal{X} \times \mathcal{Y})$. A *representation* of (x, y) is by definition an element of the equivalence class; notice that we can write it as a triplet $\big(\Omega, p, (X, Y)\big)$ where (Ω, p) is a probability space and $X : (\Omega, p) \to \mathcal{X}$ and $Y : (\Omega, p) \to \mathcal{Y}$ are two random variables.

Again, we obtain a one-to-one correspondence between abstract pairs of random variables over $\mathcal{X} \times \mathcal{Y}$ and joint distributions $D : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ by selecting a representation $\big(\Omega, p, (X, Y)\big)$ of any $(x, y) \in \mathcal{RV}(\mathcal{X} \times \mathcal{Y})$ and assigning $(x, y) \mapsto D_{XY}$. Notice that the concept of independence is well-defined for abstract random variables.

Remark 2.3.3. We stress the fact that there is a difference between an abstract

pair of random variables $(x, y) \in_\$ \mathcal{X} \times \mathcal{Y}$ and a pair of abstract random variables $x \in_\$ \mathcal{X}$, $y \in_\$ \mathcal{Y}$: in the latter case only the marginal distributions D_x and D_y are specified, and no joint distribution $D_{(x,y)}$ is a priori defined. In the latter case, we endow (x, y) with the so-called *product distribution*, namely we consider the abstract pair of random variables $(x, y) \in_\$ \mathcal{X} \times \mathcal{Y}$ defined by $D_{(x,y)}(\hat{x}, \hat{y}) = D_x(\hat{x}) \cdot D_y(\hat{y})$ for any $\hat{x} \in \mathcal{X}$, $\hat{y} \in \mathcal{Y}$.

Let $x \in \mathcal{RV}(\mathcal{X})$ be an abstract random variable, and let $f : \mathcal{X} \to \mathcal{Y}$ be a function, where we assume the set $\mathcal{Y}$ to be finite. The abstract random variable $f(x)$ over $\mathcal{Y}$ is then naturally defined to be the equivalence class $(\Omega, p, f \circ X)$, where (Ω, p, X) is an arbitrary representation of x. Notice that the distribution of $f(x)$ is given by

$$D_{f(x)}(\hat{y}) = \sum_{\hat{x}: f(\hat{x}) = \hat{y}} D_x(\hat{x}).$$

2.4 Modeling Algorithms and Protocols

The standard approach in Cryptography (and other areas of applied mathematics) to formalize algorithms and protocols is to use *Turing machines* and variants thereof. Informally speaking, a Turing machine is an abstract machine that takes some input values, then manipulates symbols on an infinite tape according to a set of instructions, and produces output values; in particular, Turing machines let formally define the complexity of an algorithm.

We choose, however, a different approach, given that we are only interested in the input/output behaviour of algorithms and protocols. More specifically, we focus on *information-theoretic* security properties, i.e. which do not require attackers to have limited computing power; complexity is thus not necessary to formalize security (although we will take into account the efficiency of algorithms). Under these considerations, using Turing machines would add extra complexity without a real need for it.

We thus use a different approach, describing algorithms and protocols in terms of *(randomized) functions*. We first focus on non-interactive algorithms - i.e., algorithms that are executed by a single party. These are formalized as follows.

Let $\mathcal{X}$ and $\mathcal{Y}$ be two finite, non-empty sets; a *deterministic algorithm* with input space $\mathcal{X}$ and output space $\mathcal{Y}$ is a *function* $\texttt{Alg} : \mathcal{X} \to \mathcal{Y}$. Similarly, a *randomized algorithm* $\texttt{Alg}$ with input space $\mathcal{X}$ and output space $\mathcal{Y}$ is a *random variable* over the set of functions $\{f : \mathcal{X} \to \mathcal{Y}\}$.

We usually write $\texttt{Alg} : \mathcal{X} \to_\$ \mathcal{Y}$ instead of $\texttt{Alg} \in_\$ \{f : \mathcal{X} \to \mathcal{Y}\}$; we will

sometimes be flexible with the notation and simply speak of *randomized functions,* if algorithmic aspects are not important. Moreover, we often describe a randomized algorithm $\mathtt{Alg} : \mathcal{X} \to_{\$} \mathcal{Y}$ by simply defining the random variables $\mathtt{Alg}(x) \in_{\$} \mathcal{Y}$ for any $x \in \mathcal{X}$; $\mathtt{Alg}$ is then implicitly assumed to be given by the product distribution.

Just as for arbitrary random variables, unless otherwise specified or clear from the context we assume that any two randomized algorithm $\mathtt{Alg}$, $\mathtt{Alg}'$ are independent as a pair of random variables.

We now give an informal overview on interactive algorithms and protocols. Fix an integer $n > 0$, denoting the number of interacting parties. As a first step, an *n-party interactive algorithm* is defined to be an algorithm that exchanges messages with $n-1$ other parties at given steps of the computation. A *protocol* is then defined to be a collection of n interactive algorithms; the *execution* of a protocol is the function obtained by "connecting" the n interactive algorithms in the intuitive way. We will frequently describe protocols in "concrete" terms, e.g. by using expressions like "party i receives x_j from each party j, then communicates the value of $f(x_1, \ldots, x_n)$ to all the parties"; it will always be clear how these statements can be formalized in terms of interactive algorithms and protocols as described above.

Finally, notice that in order to formalize the security of a protocol, one needs to model the behaviour of an attacker: for multi-lateral security (cf. Chapter 1), this is done by replacing the interactive algorithm $\mathtt{Alg}_i$ of any corrupted party with another arbitrary algorithm $\mathtt{Alg}'_i$.

2.4.1 Complexity

As stated at the beginning of this section, complexity is not the main focus of this dissertation; nevertheless, we do take into account the efficiency of the algorithms and protocols that we design. We thus discuss in this section how to analyze complexity, although we will not be completely formal.

Whenever we speak of the *computational complexity* of a (possibly randomized) algorithm $\mathtt{Alg}$, we assume that a suitable computation model has been fixed; statements on complexity will then be meaningful in this model. For instance, when we speak of linear-time or polynomial-time algorithm (in a given parameter), it is understood that these are meant in the Turing-machine sense.

When making precise statement about complexity parameters, we generally use a different computation model, known as the *arithmetic-circuit model:* we

fix a finite field $\mathbb{F}$ (how to choose it will be clear from the context), then decompose the algorithm `Alg` in a number of elementary operations (e.g. additions and multiplications) over $\mathbb{F}$; the complexity of `Alg` is then defined to be the number of these elementary operations. In both cases, when measuring the complexity of a randomized algorithm `Alg`, we always implicitly refer to its worst-case complexity.

Finally, we focus on protocols. Since the execution of a protocol is an algorithm, its (computational) complexity is well-defined; again, we will often be flexible with the terminology and speak of the complexity of the protocol itself.

Also notice in this context, there are other important factors to measure the efficiency of a protocol, like the number of rounds and the amount of data that parties need to communicate to each other (known as *communication complexity*). The precise way to measure the communication complexity, however, changes slightly for different fields, so we refer to the corresponding chapters for a more accurate definition.

2.5 Secret Sharing

Secret Sharing was introduced independently by Shamir [58] and Blakley [9] in 1979. In its basic form, a secret-sharing scheme takes as input an element s, called the *secret,* and produces as outputs n elements called *shares* via a sharing algorithm; the goal of the scheme is to ensure that some "privileged" subsets of the shares allow s to be recovered by means of a reconstruction algorithm, while other "rejected" subsets should yield no information at all on s – i.e., any secret should equally likely to have generated the shares in a rejected set. We focus here on the case where "privileged" sets are large enough sets, and "rejected" sets are small enough sets.

In [58], the following direct application of Secret Sharing to a real-world scenario was suggested: in order to store securely and reliably a secret value, a secret-sharing scheme is used to produce associated shares, where each share is subsequently stored in a separate device. Thus the secret value can be later reconstructed even if some of the devices are no longer available (as long as the remaining ones are in sufficient number, i.e. form a "privileged" set), while at the same time an attacker that breaks into a limited number of devices would obtain no information at all on the secret (since he would only collect shares in a "rejected" set).

Another possible application is to one-way secure and reliable communication of a secret message: assume that a sender can communicate with a receiver via several distinct channels; then to communicate a secret message, the sender

can use a secret-sharing scheme to produce associated shares, where each share is subsequently sent to the receiver over a distinct channel. It is thus guaranteed that the secret can be received even if some of the channels fail to deliver the message (the receiver would simply reconstruct the secret from the received shares, assuming these are in sufficient number), while an attacker eavesdropping on some of the channels would obtain no information at all on the secret message. This last application is especially relevant for this dissertation, since it allows us to cast the new constructions of Chapters 3 and 4 within a common framework.

Furthermore, Secret Sharing has become a fundamental cryptographic primitive since its introduction; for instance, secret-sharing schemes form building blocks for multi-party computation [6, 13, 18], byzantine agreement [56], threshold cryptography [25], access control [54], attribute-based encryption [34, 72], and generalized oblivious transfer [59, 71].

2.5.1 Basic Definitions and Properties

We formally introduce Secret Sharing in this section. We present secret-sharing schemes in two flavors; this gives us the flexibility to discuss new schemes in the most natural and convenient formulation. The two concepts are basically equivalent, namely there is a natural one-to-one correspondence between the two families of schemes; however, this correspondence does not, in general, preserve the *efficiency* of corresponding schemes.

Secret Sharing. We first introduce Secret Sharing in a flavor that matches the intuition given at the beginning of the section. We begin with the following basic definition:

Definition 2.5.1 (**Secret-Sharing Scheme**). *Let $n \geq 1$ be an integer, and $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_n$ be non-empty, finite sets; an* n-players Secret-Sharing Scheme $\mathtt{SSS}$ *is given by a randomized* sharing *algorithm* $\mathtt{Share} : \mathcal{S}_0 \to_{\$} \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ *and a deterministic* reconstruction *algorithm* $\mathtt{Reconstruct} : \big(\mathcal{S}_1 \sqcup \{\bot\}\big) \times \cdots \times \big(\mathcal{S}_n \sqcup \{\bot\}\big) \to \mathcal{S}_0 \sqcup \{\bot\}$ *with the property that* $\mathtt{Reconstruct}(\mathtt{Share}(s)) = s$ *with probability* 1 *for any* $s \in \mathcal{S}_0$.

The set $\mathcal{S}_0$ is called the secret space *of* $\mathtt{SSS}$*, while the sets $\mathcal{S}_1, \ldots, \mathcal{S}_n$ form the* share spaces.

Two families of subsets are associated to each secret-sharing scheme, namely its adversary structure (the sets of shares that yield no information on the secret) and its access structure (the sets of shares which allow the secret to be recovered). We do not formalize these notions in full generality: as said above,

we focus instead on "threshold" structures, meaning that small enough sets of shares belong to the adversary structure, while large enough sets sit in the access structure. We describe this setting by means of the privacy threshold and the reconstruction threshold of a scheme:

Definition 2.5.2. *Given an n-player Secret-Sharing Scheme* $\mathtt{SSS}$ *and two integers t, r with $0 \leq t < r \leq n$, we say that* $\mathtt{SSS}$ *has* t-privacy *and* r-reconstruction *if the following respective conditions hold:*

- *t-privacy: for any $\mathcal{A} \subseteq [n]$ with $1 \leq |\mathcal{A}| \leq t$, the distribution of $\mathtt{Share}(s)_{\mathcal{A}}$ does not depend on the choice of $s \in \mathcal{S}_0$ (where $\mathtt{Share}(s)_{\mathcal{A}} = \pi_{\mathcal{A}}(\mathtt{Share}(s))$).*
- *r-reconstruction: for any $\mathcal{B} \subseteq [n]$ with $|\mathcal{B}| \geq r$ and for any $s \in \mathcal{S}_0$, it holds for $\mathbf{s} = \mathtt{Share}(s)$ that $\mathtt{Reconstruct}(\tilde{\mathbf{s}}) = s$ with probability 1, where $\tilde{\mathbf{s}}$ is such that $\tilde{\mathbf{s}}_{\mathcal{B}} = \mathbf{s}_{\mathcal{B}}$ and $\tilde{\mathbf{s}}_i = \perp$ for any $i \notin \mathcal{B}$.*

If $r = t + 1$, we say that $\mathtt{SSS}$ *is a* threshold *scheme; otherwise, it is called a* ramp *scheme.*

The notion of t-privacy naturally extends to a *randomized* secret: for any random variable $x \in_{\$} \mathcal{S}_0$ independent of $\mathtt{Share}$, we have that x and $\mathtt{Share}_{\mathcal{A}}(x)$ are independent (as random variables) for any $\mathcal{A} \subseteq [n]$ with $|\mathcal{A}| \leq t$.

Randomized Secret Sharing. According to Definition 2.5.1, the sharing algorithms of a secret-sharing scheme takes the secret as input and outputs the corresponding shares. We discuss here a variant where the secret is randomly chosen and output by the scheme; this formulation is conceptually simpler, and some of the schemes we discuss are more naturally expressed in these terms.

Definition 2.5.3 (**Randomized Secret Sharing**). *Let $n \geq 1$ be an integer, and $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_n$ be non-empty, finite sets; an n-players* Randomized Secret-Sharing Scheme *is given by an $(n+1)$-tuple of random variables $(s, \mathbf{s}) \in_{\$} \mathcal{S}_0 \times (\mathcal{S}_1 \times \cdots \times \mathcal{S}_n)$ and a deterministic algorithm* $\mathtt{Reconstruct} : \big(\mathcal{S}_1 \sqcup \{\perp\}\big) \times \cdots \times \big(\mathcal{S}_n \sqcup \{\perp\}\big) \to \mathcal{S}_0 \sqcup \{\perp\}$*, with the following two properties:*

- *s is uniformly distributed over $\mathcal{S}_0$;*
- *$s = \mathtt{Reconstruct}(\mathbf{s})$ with probability 1.*

We will often be flexible with the notation and identify a randomized scheme with the n-tuple $(s, \mathbf{s})$, leaving the reconstruction algorithm implicit.

Furthermore, in case $(s, \mathbf{s})$ is uniformly distributed over a set $\mathcal{C}$ (which will often be the case throughout this thesis), we then identify the secret-sharing scheme with $\mathcal{C}$.

We will sometimes call *non-randomized* a secret-sharing scheme according to Definition 2.5.1, if it is necessary to distinguish the two.

The notions of privacy and reconstruction naturally translate to randomized secret-sharing schemes as follows:

Definition 2.5.4. *Let* SSS *be an n-player Randomized Secret-Sharing Scheme given by random variables $(s, \mathbf{s}) \in_\$ \mathcal{S}_0 \times (\mathcal{S}_1 \times \cdots \times \mathcal{S}_n)$; given two integers t, r with $0 \leq t < r \leq n$, we say that* SSS *has* t-privacy *if s and $\mathbf{s}_\mathcal{A}$ are independent for any $\mathcal{A} \subseteq [n]$ with $1 \leq |\mathcal{A}| \leq t$, and that it has* r-reconstruction *if it holds that* $\texttt{Reconstruct}(\tilde{\mathbf{s}}) = s$ *with probability* 1 *for any $\mathcal{B} \subseteq [n]$ with $|\mathcal{B}| \geq r$, where $\tilde{\mathbf{s}}$ is obtained from $\mathbf{s}$ by replacing each $\mathbf{s}_i$ for $i \notin \mathcal{B}$ with $\perp$.*

There is an obvious one-to-one correspondence between randomized and non-randomized schemes, given by assigning to a non-randomized scheme SSS with algorithms $(\texttt{Share}, \texttt{Reconstruct})$ the random variable $(s, \texttt{Share}(s))$ where $s \in_\$ \mathcal{S}_0$ is uniformly distributed and independent of Share; the reconstruction algorithm is still Reconstruct.

It is easily seen that the privacy and reconstruction thresholds are preserved by this correspondence; we can thus see the two definitions of secret sharing as two flavors of the same notion. Nevertheless, we stress once again the fact that this correspondence does not necessarily preserves efficiency: thus the two definitions are not entirely equivalent when it comes to implementing them.

Two examples of secret-sharing schemes. We first discuss the straightforward *n-out-of-n scheme,* where n is a positive integer. Let $\mathcal{S}$ be a finite abelian group, with additive notation; $\mathcal{S}$ will serve as secret space and share space. The randomized version of the scheme is given by the random variable $\left(\sum_{i=1,\ldots,n} \mathbf{s}_i, (\mathbf{s}_1, \ldots, \mathbf{s}_n)\right)$ where $(\mathbf{s}_1, \ldots, \mathbf{s}_n) \in_\$ \mathcal{S}^n$ is uniformly distributed.

The reconstruction algorithm simply adds the n shares; the non-randomized version of the scheme is given by the sharing algorithm Share, where $\texttt{Share}(s) = (\mathbf{s}_1, \ldots, \mathbf{s}_n)$ is a random vector with the property that $\sum_i \mathbf{s}_i = s$, and by the same reconstruction algorithm.

It is immediately seen that the scheme has $(n-1)$-privacy and n-reconstruction, hence justifying the name.

Another important and widely used example is *Shamir's scheme* [58], based on polynomial evaluation, which can be seen as a Reed-Solomon code adapted for Secret Sharing. The scheme is defined by first selecting a finite field $\mathbb{F}_q$ with $q \geq n + 1$, a privacy threshold $t \leq n$ and $n + 1$ elements $(\alpha_0, \alpha_1, \ldots, \alpha_n) \in \mathbb{F}_q^{n+1}$ with $\alpha_i \neq \alpha_j$ for any $i \neq j$. The two variants (non-randomized and randomized) of the scheme are then given as follows:

- The non-randomized scheme has sharing algorithm $\texttt{Share} : \mathbb{F}_q \to_\$ \mathbb{F}_q^n$ given as follows: for any $s \in \mathbb{F}_q$, $\texttt{Share}(s)$ is computed by first sampling a uniformly random polynomial in $\{P(x) \in \mathbb{F}_q[x] : \deg P \leq t,\ P(\alpha_0) = s\}$, then setting $\texttt{Share}(s) := (P(\alpha_1), \ldots, P(\alpha_n))$.

  The reconstruction algorithm works as follows: on input $\tilde{\mathbf{x}} \in (\mathbb{F}_q \sqcup \{\perp\})^n$, $\texttt{Reconstruct}$ first computes a polynomial $P(x) \in \mathbb{F}_q[x]$ with $\deg P \leq t$ and $P(\alpha_i) = \tilde{\mathbf{x}}_i$ for any $i : \tilde{\mathbf{x}}_i \neq \perp$, then outputs $P(\alpha_0)$.

- The randomized scheme is given the following Reed-Solomon code: $\{(P(\alpha_0), P(\alpha_1), \ldots, P(\alpha_n)) : P \in \mathbb{F}_q[x], \deg P \leq t\} \subseteq \mathbb{F}_q \times \mathbb{F}_q^n$, with the same reconstruction algorithm of the non-randomized scheme.

It is easily seen that the scheme has t-privacy and $(t+1)$-reconstruction (simply observe that the number of roots of a polynomial cannot exceed its degree).

2.6 Linear Secret Sharing and Error-Correcting Codes: Massey's Paradigm

In the examples we gave above, we pointed out that Shamir's scheme can be seen as a cryptographic variant of Reed-Solomon codes. More generally, notice that error-correcting codes and secret-sharing schemes share the goal of recovering data from a partially incomplete input; we speak of erasure-correction for codes and of (r-) reconstruction for secret-sharing schemes. In fact, there is a rich and fruitful interplay between Secret Sharing and Coding Theory [51, 19, 14, 69]. As a first step to discuss the connection between the two fields, we recall the concept of *linear* secret-sharing scheme, which also serves as a building block for other primitives such as Secure Multi-Party Computation. For the rest of this section, we focus on *randomized* schemes.

Definition 2.6.1. *Let $\mathbb{F}$ be a finite field and let $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_n$ be $\mathbb{F}$-vector spaces. A randomized secret-sharing scheme $(s, \mathbf{s}) \in_\$ \mathcal{S}_0 \times (\mathcal{S}_1 \times \cdots \times \mathcal{S}_n)$ is said to be* $\mathbb{F}$-linear *if the set $\mathcal{C}_{(s,\mathbf{s})} := \big\{(\hat{s}, \hat{\mathbf{s}}) : p\big((s, \mathbf{s}) = (\hat{s}, \hat{\mathbf{s}})\big) > 0\big\}$ is an $\mathbb{F}$-subspace of $\mathcal{S}_0 \times (\mathcal{S}_1 \times \cdots \times \mathcal{S}_n)$.*

Thus to every $\mathbb{F}$-linear secret sharing scheme $(s, \mathbf{s})$ is associated an $\mathbb{F}$-linear code $\mathcal{C}_{(s,\mathbf{s})}$. Massey [51] pointed out that the converse is also true: namely, any linear code satisfying certain non-degeneracy conditions gives rise to a secret-sharing scheme; recall that we typically identify a randomized scheme $(s, \mathbf{s})$ uniformly distributed over a set $\mathcal{C}$ with the set $\mathcal{C}$ itself (cf. Definition 2.5.3).

Lemma 2.6.1. *Let $\mathcal{C}$ be a linear code of length $n+1$ over a finite field $\mathbb{F}$; then $\mathcal{C}$ is a linear, randomized secret-sharing scheme if the following two conditions are met:*

(i) $\pi_0(\mathcal{C}) \neq \{0\}$.

(ii) If $(\mathbf{x}_0, 0, \ldots, 0) \in \mathcal{C}$, *then* $\mathbf{x}_0 = 0$ *as well.*

Proof. Since the projection π_0 is a linear map into $\mathbb{F}$ and is not identically zero by condition (i), then it is regular, i.e. every element in $\mathbb{F}$ has the same number of pre-images under π_0; this means that s is uniformly distributed over $\mathbb{F}$.

The reconstruction algorithm `Reconstruct` recovers the codeword $\mathbf{x} \in \mathcal{C}$ that matches the input symbols, and outputs $\mathbf{x}_0$; condition (ii) ensures that `Reconstruct` is well-defined, given that $\mathcal{C}$ is linear. Finally, the secret-sharing scheme obtained in this way is linear by definition.

□

While this connection may seem *a posteriori* straightforward, it is not straightforward to see how the parameters of the code translate to parameters of the corresponding secret-sharing scheme. We give here only a brief overview of Massey's analysis; the reader can refer to [19] for a more accurate and insightful discussion.

Let thus $\mathcal{C}$ be a linear code of parameters $[n+1, K, D]$ over a field $\mathbb{F}$. Then the reconstruction threshold of the secret-sharing scheme obtained from $\mathcal{C}$ is upper-bounded by $(n+1)-D+1$ (this can be easily proved by arguing over the minimum distance of $\mathcal{C}$); on the other hand, computing (or lower-bounding) the privacy threshold of the scheme requires working with the dual code $\mathcal{C}^\perp$, as showed in the following lemma.

Lemma 2.6.2. *Given a linear code* $\mathcal{C}$ *of length* $n+1$, *let* $\mathcal{C}^\perp$ *be its dual; let* $w := \min\{w_{\mathrm{H}}(\mathbf{y}) : \mathbf{y} \in \mathcal{C}^\perp, \mathbf{y}_0 \neq 0\}$, *and assume that* $w \geq 2$. *We then have that the secret-sharing scheme induced from* $\mathcal{C}$ *has* $t = (w-2)$*-privacy, where this bound is tight. In particular, we have that* $t \geq d_{\min}(\mathcal{C}^\perp) - 2$*; further, for any secret* $s \in \mathbb{F}$ *we have that any set of* $d_{\min}(\mathcal{C}^\perp) - 2$ *shares associated to* s *is uniformly distributed in* $\mathbb{F}^{d_{\min}(\mathcal{C}^\perp)-2}$.

Proof. We only give a sketch of the proof. Fix an arbitrary $\mathcal{B} \subseteq [n]$ with $|\mathcal{B}| = n - w + 2$ and let $\mathcal{A} := [n] \setminus \mathcal{B}$ (thus $\mathcal{A}$ is an arbitrary subset of $[n]$ of cardinality $w-2$). We then have that there exists a codeword $\hat{\mathbf{x}} \in \mathcal{C}$ with $\hat{\mathbf{x}}_0 = 1$ and $\hat{\mathbf{x}}_{\mathcal{A}} = \mathbf{0}$: this can be proved by observing that the map $\rho : \pi_{\mathcal{B}}(\mathcal{C}^\perp) \to \mathbb{F}$, $\mathbf{y}_{\mathcal{B}} \mapsto \mathbf{y}_0$ is well-defined (notice that since $|\mathcal{B}| = n - w + 2$, for any $\mathbf{y} \in \mathcal{C}^\perp$, if $\mathbf{y}_{\mathcal{B}} = \mathbf{0}$ then $\mathbf{y}_0 = 0$ as well), and then constructing $\hat{\mathbf{y}}$ from the coefficients of the linear form ρ (extended to $\mathbb{F}^{\mathcal{B}}$).

Thus for any $\mathcal{A} \subseteq [n]$ with $|\mathcal{A}| = w - 2$ there exists a codeword $\hat{\mathbf{x}} \in \mathcal{C}$ with $\hat{\mathbf{x}}_0 = 1$ and $\hat{\mathbf{x}}_{\mathcal{A}} = \mathbf{0}$. This means that the map $\mathcal{C} \to \mathbb{F} \times \pi_{\mathcal{A}}(\mathcal{C})$, $\quad \mathbf{x} \mapsto (\mathbf{x}_0, \mathbf{x}_{\mathcal{A}})$

is surjective: indeed, notice that a preimage of $(s, \mathbf{x}_{\mathcal{A}})$ is given by $\mathbf{x}+(s-\mathbf{x}_0)\cdot\hat{\mathbf{x}}$.

By the linearity of this map, it can then be easily argued that for a random $\mathbf{x} \in_{\$} \mathcal{C}$, $\mathbf{x}_0$ and $\mathbf{x}_{\mathcal{A}}$ are independently distributed: thus the scheme has $(w-2)$-privacy.

To see that the bound is tight, observe that by definition of w there exists $\hat{\mathbf{y}} \in \mathcal{C}^{\perp}$ and $\mathcal{B} \subseteq [n]$, $|\mathcal{B}| = w-1$ with $\hat{\mathbf{y}}_0 \neq 0$ and $\hat{\mathbf{y}}_{\bar{\mathcal{B}}} = \mathbf{0}$. Now since $\hat{\mathbf{y}} \in \mathcal{C}^{\perp}$, we have that $\mathbf{x}\hat{\mathbf{y}}^T = 0$ for any $\mathbf{x} \in \mathcal{C}$, which means that $\mathbf{x}_0 = -(\mathbf{x}_{\mathcal{B}}\hat{\mathbf{y}}_{\mathcal{B}}^T)/\hat{\mathbf{y}}_0$; thus the $w-1$ symbols of $\mathbf{x}$ in $\mathcal{B}$ uniquely determine $\mathbf{x}_0$, so that there is no $(w-1)$-privacy.

For what concerns our last claim, let $\mathcal{A}'$ be a subset of $\{1, \ldots, n\}$ of cardinality $d_{\min}(\mathcal{C}^{\perp}) - 2$; thanks to the linearity of π_0 and $\pi_{\mathcal{A}'}$, it suffices to prove that the projection map $\pi_{\mathcal{A}'}$ is surjective on $\{\mathbf{x} \in \mathcal{C} : \mathbf{x}_0 = s\}$ for any secret $s \in \mathbb{F}$.

To this end, let $\mathcal{A}'' := \{0\} \cup \mathcal{A}'$; we show that the projection map $\pi_{\mathcal{A}''} : \mathcal{C} \to \mathbb{F}^{\mathcal{A}''}$ is surjective, which implies the claim. Indeed, let $G := (\mathbf{x}^{(j)} : j = 1, \ldots, \dim(\mathcal{C}))$ be a generator matrix of $\mathcal{C}$, and let $G_{\mathcal{A}''}$ be the submatrix of G with column indexes in $\mathcal{A}''$. Since $|\mathcal{A}''| = d_{\min}(\mathcal{C}^{\perp}) - 1$, we have that if $G_{\mathcal{A}''} \cdot \mathbf{y}_{\mathcal{A}''}^T = \mathbf{0}$, then $\mathbf{y}_{\mathcal{A}''} = \mathbf{0}$: indeed, by adding 0 on the coordinates outside $\mathcal{A}''$ we can complete $\mathbf{y}_{\mathcal{A}''}$ to $\mathbf{y} \in \mathcal{C}^{\perp}$ with $w_{\mathrm{H}}(\mathbf{y}) = w_{\mathrm{H}}(\mathbf{y}_{\mathcal{A}''})$. This means that $\ker(G_{\mathcal{A}''}) = \{\mathbf{0}\}$; by the rank-nullity theorem, this implies that the rows of $G_{\mathcal{A}''}$ span the whole $\mathbb{F}^{\mathcal{A}''}$, which concludes the proof.

□

A similar tight bound can be computed for the reconstruction threshold, though discussing it would be beyond the scope of this dissertation. Also notice that in case $\mathcal{C}$ is an MDS code of parameters $[n+1, t+1, n-t+1]$, then the corresponding scheme has t-privacy and $(t+1)$-reconstruction: this can be easily proved by working with information sets of $\mathcal{C}$.

This connection has the following downside: computing the parameter w of Lemma 2.6.2 is often not possible, and the dual distance $d_{\min}(\mathcal{C}^{\perp})$ is used instead as a lower bound. Thus codes are needed that have at the same time good distance (to control reconstruction) and good dual distance (to control privacy), and this can be problematic. We further discuss this issue in Chapter 3, motivating our search for a new connection between Secret Sharing and Error-Correcting Codes.

Chapter 3

New Constructions of Secret-Sharing Schemes from Error-Correcting Codes

We present in this chapter our contributions to the field of Secret Sharing. We start by establishing a new connection between Secret Sharing and Error-Correcting Codes, that allows for a better control of the privacy and reconstruction thresholds compared to previous work.

We then show two applications of this connection: the first and more direct one is discussed in Section 3.2, and yields a family of secret-sharing schemes with linear-time sharing and reconstruction.

The second application is presented in Section 3.3; we use our new connection as a starting point to construct robust secret-sharing schemes, making use of highly list-decodable codes and AMD codes. We show how in this way we can obtain robust secret-sharing schemes with optimal overhead.

The content of this chapter is based on the article [20], written with Ronald Cramer, Ivan Damgård, Nico Döttling and Serge Fehr.

3.1 A New Connection between Secret Sharing and Error-Correcting Codes

As we have discussed in Chapter 2, Section 2.6, linear secret-sharing schemes and linear error-correcting codes are tightly connected, namely Massey pointed out that there is a natural one-to-one correspondence between them (albeit some degenerate codes have to be excluded).

In the same section, we remarked that the key point of Massey's analysis is that it allows us to compute (or estimate) fundamental properties of a secret-sharing scheme in terms of the underlying code and of its dual. We now recall how this estimation works, and point out its shortcomings.

A linear code $\mathcal{C}$ of length $n+1$ over a field $\mathbb{F}$ is a secret-sharing scheme if two non-degeneracy conditions on $\mathcal{C}$ are met (we implicitly consider a random variable $(s, \mathbf{s}) \in \mathbb{F}^{n+1}$ uniformly distributed over $\mathcal{C}$). Massey's analysis allows us to compute the privacy and reconstruction thresholds in terms of parameters of $\mathcal{C}$ and of its dual code $\mathcal{C}^\perp$; tight values can be obtained in this way, but bounds (not necessarily tight) are easier to compute: namely, the reconstruction threshold r satisfies $r \leq n - d_{\min}(\mathcal{C}) + 2$, while the privacy threshold t satisfies[1] $t \geq d_{\min}(\mathcal{C}^\perp) - 2$.

Hence to estimate the privacy and reconstruction parameters in this way, one needs to control both the code $\mathcal{C}$ and its dual, which can sometimes be troublesome. For instance, if we want to use Massey's paradigm to construct a family of secret-sharing schemes with linear-time sharing and reconstruction, we need a family of linear-time encodable and decodable codes; however, currently known codes with these properties have very bad dual codes, so that the bound on the privacy is too low to be useful.

We circumvent this problem by establishing a new connection between linear secret-sharing schemes and linear codes; the key point of this alternative connection is that it de-couples the privacy threshold of a scheme from the dual of the underlying code, hence allowing to compute better bounds for the former.

Intuitively, our connection differs from Massey's for the following reason: in Massey's, both the shares *and* the secret of a secret-sharing scheme are seen as the symbols of a codeword in a certain code; on the other hand, in our construction the shares are still given by symbols of a codeword, but the secret is given by an arbitrary *function* acting on the codeword.

We formalize this in the following definition; here, the "decoding algorithm" of a code $\mathcal{C}$ that we make use of takes as input a word $\mathbf{y} \in (\mathbb{F}^m \sqcup \{\perp\})^n$, and

[1] Here, we implicitly assume that $d_{\min}(\mathcal{C}), d_{\min}(\mathcal{C}^\perp) \geq 2$.

outputs a codeword $\mathbf{x} \in \mathcal{C}$ with $d_{\mathtt{H}}(\mathbf{x}, \mathbf{y}) < d_{\min}(\mathcal{C})$. If such a codeword does not exist or is not unique, the algorithm outputs an error message $\perp$.

Definition 3.1.1 (**Secret Sharing from Codes**). *Let $\mathcal{C}$ be a (folded) $\mathbb{F}$-linear code of length n with decoding algorithm* `Decode`*, and let $f : \mathcal{C} \to \mathcal{S}_0$ be a surjective linear map, where $\mathcal{S}_0 \neq \{0\}$ is a finite-dimensional $\mathbb{F}$-vector space. The associated secret-sharing scheme* $\mathtt{SSS}_{(\mathcal{C},f)}$ *is then given by*

- `Share` $: \mathcal{S}_0 \to_{\$} \mathcal{C}$*, where* `Share`$(s)$ *is uniformly distributed over* $f^{-1}(s)$ *for any* $s \in \mathcal{S}_0$*;*
- `Reconstruct` $: (\mathbb{F}^m \sqcup \{\perp\})^n \to \mathcal{S}_0 \sqcup \{\perp\}$ *where* `Reconstruct`$(\mathbf{y}) :=$ $f($`Decode`$(\mathbf{y}))$.

Similarly, the associated randomized secret-sharing scheme $\mathtt{RanSSS}_{(\mathcal{C},f)}$ *is given by the space*

$$\{(f(\mathbf{x}), \mathbf{x}) \,:\, \mathbf{x} \in \mathcal{C}\}$$

where the reconstruction algorithm `Reconstruct` *is the same as above.*

It is immediately seen that the schemes obtained in this way are $\mathbb{F}$-linear.

Our new connection and Massey's are related in the following way: if $\mathcal{C}'$ is the code of length $n+1$ in Massey's connection, then the code $\mathcal{C}$ of our's is the *punctured* code

$$\mathcal{C} := \{\mathbf{x} \in \mathbb{F}_q^n \,:\, \exists\, x \in \mathbb{F} \text{ with } (x, \mathbf{x}) \in \mathcal{C}'\}.$$

and the reconstruction map $f : \mathcal{C} \to \mathbb{F}$ is simply given by $\mathbf{x} \mapsto x : (x, \mathbf{x}) \in \mathcal{C}'$.

Remark 3.1.1. Let `SSS` and `RanSSS` be the secret-sharing schemes obtained from a code $\mathcal{C}$ and a function f as in Definition 3.1.1. We then have that sharing a random secret via `SSS` yields the same distribution as `RanSSS`: indeed, notice that since `Share`$(\hat{s})$ is uniformly distributed over $f^{-1}(\hat{s})$ for any $\hat{s}$, the claim holds if we show that $f(\mathbf{x})$ and `Share`(s) are uniformly distributed in $\mathcal{S}_0$ and $\mathcal{C}$, respectively. Now since f is linear and surjective, then it is also *regular,* i.e. every element in the codomain of f has a constant number of preimages under f, so that the claim follows. In particular, the privacy and reconstruction thresholds of `SSS` and `RanSSS` coincide.

The interesting point of this connection is computing the privacy and reconstruction thresholds of a secret-sharing scheme in terms of the underlying code and surjective function. We discuss privacy and reconstruction separately; thanks to the above remark, it suffices to compute the thresholds for the randomized scheme.

Proposition 3.1.1 (**Reconstruction Threshold**). *Given positive integers n and $r \leq n$, let $\mathcal{C}$ be a (folded) linear code of length n over $\mathbb{F}$ and let $f : \mathcal{C} \to \mathcal{S}_0$ be a surjective, $\mathbb{F}$-linear function.*

Assume that $\mathcal{C}$ can be corrected from $n - r$ erasures; we then have the secret-sharing schemes $\mathtt{SSS}_{(\mathcal{C},h)}$ and $\mathtt{RanSSS}_{(\mathcal{C},h)}$ obtained as in Definition 3.1.1 have r-reconstruction.

Proof. We prove the statement for $\mathtt{RanSSS}_{(\mathcal{C},h)}$. Let $\mathbf{x}$ denote the random variable uniformly distributed over $\mathcal{C}$; fix a subset $\mathcal{B} \subseteq [n]$ with $|\mathcal{B}| \geq r$, and let $\tilde{\mathbf{x}} \in_{\$} (\mathbb{F}^m \sqcup \{\perp\})^n$ be defined by $\tilde{\mathbf{x}}_{\mathcal{B}} := \mathbf{x}_{\mathcal{B}}$, $\tilde{\mathbf{x}}_i := \perp$ for any $i \notin \mathcal{B}$.

Since $\mathcal{C}$ can be corrected from $n - r$ erasures, we have that $\mathtt{Decode}(\tilde{\mathbf{x}}) = \mathbf{x}$ with probability 1; hence $f(\mathtt{Decode}(\tilde{\mathbf{x}})) = f(\mathbf{x})$ with probability 1, so that the claim is proved.

□

Next, we focus on the privacy threshold, which now no longer depends on the dual of the code $\mathcal{C}$; the following characterization was first given as Theorem 10 of [14], where it was used to prove the existence of ramp schemes with high information rate (i.e., high ratio between secret size and share size). We rephrase it here to match our formulation.

Proposition 3.1.2 (**Privacy Threshold**). *Let $\mathcal{C}$ be a (folded) linear code of length n over $\mathbb{F}$ and let $f : \mathcal{C} \to \mathcal{S}_0$ be a surjective, $\mathbb{F}$-linear function.*

Let $0 < t < n$ be an integer; then the secret-sharing schemes $\mathtt{SSS}_{(\mathcal{C},h)}$ and $\mathtt{RanSSS}_{(\mathcal{C},h)}$ obtained from $\mathcal{C}$ and f have t-privacy if and only if $f(\ker(\pi_{\mathcal{A}})) = \mathcal{S}_0$ for any $\mathcal{A} \subseteq [n]$ with $|\mathcal{A}| = t$, where $\pi_{\mathcal{A}}$ denotes the projection map from $\mathcal{C}$ to $\mathbb{F}^{\mathcal{A}}$.

Proof. We prove the statement for $\mathtt{SSS}_{(\mathcal{C},h)}$. Fix an arbitrary $\mathcal{A} \subseteq [n]$ with $|\mathcal{A}| = t$; clearly, the claim holds if we prove the following equivalence:

$$\pi_{\mathcal{A}}(f^{-1}(s)) = \pi_{\mathcal{A}}(f^{-1}(0)) \text{ for any } s \in \mathcal{S}_0$$
$$\Updownarrow$$
$$\text{there exists } \mathbf{x} \in f^{-1}(s) \text{ with } \pi_{\mathcal{A}}(\mathbf{x}) = \mathbf{0} \text{ for any } s \in \mathcal{S}_0.$$

Now fix an arbitrary $s \in \mathcal{S}_0$; observe that for any $s \in \mathcal{S}_0$ we can write $f^{-1}(s) = \mathbf{x}' + \ker f$ where $\mathbf{x}'$ is any fixed element of $f^{-1}(s)$; thus $\pi_{\mathcal{A}}(f^{-1}(s)) = \pi_{\mathcal{A}}(\mathbf{x}') + \pi_{\mathcal{A}}(\ker f)$, so that

$$\begin{gathered} \pi_{\mathcal{A}}(f^{-1}(s)) = \pi_{\mathcal{A}}(f^{-1}(0)) \\ \Updownarrow \\ \text{there exists } \mathbf{x}' \in f^{-1}(s) \text{ with } \pi_{\mathcal{A}}(\mathbf{x}') = \mathbf{0}. \end{gathered}$$

The claim follows.

□

As a concluding remark for this section, notice that a downside of our connection is that it provides no obvious way to construct *multiplicative* secret-sharing schemes, where a secret-sharing scheme is called multiplicative if there exists a vector $\mathbf{v}$ such that for any two share vectors $\mathbf{s}$ and $\mathbf{s}'$ associated to secrets s, s' respectively, it holds that $ss' = \mathbf{v}(\mathbf{s} \star \mathbf{s}')^T$, where $\star$ denotes the component-wise product. Multiplicativity is an important property because it allows us to use secret-sharing schemes to construct protocols for Multi-Party Computation [18, 14]; it is thus an interesting open problem to modify our construction paradigm so that it yields multiplicative schemes.

3.2 A First Application: Linear-Time Sharing and Reconstruction via Linear Universal Hash Functions

We present a first application of the connection discussed in the previous section, using universal hash functions and linear-time encodable and decodable codes. With some twist on the construction, this yields a family of linear secret-sharing schemes with linear-time sharing and reconstruction.

3.2.1 Universal Hash Functions

We describe in this section universal hash functions, an important tool in information-theoretic cryptography; these will be used together wit suitable codes to construct a family of secret-sharing schemes according to Definition 3.1.1.

Definition 3.2.1 (**Universal Hash Functions**). *Let $\mathcal{X}$ and $\mathcal{Y}$ be finite, non-empty sets, let $\mathcal{H}$ be a set of functions $\mathcal{X} \to \mathcal{Y}$; denote by $h : \mathcal{X} \to_\$ \mathcal{Y}$ the randomized function given by the uniform distribution over $\mathcal{H}$. We say that $\mathcal{H}$ is a* family of universal hash functions *if for any $x \neq x' \in \mathcal{X}$ we have that*

$$p(h(x) = h(x')) \leq \frac{1}{|\mathcal{Y}|}.$$

For families $\mathcal{H}$ of $\mathbb{F}_q$-*linear* functions, meaning that both $\mathcal{X}$ and $\mathcal{Y}$ are $\mathbb{F}_q$-vector spaces and each $\hat{h} \in \mathcal{H}$ is a $\mathbb{F}_q$-linear mapping, Definition 3.2.1 can be rephrased as follows: $\mathcal{H}$ is a family of universal hash functions if and only if for any $x \in \mathcal{X} \setminus \{0\}$ we have that

$$p(h(x) = 0) \leq q^{-\dim \mathcal{Y}}.$$

We then naturally refer to $\mathcal{H}$ as a family of *linear* universal hash functions.

There are various efficient families of linear universal hash functions, such random matrices or random Toeplitz matrices (see e.g. [50]). Druk and Ishai [28] constructed a linear time computable family of linear universal hash functions, c.f. Section 3.2.3.

An important property of linear universal hash functions is that they are surjective on subspaces with high probability; in order to formally state and prove this result, we will need some further tools from probability theory.

First, the binary entropy function $H : [0, 1/2] \to [0, 1]$ is defined by $H(0) := 0$ and $H(x) := -x \cdot \log(x) - (1-x) \cdot \log(1-x)$ for $x \in (0, 1/2]$. For $0 \leq t/n \leq 1/2$ we can upper bound binomial coefficients by $\binom{n}{t} \leq 2^{H(t/n) \cdot n}$, for a proof see e.g. [53]. We will also use the Markov inequality (see also [53]):

Lemma 3.2.1 (**Markov Inequality**). *Let $x \in_\$ \mathcal{X}$ be a (finite) random variable where $\mathcal{X} \subseteq \mathbb{R}_{\geq 0}$. Then it holds for every $\hat{x} > 0$ that*

$$p(x \geq \hat{x}) \leq \frac{\mathsf{E}[x]}{\hat{x}}$$

where $\mathsf{E}[x] := \sum_{\hat{x} \in \mathcal{X}} \hat{x} \cdot p(x = \hat{x})$.

Corollary 3.2.2. *Let $x \in_\$ \mathcal{X}$ be an abstract random variable with $\mathcal{X} \subseteq \mathbb{R}_{\geq 0}$ such that x assumes its minimum at x_0 and its second smallest value at $x_1 > x_0$. Then it holds that*

$$\mathsf{E}[x] \geq x_0 + (x_1 - x_0) \cdot p(x \neq x_0).$$

Proof. Since x assumes its minimum at x_0 it holds that $x - x_0$ is non-negative. By the Markov inequality it holds that

$$p(x \neq x_0) = p(x \geq x_1) = p(x - x_0 \geq x_1 - x_0) \leq \frac{\mathsf{E}[x] - x_0}{x_1 - x_0},$$

as $\mathsf{E}[x - x_0] = \mathsf{E}[x] - x_0$ (by linearity of expectation - see [53]). Thus the claim follows.

□

We can now prove the following result on the surjectivity of hash functions:

Proposition 3.2.3. *Let k, l, r be positive integers; let $\mathcal{H}$ be a family of linear universal hash functions $\mathbb{F}_q^k \to \mathbb{F}_q^l$. Denote by $h : \mathbb{F}_q^k \to_{\$} \mathbb{F}_q^l$ the randomized function with uniform distribution over $\mathcal{H}$, and let V be a subspace of $\mathbb{F}_q^k$ of dimension r. We then have that*

$$p(h(V) \neq \mathbb{F}_q^l) \leq \frac{1}{q^{r-l}}$$

i.e., h is surjective over V except with probability at most $1/q^{r-l}$.

Proof. For any linear function $\hat{h} \in \mathcal{H}$, it holds that $\hat{h}(V) = \mathbb{F}_q^l$ if and only if $\dim(V \cap \ker(\hat{h})) = \dim(V) - l$, which is equivalent to $|V \cap \ker(\hat{h})| = \frac{|V|}{q^l}$. Now define the random variable $x = |V \cap \ker(h)|$ (depending on h). By the above it holds that h is surjective on V if and only if $x = |V|/q^l$. For each $\mathbf{v} \in V$, define the random variable

$$x_{\mathbf{v}} = \begin{cases} 1 & \text{if } h(\mathbf{v}) = 0 \\ 0 & \text{otherwise} \end{cases}$$

Clearly, it holds that $x = \sum_{\mathbf{v} \in V} x_v$. Since $x_{\mathbf{0}} = 1$, we have that $x = 1 + \sum_{\mathbf{v} \in V \setminus \{\mathbf{0}\}} x_{\mathbf{v}}$. Moreover, x assumes its minimum at $\frac{|V|}{q^l}$ and its second smallest value at $\frac{|V|}{q^{l-1}}$. We will now compute the expectation $\sum_{\hat{x}} \hat{x} \cdot p(x = \hat{x})$ of x. For each $\mathbf{v} \in V \setminus \{\mathbf{0}\}$ it holds that

$$\mathsf{E}[x_{\mathbf{v}}] = p(h(\mathbf{v}) = \mathbf{0}) \leq q^{-l},$$

as $\mathcal{H}$ is a family of universal hash functions. By linearity of expectation, it holds that

$$\mathsf{E}[x] = 1 + \sum_{\mathbf{v} \in V \setminus \{\mathbf{0}\}} \mathsf{E}[x_{\mathbf{v}}] = 1 + \frac{|V| - 1}{q^l}.$$

By Corollary 3.2.2 and the fact that $|V| \geq q^r$ it holds that

$$\begin{aligned} p\left(x \neq \frac{|V|}{q^l}\right) &\leq \frac{1 + \frac{|V|-1}{q^l} - \frac{|V|}{q^l}}{\frac{|V|}{q^{l-1}} - \frac{|V|}{q^l}} \\ &= \frac{q^l - 1}{|V| \cdot (q-1)} \\ &\leq \frac{q^l}{|V|} \leq q^{-(r-l)}. \end{aligned}$$

Consequently, it holds that $h(V) = \mathbb{F}_q^l$, except with probability $q^{-(k-l)}$.

□

By applying a union bound, we get the following corollary:

Corollary 3.2.4. *Let k, l, r be positive integers, let $\mathcal{H}$ be a family of linear universal hash functions $\mathbb{F}_q^k \to \mathbb{F}_q^l$ and $\mathcal{V}$ be a collection of subspaces of $\mathbb{F}_q^k$, each of dimension at least r; let $h : \mathbb{F}_q^k \to_{\$} \mathbb{F}_q^l$ be given by the uniform distribution over $\mathcal{H}$. Then it holds that $h(V) = \mathbb{F}_q^l$ for all $V \in \mathcal{V}$ (i.e. h is surjective on all subspaces in V), except with probability at most $|\mathcal{V}| \cdot q^{-(r-l)}$.*

3.2.2 A New Scheme from Codes and Universal Hash Functions

In this section we implement the connection of Definition 3.1.1 to obtain a secret-sharing scheme from linear codes and universal hash functions. Notice however that we make use of a slight variation of the concept, by means of the encoding and decoding algorithms `Enc` and `Dec` of the code $\mathcal{C}$ as well. Here, $\texttt{Enc} : \mathbb{F}_q^k \to \mathcal{C}$ simply realizes the isomorphism between the two spaces, while $\texttt{Dec} : (\mathbb{F}_q^m \sqcup \{\perp\})^n \to \mathbb{F}_q^k \sqcup \{\perp\}$ first decodes (if possible) the input $\mathbf{y}$ from errors or erasures to obtain a codeword $\mathbf{x} \in \mathcal{C}$, and then inverts `Enc` to obtain a vector in $\mathbb{F}_q^k$.

In this whole section, whenever we speak of a family of linear universal hash functions $\mathbb{F}_q^k \to \mathbb{F}_q^l$, we assume that q, k and l are fixed, i.e. they are the same for each function in the family.

Construction 3.2.1. *Let $\mathcal{C}$ be an m-folded, $\mathbb{F}_q$-linear code of parameters $[n,k,d]$, endowed with encoding and decoding algorithms $\mathtt{Enc} : \mathbb{F}_q^k \to \mathcal{C}$ and $\mathtt{Dec} : (\mathbb{F}^m \sqcup \{\perp\})^n \to \mathbb{F}_q^k \sqcup \{\perp\}$. Further, let $\mathcal{H}$ be a family of linear universal hash functions $\mathbb{F}_q^k \to \mathbb{F}_q^l$; fix an arbitrary $h \in \mathcal{H}$.*

We can then connect $\mathcal{C}$ and $h : \mathcal{C} \to \mathbb{F}_q^l$ to obtain a secret-sharing scheme $\mathtt{SSS} = \mathtt{SSS}_{(\mathcal{C},h)}$ following Definition 3.1.1, namely $\mathtt{SSS}$ is given by the following sharing and reconstruction algorithms:

- $\mathtt{Share} : \mathbb{F}_q^l \to_{\$} \mathcal{C}$ *where* $\mathtt{Share}(s)$ *is uniformly distributed over* $\mathtt{Enc}(h^{-1}(s))$*;*
- $\mathtt{Reconstruct} : \left(\mathbb{F}_q^m \sqcup \{\perp\}\right)^n \to \mathbb{F}_q^l \sqcup \{\perp\}$ *where* $\mathtt{Reconstruct}(\mathbf{y}) := h(\mathtt{Dec}(\mathbf{y}))$.

Similarly, we obtain the randomized secret-sharing scheme $\mathtt{RanSSS}$ given by the set $\left\{(h(\mathbf{z}), \mathtt{Enc}(\mathbf{z})) \in_{\$} \mathbb{F}_q^l \times \mathcal{C} \,:\, \mathbf{z} \in_{\$} \mathbb{F}_q^k\right\}$, and with the same reconstruction algorithm.

We can compute the parameters of these secret-sharing schemes in terms of the underlying codes and hash functions, using Propositions 3.1.1 and 3.1.2:

Lemma 3.2.5. *Let $\mathcal{C}$ be an m-folded linear code of parameters $[n,k,d]$ and $\mathcal{H}$ a family of linear universal hash functions as in Construction 3.2.1. Let $R := \frac{k}{mn}$ be the rate of $\mathcal{C}$, let $\rho := \frac{l}{nm}$ and let $\tau > 0$ and $\eta > 0$ be real constants; assume that $R \geq \rho + \eta + \tau + H(\tau)/(m \cdot \log(q))$.*

Then for every $h \in \mathcal{H}$, the scheme $\mathtt{SSS} := \mathtt{SSS}_{(\mathcal{C},h)}$ given by Construction 3.2.1 has $(n-d+1)$-reconstruction. Furthermore, there exists a function $h \in \mathcal{H}$ such that $\mathtt{SSS}_{(\mathcal{C},h)}$ has τn privacy; such a function can be chosen randomly with success probability $1 - q^{-\eta n m}$.

Proof. Reconstruction is an immediate consequence of Proposition 3.1.1, since $\mathcal{C}$ can correct from $d-1$ erasures.

For privacy, notice that for each set $\mathcal{A} \subseteq \{1,\ldots,n\}$ of size at most $t := \tau n$, it holds that $\ker(\pi_{\mathcal{A}}) \subseteq \mathcal{C}$ is a subspace of dimension at least $k - mt$, as $\mathcal{C}$ has dimension k and the image of $\pi_{\mathcal{A}}$ has dimension at most mt. Thus, $\mathtt{Dec}(\ker(\pi_{\mathcal{A}})) \subseteq \mathbb{F}_q^k$ also has dimension at least $k - mt$, as $\mathtt{Dec}$ is an isomorphism. Consequently $\mathcal{V} = \{\mathtt{Dec}(\ker(\pi_{\mathcal{A}})) \;:\; \mathcal{A} \in \{1,\ldots,n\}, |\mathcal{A}| = t\}$ is a

collection of subspaces of dimension at least $k - mt$. Moreover, as $\mathcal{A}$ is taken over all subsets of $\{1, \ldots, n\}$ of size t, it holds that

$$|\mathcal{V}| \leq \binom{n}{t} \leq 2^{H(t/n)\cdot n} = 2^{H(\tau)\cdot n} = q^{\frac{H(\tau)}{m\log(q)}\cdot mn}.$$

By Proposition 3.1.2, SSS has t-privacy if it holds that $h(V) = \mathbb{F}_q^l$ for each $V \in \mathcal{V}$. By Corollary 3.2.4, it holds for all $V \in \mathcal{V}$ that $h(V) = \mathbb{F}_q^l$, except with probability

$$|\mathcal{V}| \cdot q^{-(k-mt-l)} \leq q^{-(k-mt-l-\frac{H(\tau)}{m\log(q)}\cdot mn)} = q^{-(R-\tau-\rho-\frac{H(\tau)}{m\log(q)})\cdot mn} \leq q^{-\eta mn},$$

as $R \geq \rho + \eta + \tau + H(\tau)/(m \cdot \log(q))$, where with a slight abuse of notation, we still denote by h the randomized function given by the uniform distribution over $\mathcal{H}$. Thus, SSS has t-privacy, except with probability $q^{-\eta mn}$ over the choice of h. This concludes the proof.

□

In the next section we show that, with some tweaking, Construction 3.2.1 allows us to define a family of linear secret-sharing schemes with linear-time sharing and reconstruction.

3.2.3 A Linear-Time Family of Secret-Sharing Schemes

The goal of this section is to instantiate Construction 3.2.1 to obtain a family of secret-sharing schemes with linear-time sharing and reconstruction. Some tweaking of the original construction will be necessary to achieve the goal.

We start by recalling the existence of linear-time computable universal hash functions and linear-time encodable and decodable codes. Ishai et al. [39] construct a family of $\mathbb{F}_2$-linear universal hash functions which can be computed in linear time; this result has recently been generalized by Druk and Ishai [28] to any finite field.

Theorem 3.2.6 (**Druk and Ishai [28]**). *For every integers $0 < l < k$, there exists a family of $\mathbb{F}_2$-linear universal hash functions $\mathcal{H}$ mapping $\mathbb{F}_2^k$ to $\mathbb{F}_2^l$ which can be computed in time linear in k.*

We give here some intuition on how the above family is constructed for given l and k. The process works in three phases: first, a linear-time encodable code $\mathcal{C}$ is selected; on input $\mathbf{x} \in \mathbb{F}^k$, the first step requires computing $\mathbf{y} := \mathtt{Enc}(\mathbf{x})$,

where $\texttt{Enc}$ denotes the encoding function of $\mathcal{C}$. We stress the fact that $\mathcal{C}$ may have have alphabet size greater than 2, allowing it to have a relative minimum distance bigger than $1/2$. For the second step, an independent linear universal hash function h_i is applied to each $\mathbf{y}_i$, resulting in $\mathbf{y}'$; there are no requirements on the efficiency of the (h_i), but it is assumed that each h_i has constant size, so that the second step can still be performed in linear time. Finally, a linear-time encodable code $\mathcal{C}'$ with good minimal distance is selected; if $\mathbf{v} \mapsto G\mathbf{v}^T$ denotes the encoding function of $\mathcal{C}'$, we then define the final output to be $\mathbf{y}'' := G^T(\mathbf{y}')^T$. It can be proved that this last step can also be performed in linear time, so that the overall mapping can indeed be computed in linear time. Notice that $\mathcal{C}$ and $\mathcal{C}'$ are fixed, so that a specific function in the family is specified by the choice of the (h_i).

Informally stated, this construction yields a family of universal hash functions for the following reason. Let $\mathbf{x}$ be a non-zero input; then $\mathbf{y} = \texttt{Enc}(\mathbf{x})$ is non-zero at many coordinates, since $\mathcal{C}$ has large minimum distance. In turn, this means that $\mathbf{y}'$ is non-zero at these coordinates with high probability; finally, it can be showed that this implies that $\mathbf{y}'' = G^T(\mathbf{y}')^T$ is non-zero *tout court* with high probability (we say that $\mathbf{v} \mapsto G^T\mathbf{v}^T$ is an *extractor for bit-fixing sources*), which proves the claim.

There is a large corpus of work dealing with linear-time encodable codes, starting with the seminal work of Spielman [64]. To the best of our knowledge, the currently best known parameters can be obtained using a family of codes by Guruswami and Indyk [36].

Theorem 3.2.7 (**Guruswami-Indyk [36]**). *For every real number $R > 0$ and every sufficiently small $\epsilon \in \mathbb{R}_{>0}$ (depending on R) there exists an infinite family of m-folded $\mathbb{F}_2$-linear codes $\{\mathcal{C}_n\}$ of rate R, where $m = O\left(\frac{\log(1/\epsilon)}{\epsilon^4 R}\right)$ for $\epsilon \to 0$, such that the codes from the family can be encoded in linear time and also decoded in linear time from an $1 - R - \epsilon$ fraction of erasures.*

We will now instantiate the the randomized secret-sharing scheme $\texttt{RanSSS}_{(\mathcal{C},h)}$ of Construction 3.2.1 with the codes from Theorem 3.2.7 and universal hash functions from Theorem 3.2.6.

Lemma 3.2.8. *For all real constants $0 < \tau < \sigma < 1$ there exists an infinite family of $\mathbb{F}_2$-linear randomized secret-sharing schemes $\{\texttt{RanSSS}_n\}$ with τn-privacy and σn-reconstruction. The shares of $\texttt{RanSSS}_n$ have size m bits, where $m > 0$ is a constant integer; furthermore, the random shares and associated secret of $\texttt{RanSSS}_n$ can be computed in time linear in n, and reconstructing from σn shares can also be performed in linear time.*

Such a scheme $\texttt{RanSSS}_n$ can be constructed randomly with success probability $1 - 2^{-\eta m n}$ (for some real constant $\eta > 0$ depending on τ and σ).

Proof. We will instantiate the randomized secret sharing scheme $\texttt{RanSSS}_{(\mathcal{C},h)}$ from Construction 3.2.1 with a linear code $\mathcal{C}_n$ from the family $\{\mathcal{C}_n\}$ of $\mathbb{F}_2$-linear codes from Theorem 3.2.7 and a function h from the family $\mathcal{H}$ of $\mathbb{F}_2$-linear universal hash functions from Theorem 3.2.6. We now show how to choose the parameters for this instantiation.

By Lemma 3.2.5, in order to obtain a secret-sharing scheme with τn privacy, we need to select an m-folded code $\mathcal{C}_n$ from the above family of length n and rate R such that $R \geq \rho + \eta + \tau + H(\tau)/m$ for arbitrarily small constants η and ρ. Moreover, as by Proposition 3.1.1 we need to be able to correct a $1-\sigma$ fraction of erasures to have σn reconstruction, we need to choose $\mathcal{C}_n$ such that $1 - \sigma \leq 1 - R - \epsilon$, equivalently $R \leq \sigma - \epsilon$. Both constraints together yield

$$\sigma - \tau - \rho \geq \epsilon + \eta + \frac{H(\tau)}{m}. \tag{3.1}$$

Since $\sigma > \tau$ and since we can take ρ to be smaller than $\sigma - \tau$, the left-hand side of Inequality 3.1 is a constant greater than 0. It is clear from Theorem 3.2.7 that we can choose the folding parameter m as an arbitrarily large constant, thereby also decreasing ϵ. Consequently, the terms ϵ and $\frac{H(\tau)}{m}$ become arbitrarily small and we can choose sufficiently small $\eta, \rho > 0$ such that the inequality is satisfied. Setting $R := \sigma - \epsilon$ we found admissible constants $R, m, \eta, \epsilon > 0$ such that $R \geq \rho + \eta + \tau + H(\tau)/m$. Now let $\mathcal{C}_n$ be a code of length n from the above family that matches these constants. By Theorem 3.2.7 such a code exists for all constants $R, m, \epsilon > 0$. Now let $\mathcal{H}$ be a the family of universal hash functions mapping $\mathbb{F}_2^{Rmn}$ to $\mathbb{F}_2^{\rho mn}$ obtained by Theorem 3.2.6. By Lemma 3.2.5, choosing the universal hash function h randomly from $\mathcal{H}$ yields that $\texttt{LSSS}_{\mathcal{C},h}$ has τ-privacy, except with probability $2^{-\eta mn}$.

Finally, the claim on the efficiency is easily seen to be true by definition of $\texttt{RanSSS}_n$.

□

The above lemma shows that Construction 3.2.1 immediately yields a randomized secret-sharing scheme with liner-time sharing and reconstruction, by simply choosing linear-time computable hash functions and linear-time encodable and decodable codes. Unfortunately, this straightforward approach does not yield the same result for the *non-randomized* secret-sharing scheme of Construction 3.2.1: indeed, the associated sharing function `Share` requires computing the inverse function h^{-1}, which may not be realizable in linear time.

We circumvent the problem as follows: to share an arbitrary secret s, we first compute random shares and associated secret $(z, \mathbf{z})$ with a randomized scheme

RanSSS. We then one-time pad s with z, and "disperse" the padded element $s+z$ (i.e., share $s+z$ among the players by means of a suitable code, with no privacy assumptions). This technique bears resemblance to the construction of Krawczyk [44]; however, while in [44] the information dispersal is applied to reduce the share size, we use this technique to preserve the linear-time computability of the sharing algorithm.

Formally, the scheme is defined as follows:

Construction 3.2.2. *Let $\mathcal{C}'$ be an m'-folded, $\mathbb{F}_q$-linear code of parameters $[n,l,d]$, endowed with encoding and decoding algorithms* $\texttt{Enc} : \mathbb{F}_q^l \to \mathcal{C}'$ *and* $\texttt{Dec} : (\mathbb{F}_q^{m'} \sqcup \{\perp\})^n \to \mathbb{F}_q^l \sqcup \{\perp\}$*; let* RanSSS *be a random secret-sharing scheme given by the random variables $(z \times \mathbf{z}) \in_{\$} \mathbb{F}_q^l \times F^n$, with reconstruction algorithm* RanRec*.*

We then obtain a linear secret-sharing scheme $\texttt{SSS} = \texttt{SSS}_{(\mathcal{C}',\texttt{RanSSS})}$ *with sharing and reconstruction algorithms given as follows:*

- $\texttt{Share} : \mathbb{F}_q^l \to_{\$} (\mathbb{F}_q^{m'} \times F)^n$ *where* $\texttt{Share}(s) := \big(\texttt{Enc}\big(s+z\big), \mathbf{z}\big)$*;*
- $\texttt{Reconstruct} : (\mathbb{F}_q^{m'} \times F \sqcup \{\perp\})^n \to \mathbb{F}_q^l \sqcup \{\perp\}$ *where for* $(\tilde{\mathbf{x}}_i, \tilde{\mathbf{z}}_i)_{i=1,\ldots,n}$*,* Reconstruct *first computes* $\tilde{x} := \texttt{Dec}(\tilde{\mathbf{x}})$ *and* $\tilde{z} := \texttt{RanRec}(\tilde{\mathbf{z}})$*; if $\tilde{x} = \perp$ or $\tilde{z} = \perp$,* Reconstruct *then outputs $\perp$ as well, otherwise it outputs $\tilde{x} - \tilde{z}$.*

Notice that if RanSSS is $\mathbb{F}_q$-linear, then so is SSS. We analyze the privacy and reconstruction thresholds in the following lemma:

Lemma 3.2.9. *In the setting of Construction 3.2.2, assume that* RanSSS *has t-privacy and r-reconstruction, and that its random shares and associated secret can be computed in linear time; Assume further that $\mathcal{C}'$ is linear-time encodable and that it can be decoded from $n-r$ erasures. Then the scheme* SSS *also has t-privacy and r-reconstruction, and can share an arbitrary secret in linear time. Furthermore, if the reconstruction algorithm* RanRec *of* RanSSS *is linear-time computable, and if $\mathcal{C}'$ can be decoded from $n-r$ erasures in linear time, then the reconstruction algorithm* Reconstruct *can also be computed in linear time.*

Proof. Linear-time computability of the sharing function of SSS follows straightforwardly from the linear-time computability of the random shares and secret

of $\texttt{RanSSS}$ and of the encoding function of $\mathcal{C}'$; similarly, the linear-time computability of $\texttt{Reconstruct}$ follows from the properties of $\texttt{RanSSS}$ and $\mathcal{C}'$.

To see that $\texttt{SSS}$ has t-privacy, fix an arbitrary subset $\mathcal{A} \subseteq [n]$ with $|\mathcal{A}| \leq t$. We then need to prove that the distribution of $(\texttt{Enc}(s+z)_{\mathcal{A}}, \mathbf{z}_{\mathcal{A}})$ does not depend on the choice of $s \in \mathbb{F}_q^l$; clearly, it suffices to show that the distribution of $(s+z, \mathbf{z}_{\mathcal{A}})$ does not depend on the choice of $s \in \mathbb{F}_q^l$. Since z is uniformly random, and since $\texttt{RanSSS}$ has t-privacy, we have that z and $\mathbf{z}_{\mathcal{A}}$ are independent, so that the claim holds.

Finally, for r-reconstruction, let $\mathcal{B} \subseteq [n]$ with $|\mathcal{B}| \geq r$; fix $s \in \mathcal{S}_0$, and let $(\tilde{\mathbf{x}}_i, \tilde{\mathbf{z}}_i)_{i=1,\ldots,n}$ where $\tilde{\mathbf{x}}_i = \tilde{\mathbf{z}}_i := \perp$ for any $i \notin \mathcal{B}$, $\tilde{\mathbf{x}}_{\mathcal{B}} := (\texttt{Enc}(s+z)_{\mathcal{B}})$, and $\tilde{\mathbf{z}}_{\mathcal{B}} := \mathbf{z}_{\mathcal{B}}$. We then have that $\texttt{Dec}(\tilde{\mathbf{x}}) = (s+z)$ with probability 1 (since $\mathcal{C}'$ can decode from $n-r$ erasures) and $z = \texttt{RanRec}(\tilde{\mathbf{z}})$ (since $\texttt{RanSSS}$ has r-reconstruction); the claim follows.

□

Finally, plugging the randomized secret sharing scheme $\texttt{RanSSS}_n$ obtained in Lemma 3.2.8 into Construction 3.2.2, we obtain the main result for this section. For the sake of simplicity, as code $\mathcal{C}'$ in Construction 3.2.2 we can choose the same code $\mathcal{C}$ as in Lemma 3.2.8.We thus obtain the following theorem; its proof is straightforward in view of Lemma 3.2.9 above.

Theorem 3.2.10. *For all real constants $0 < \tau < \sigma < 1$ there exists an infinite family of $\mathbb{F}_2$-linear secret scheme $\{\texttt{SSS}_n\}$ with τn-privacy and σn-reconstruction. The shares of $\texttt{SSS}_n$ have size m, where $m > 0$ is an integer constant; $\texttt{SSS}_n$ can share and reconstruct an* arbitrary *secret in linear time. Moreover, such a scheme $\texttt{SSS}_n$ can be constructed randomly with success-probability $1 - 2^{-\eta m n}$ (for some real constant $\eta > 0$ depending on τ, σ and ρ).*

3.3 The Second Application: Robust Secret Sharing via List-Decodable Codes and AMD Codes

The second application of our connection involves *Robust* Secret Sharing, a variant of Secret Sharing with the additional goal to reconstruct the secret even in the presence of incorrect shares; we show how to obtain robust secret-sharing schemes by connecting highly list-decodable codes with AMD codes.

3.3.1 Robust Secret Sharing

We recall here the concept of Robust Secret Sharing; we review the different variations of the topic that have been studied, and we express the goals of our contribution.

In a nutshell, standard Secret Sharing combines two properties, namely privacy and reconstruction, where reconstruction can be seen as recovery from erasures. *Robust* Secret Sharing combines two properties, namely privacy and recovery from *errors:* a secret-sharing scheme is robust if its reconstruction algorithm can recover the secret even if some of the shares are incorrect. This intuition is formalized in the following definition; notice that we speak of a *tampering function* $\texttt{Tamper} : \mathbf{x} \mapsto \mathbf{y}$ over $\mathcal{A} \subseteq [n]$: by this we mean that $\texttt{Tamper}(\mathbf{x})$ only depends on $(\mathbf{x}_i : i \in \mathcal{A})$ and that it acts as the identity on the coordinates outside $\mathcal{A}$.

Definition 3.3.1. *Let* SSS *be a secret-sharing scheme with sharing algorithm* $\texttt{Share} : \mathcal{S}_0 \to_\$ \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ *and reconstruction algorithm* $\texttt{RobustRec} : (\mathcal{S}_1 \sqcup \{\perp\}) \times \cdots \times (\mathcal{S}_n \sqcup \{\perp\}) \to \mathcal{S}_0 \sqcup \{\perp\}$. *Given a positive integer t and a real number $\varepsilon > 0$, we say that* SSS *is* (t, ε)-robust *if the following property holds for any secret $s \in \mathcal{S}_0$, any subset $\mathcal{A} \subseteq [n]$ with $|\mathcal{A}| \leq t$ and any tampering function $\texttt{Tamper} : \mathcal{S}_1 \times \cdots \times \mathcal{S}_n \to \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ over $\mathcal{A}$. If $\mathbf{x} = \texttt{Share}(s)$, then $\texttt{RobustRec}(\texttt{Tamper}(\mathbf{x})) = s$ except with probability at most ε.*

A (t, ε)-robust scheme is typically required to have t-privacy as well. We further define the overhead[2] *of the scheme to be* $\max_{i \geq 1} \log |\mathcal{S}_i| - \log |\mathcal{S}_0|$.

Whether Robust Secret Sharing is possible or not depends on the ratio between the number t of incorrect shares and the total number n of shares: if $t < n/3$, then standard error correction applied to Shamir's scheme (cf. Section 2.5.1) provides robustness for free; on the other hand, if $t \geq n/2$ then it is easily seen that Robust Secret Sharing cannot be achieved.

The interesting range is thus $n/3 \leq t < n/2$; here, robust secret sharing is possible, but we have to allow for a small error probability and additional "checking data" needs to be appended to the actual shares. A typical goal is to optimize the tradeoff between the error probability and the increase in the share size.

Cramer, Damgård and Fehr [17] gave a construction of a robust secret sharing scheme based on so-called *Algebraic Manipulation Detection* (AMD) codes (even though the terms robust secret-sharing and AMD codes were not used

[2] We remark that the notion of overhead is meaningful, a priori, only for threshold schemes, since for ramp schemes it is actually possible to have a share size smaller than the secret size.

there). Roughly speaking, an AMD code enables to detect certain manipulations – namely *algebraic* manipulations – of encoded messages. The robust secret-sharing scheme then simply works by sharing an AMD encoding of the secret (using a linear secret sharing scheme), and the robust reconstruction is done by going through all sets of possibly honest players, reconstruct from their shares, and verify correctness of the reconstructed AMD encoding. By making the AMD codeword large enough, resulting in an overhead in the share size of $O(\kappa + n)$, this procedure finds the correct secret except with probability $2^{-\kappa}$. An obvious downside of this scheme is that the robust reconstruction procedure is not efficient, as there is an exponential number of sets of possibly honest players to be considered.

In [12], based on very different techniques, Cevallos, Fehr, Ostrovsky and Rabani proposed a robust secret-sharing scheme with similar parameters: overhead $O(\kappa+n \log n)$ for an error probability of $2^{-\kappa}$, but which offers an *efficient* robust reconstruction. Both these schemes work for any fraction $t/n < \frac{1}{2}$, and neither becomes significantly better in terms of this error probability versus the size of the checking data if we bound t/n away from $\frac{1}{2}$ by a small constant.

Based on the paradigm of Definition 3.1.1 for building secret-sharing schemes, we construct a new robust secret sharing scheme. Our construction works when t/n is bounded away from $\frac{1}{2}$ by an arbitrary small positive constant.

Our construction can be seen as an efficient variant of the approach from [17]: we secret-share an AMD codeword, but this time choosing the underlying code $\mathcal{C}$ to be one that allows efficient *list decoding*. This means that we can consider the contributed shares as a codeword with errors and apply the list decoding algorithm. This will return a small (i.e., polynomial-sized) list of possible codewords from $\mathcal{C}$, each of these will suggest a possible AMD codeword. Thus, we only have a small number of candidates to check for correctness of the AMD encoding. This not only provides efficiency of the reconstruction (in contrast to the scheme of [17]), but also allows for better parameters, as we will see in Sections 3.3.5 and 3.3.6.

As a final remark, we stress the fact that there are several variants of and concepts related to Robust Secret Sharing. For instance, we assume that a tampering function `Tamper` over $\mathcal{A}$ acts a the identity on the coordinates outside $\mathcal{A}$, and only depends on the coordinates in $\mathcal{A}$; if only the first condition is assumed, then a secret-sharing schemes satisfying Definition 3.3.1 is said to be robust against a *rushing* adversary.

Furthermore, a related concept of special relevance is *Verifiable* Secret Sharing [15], which is a fundamental building block for Multi-Party Computation; loosely speaking, VSS can be seen as an enhanced version of Robust Secret Sharing, where security is guaranteed even if the sharing procedure is not

correctly executed.

Moreover, a scenario somewhat in between robust and verifiable secret sharing has been studied by Cramer et al. in [17]; this variant is called "single-round honest-dealer VSS", and differs from Definition 3.3.1 in that the reconstruction algorithm is not allowed to output an incorrect value, although it can output an error message.

Finally, Ishai et al. explore in [40] the notion of *identifiable* secret sharing, which aims at providing some security even when half or more of the shares are incorrect.

3.3.2 AMD Codes

Algebraic Manipulation Detection codes (AMD codes for short) are the first ingredient of our construction. AMD codes have been introduced by Cramer et al. [21]; as the name suggests, they are used to detect algebraic manipulations on some input data.

Definition 3.3.2 (Algebraic Manipulation Detection codes [21])**.** *Let $\mathcal{S}$ be a non-empty set and $\mathcal{G}$ be an abelian group; given a real number $\delta > 0$, a* δ-secure Algebraic Manipulation Detection *(or* AMD*)* code *with message space $\mathcal{S}$ and cyphertext space $\mathcal{G}$ is given by two functions:*

- $\texttt{Encode} : \mathcal{S} \to_{\$} \mathcal{G}$ *(randomized);*
- $\texttt{Decode} : \mathcal{G} \to \mathcal{S} \sqcup \{\perp\}$ *(deterministic)*

Satisfying the following defining properties:

- *Correctness:* $\texttt{Decode}(\texttt{Encode}(s)) = s$ *with probability 1 for any $s \in \mathcal{S}$;*
- *Robustness:* $p(\texttt{Decode}(\texttt{Encode}(s) + \Delta) \notin \{s, \perp\}) \leq \delta$ *for any $s \in \mathcal{S}$ and any $\Delta \in \mathcal{G}$.*

Remark 3.3.1. Notice that the robustness definition also holds when the error is non-deterministic: to be precise, let $\Delta \in_{\$} \mathcal{G}$ be a random variable independent of $\texttt{Encode}(s)$; then it still holds that

$$p(\texttt{Decode}(\texttt{Encode}(s) + \Delta) \notin \{s, \perp\}) \leq \delta.$$

3.3.3 List-Decodable Codes

The second ingredient of the construction is given by error-correcting codes with high-list decodability properties. The definition of list-decodability is given as follows:

Definition 3.3.3. *Let $\mathcal{C} \subseteq F^n$ be a code of block length n over some alphabet F; then $\mathcal{C}$ is said to be* (t, ℓ)-list decodable *if for any vector $\mathbf{y} \in F^n$ with $d_{\mathrm{H}}(\mathbf{y}, \mathcal{C}) \leq t$, there are at most ℓ codewords $\mathbf{x} \in \mathcal{C}$ with $d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) \leq t$.*

We generally assume that $t < d_{\min}(\mathcal{C})$.

Given a (t, ℓ)-list decodable code $\mathcal{C}$, we denote by $\texttt{ListDecode}$ the function $\mathbf{y} \mapsto \{\mathbf{x} \in \mathcal{C} : d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) \leq t\}$. Notice that the set $L := \{\mathbf{x} \in \mathcal{C} : d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) \leq t\}$ has a natural ordering given by assigning to each $\mathbf{x} \in L$ its *error pattern* $\mathbf{x} - \mathbf{y}$, and then sorting the error patterns according to any order (e.g., a lexicographic one). We will thus view $\texttt{ListDecode}(\mathbf{y})$ as a *list,* i.e. a vector, although we will keep using set notation.

Linear list-decodable codes enjoy the following property:

Remark 3.3.2. Let $\mathcal{C}$ be a (folded) $\mathbb{F}_q$-linear code of length n; denote by $\texttt{ListDecode}$ the function $\mathbf{y} \mapsto (\mathbf{x} \in \mathcal{C} : d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) \leq t)$. Then for any $\mathbf{x} \in \mathcal{C}$ and $\mathbf{e} \in (\mathbb{F}_q^m)^n$ with $w_{\mathrm{H}}(\mathbf{e}) \leq t$, we have that

$$\texttt{ListDecode}(\mathbf{x} + \mathbf{e}) = \mathbf{x} + \texttt{ListDecode}(\mathbf{e}),$$

where $\mathbf{x} + \texttt{ListDecode}(\mathbf{e}) := (\mathbf{x} + \mathbf{x}' : \mathbf{x}' \in \texttt{ListDecode}(\mathbf{e}))$.

Indeed, equality trivially holds as sets since for any $\mathbf{x}' \in \mathcal{C}$, $\mathbf{x} + \mathbf{x}'$ also sits in $\mathcal{C}$ and $d_{\mathrm{H}}(\mathbf{x}', \mathbf{e}) = d_{\mathrm{H}}(\mathbf{x} + \mathbf{x}', \mathbf{x} + \mathbf{e})$.

For what concerns equality as lists, simply notice that for any $\mathbf{x}' \in \texttt{ListDecode}(\mathbf{e})$, the error pattern $\mathbf{x}' - \mathbf{e}$ coincides with the error pattern $(\mathbf{x} + \mathbf{x}') - (\mathbf{x} + \mathbf{e})$, so that the claim holds.

3.3.4 The Construction

We show in this section how to combine AMD codes and highly list-decodable codes to obtain a robust secret-sharing scheme:

Construction 3.3.1. *Let* $\texttt{AMD}$ *be a δ-secure AMD code given by functions* $\texttt{AMD.Encode} : \mathbb{F}_q^k \to_{\$} \mathbb{F}_q^l$ *and* $\texttt{AMD.Decode} : \mathbb{F}_q^l \to \mathbb{F}_q^k \sqcup \{\perp\}$.

Further, let $\texttt{LSSS} = \texttt{LSSS}_{(\mathcal{C},f)}$ *be the (non-randomized) secret-sharing scheme obtained from a code* $\mathcal{C}$ *and a function* f *as in Definition 3.1.1, where* $\mathcal{C}$ *is a (folded)* $\mathbb{F}_q$*-linear,* (t,ℓ)*-list decodable code and* $f : \mathcal{C} \to \mathbb{F}_q^l$ *is a surjective,* $\mathbb{F}_q$*-linear map. We assume that* $t' \geq t$*, where* t' *denotes the privacy threshold of* LSSS*.*

We then obtain a secret-sharing scheme RobustSSS *by defining its sharing and reconstruction algorithms as follows:*

- $\texttt{Share} := \texttt{LSSS.Share} \circ \texttt{AMD.Encode} : \mathbb{F}_q^l \to_{\$} (\mathbb{F}_q^m)^n$*;*
- $\texttt{RobustRec} : \left(\mathbb{F}_q^m \sqcup \{\perp\}\right)^n \to \mathbb{F}_q^l \sqcup \{\perp\}$ *where given* $\mathbf{y} \in (\mathbb{F}_q^m)^n$*,* RobustRec *first computes* $\texttt{ListDecode}(\mathbf{y})$ *and applies* f *to each element of* $\texttt{ListDecode}(\mathbf{y})$*, obtaining a list* L *of "candidates".*

  Then, for any $z \in L$*,* RobustRec *computes* $s' := \texttt{AMD.Decode}(z)$*; if* $s' \neq \perp$*, then* RobustRec *outputs* s'*. If the AMD-decoding of all symbols of* L *produces error symbols* $\perp$*, then* RobustRec *outputs* $\perp$*.*

The main result of this Section shows how the parameters of the underlying components control the robustness of the induced scheme:

Proposition 3.3.1. RobustSSS *is a Secret-Sharing Scheme with* t'*-privacy and* (t,ε)*-robustness for* $\varepsilon := \ell\delta$*.*

Proof. **Reconstruction.** We need to show that $\texttt{RobustRec}(\texttt{Share}(s)) = s$ with probability 1 for any secret $s \in \mathbb{F}_q^l$; let thus $s \in \mathbb{F}_q^l$ be an arbitrarily fixed secret, and let $\mathbf{x} := \texttt{Share}(s) = \texttt{LSSS.Share}(\texttt{AMD.Encode}(s))$. By definition of LSSS, we have that $\mathbf{x} \in \mathcal{C}$ with probability 1, so that $\texttt{ListDecode}(\mathbf{x}) = \{\mathbf{x}\}$ with probability 1; the claim then follows by the reconstruction property of LSSS and the correctness of AMD.

Privacy. Fix a subset $\mathcal{A} \subseteq [n]$ with $1 \leq |\mathcal{A}| \leq t'$; we need to show that the distribution of $\texttt{Share}_{\mathcal{A}}(s)$ does not depend on the choice of the secret $s \in \mathbb{F}_q^l$. Now this holds for LSSS; hence since $\texttt{Share} = \texttt{LSSS.Share} \circ \texttt{AMD.Encode}$, the claim holds.

Robustness. Finally, we need to prove that RobustSSS is (t,ε)-robust: hence fix a secret $s \in \mathbb{F}_q^l$, and let $\mathbf{x} := \texttt{Share}(s)$; furthermore, let $\texttt{Tamper} : (\mathbb{F}_q^m)^n \to (\mathbb{F}_q^m)^n$ be a tampering function over a set $\mathcal{A} \subseteq [n]$ with $|\mathcal{A}| \leq t$. Let $\mathbf{y} := \texttt{Tamper}(\mathbf{x})$. We need to prove the following:

$$\texttt{RobustRec}(\mathbf{y}) = s, \text{ except with probability at most } \varepsilon.$$

Now recall that RobustRec works as follows: first, a list of candidates L is computed by applying f to each element of $\texttt{ListDecode}(\mathbf{y})$; then the elements of L are AMD-decoded, as as soon as a decoding succeeds, RobustRec outputs that AMD-decoded element.

As a first step, we show that $\texttt{AMD.Encode}(s) \in L$ with probability 1: indeed, let $z := \texttt{AMD.Encode}(s)$, so that $\mathbf{x} \in_{\$} f^{-1}(z)$. By definition of tampering function, $d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) \leq t$ with probability 1, so that $\mathbf{x} \in \texttt{ListDecode}(\mathbf{y})$, and, in turn, $z = f(\mathbf{x}) \in L$ with probability 1. Hence the claim holds.

Now since $\texttt{AMD.Encode}(s)$ belongs to the list of candidates L with probability 1, we have that

$$\texttt{RobustRec}(\mathbf{y}) \neq s \text{ only if there exists } z' \in L \text{ such that } \texttt{AMD.Decode}(z') \notin \{s, \perp\}.$$

We thus need to upper bound the probability that such a z' exists. First notice that we can assume that L is a list of length ℓ (up to adding $\perp$ symbols to it), i.e. we can write $L = (z^{(1)}, \ldots, z^{(\ell)})$. Hence we have that

$$p\big(\exists\, z' \in L \text{ s. t. } \texttt{AMD.Decode}(z') \notin \{s, \perp\}\big) \leq \sum_{j=1}^{\ell} p\left(\texttt{AMD.Decode}(z^{(j)}) \notin \{s, \perp\}\right).$$

Hence to conclude it suffices to show that any element of the sum is upper-bounded by δ; in view of Remark 2.3.3, this can be showed by proving that $z^{(j)} = z + e^{(j)}$ for some error $e^{(j)}$ independent of $\texttt{AMD.Encode}(s)$.

To this end, notice that we can write $\mathbf{y} = \mathbf{x} + \mathbf{e}$ where $w_{\mathrm{H}}(\mathbf{e}) \leq t$ with probability 1, and where $\mathbf{e}$ is a function of $\mathbf{x}_{\mathcal{A}}$. Now thanks to Remark 3.3.2, we have that $\texttt{ListDecode}_j(\mathbf{y}) = \mathbf{x} + \texttt{ListDecode}_j(\mathbf{e})$, which means that $z^{(j)} = z + f(\texttt{ListDecode}_j(\mathbf{e}))$ by linearity of f. Hence to conclude, it suffices to show that $f(\texttt{ListDecode}_j(\mathbf{e}))$ is independent of $\texttt{AMD.Encode}(s)$; now by t'-privacy, $\mathbf{x}_{\mathcal{A}}$ is independent of $z = \texttt{AMD.Encode}(s)$ since $|\mathcal{A}| \leq t'$, so that $\mathbf{e}$ and, in turn, $f(\texttt{ListDecode}_j(\mathbf{e}))$ are also independent of $\texttt{AMD.Encode}(s)$. This concludes the proof.

□

The next subsections show two implementations of this construction.

3.3.5 A Shamir-based Scheme

Let $n, t < n/2$ be two fixed positive integers. The two ingredients of the scheme are the following.

The code $\mathtt{AMD}_{q,d}$ due to Cramer et al. [21] (which has optimal size of tags for a given security parameter), given by the following encoding function:

$$\begin{array}{rccl} \mathtt{AMD.Encode} : \mathbb{F}_q^d & \to_{\$} & \mathbb{F}_q^d \times \mathbb{F}_q \times \mathbb{F}_q & \\ \mathbf{s} & \mapsto & \left(\mathbf{s}, r, r^{d+2} + \sum_{i=1}^{d} \mathbf{s}_i r^i\right) & \text{for uniform } r \in_{\$} \mathbb{F}_q \end{array}$$

where d is an integer and $\mathbb{F}_q$ a finite field of cardinality q, with $\mathrm{char}(\mathbb{F}_q) \nmid d+2$, and with the obvious decoding function $\mathtt{AMD.Decode}$ (that checks that the three coordinate satisfy the above relationship). We have that $\mathtt{AMD}_{q,d}$ is a δ-secure AMD code for $\delta \leq (d+1)/q$, which means that the following lemma holds:

Lemma 3.3.2 ([21])**.** *For any prime power q and any integer d such that $\mathrm{char}(\mathbb{F}_q) \nmid d+2$ there exists a $(d+1)/q$-secure AMD code with message space $\mathbb{F}_q^d$ and cyphertext space $\mathbb{F}_q^d \times \mathbb{F}_q \times \mathbb{F}_q$.*

As a code-based secret-sharing scheme, we use a variant of Shamir's scheme, namely we instantiate Massey's blueprint with *folded* Reed-Solomon codes:

Definition 3.3.4 (**Guruswami and Rudra** [37])**.** *Given a finite field $\mathbb{F}_q$ with q elements, a generator γ of the multiplicative group $\mathbb{F}_q^\times$ and positive integers m, n, $k \leq n$ with $q > mn$, the* Folded Reed-Solomon Code $\mathtt{FRS}$ *of length n over $\mathbb{F}_q^m$ is given as follows.*

Let $\tilde{\mathcal{C}}$ be the following (standard) Reed-Solomon code:

$$\tilde{\mathcal{C}} := \left\{ \left(\varphi(1), \varphi(\gamma), \varphi(\gamma^2), \cdots, \varphi(\gamma^{mn})\right) \,:\, \varphi(x) \in \mathbb{F}_q[x], \deg \varphi \leq k-1 \right\} \leq \mathbb{F}_q^{mn}$$

Then $\mathtt{FRS}$ is simply $\tilde{\mathcal{C}}$ viewed as a code over $F := \mathbb{F}_q^m$, i.e., it is obtained from $\tilde{\mathcal{C}}$ by bundling together blocks of m consecutive symbols.

Notice that $\mathtt{FRS}$ thus has rate R and minimum distance d where

$$R := \frac{\log_{q^m}(|\mathtt{FRS}|)}{n} = \frac{k}{mn}, \quad d \geq n - \frac{k-1}{m}$$

Now given a folded Reed-Solomon code $\mathtt{FRS}$ with $q > m(n+1)$, we define the function $f : \mathtt{FRS} \to F$ to be

$$\begin{array}{rcl} f : \mathtt{FRS} & \to & F \\ \mathbf{x} = (\varphi(1), \cdots, \varphi(\gamma^{mn})) & \mapsto & \big(\varphi(\gamma^{mn+1}), \cdots, \varphi(\gamma^{mn+m})\big) \end{array}$$

It is immediately seen that the scheme $\mathtt{LSSS}_{\mathtt{FRS},f}$ obtained this way enjoys $(Rn-1)$-privacy.

Now the key point of folded RS codes is that they are highly list-decodable, as shown in the following theorem:

Theorem 3.3.3 ([37])**.** *For any real numbers $\epsilon > 0$ and $0 < R < 1$, and for any large enough integer $m > 0$ (depending on ε and R) and any integer $n > 0$, there is folded Reed-Solomon code $\mathtt{FRS}_n$ with folding parameter m and field size $q = O(mn)$, which have rate at least R and that are $((1-R-\epsilon)n, \mathrm{poly}(q))$-list decodable.*

Furthermore, such list decoding can be realized by an algorithm with running time $\mathrm{poly}(n, m)$.

We can now connect this code-based secret-sharing scheme and the above AMD codes as in Construction 3.3.1 to get the following result:

Proposition 3.3.4. *Let κ and M be positive integers, and let $0 < \tau < 1/2$. We then have that for any large enough integer n, there exists an n-player secret-sharing scheme $\mathtt{RobustSSS}_n$ such that:*

- *the secret space has cardinality at least 2^M;*
- *the scheme enjoys τn-privacy and $(\tau n, 2^{-\kappa})$-robustness.*

The overhead of the scheme is in $O(\kappa + \log M)$; furthermore, both sharing and robust reconstruction can be implemented in time polynomial in n and κ.

Proof. Since $\tau < 1/2$, for large enough n there exists a constant ε such that $\tau + 1/n < 1 - \tau - \varepsilon$. Let thus R be any constant with $\tau + 1/n < R < 1 - \tau - \varepsilon$, and let $\mathtt{FRS}_n$ be the code obtained from Theorem 3.3.3; then $\mathtt{FRS}_n$ has rate at least $\tau + 1/n$ and is $(\tau n, \ell)$-list decodable with $\ell = \mathrm{poly}(q)$.

Let $\mathtt{LSSS}_n = \mathtt{LSSS}_{(\mathtt{FRS}_n, f_n)}\}$ be the induced code-based secret-sharing scheme; by the above discussion, each $\mathtt{LSSS}_n$ enjoys $(Rn-1) = \tau n$-privacy.

Now write $\mathbb{F}_q^m \simeq \mathbb{F}_q^{m'\kappa'} \times \mathbb{F}_q^{\kappa'} \times \mathbb{F}_q^{\kappa'}$ where $m' = m/\kappa' - 2$ for a parameter κ' to be determined; we then apply Construction 3.3.1 and connect $\mathtt{LSSS}_n$ with the AMD code $\mathtt{AMD}_{(q^{\kappa'}, L)}$ given by $\mathtt{AMD.Encode} : \mathbb{F}_q^{m'\kappa'} \to \mathbb{F}_q^{m'\kappa'} \times \mathbb{F}_q^{\kappa'} \times \mathbb{F}_q^{\kappa'}$ from Lemma 3.3.2.

Now the resulting scheme $\texttt{RobustSSS}_n$ still enjoys τn-privacy and is $(\tau n, \varepsilon)$-robust for $\varepsilon \leq \ell \cdot ((m'+1)/q^{\kappa'})$; furthermore, its secret space has cardinality $q^{m'\kappa'}$. Hence to obtain the claimed security and secret space size we need to set

$$\begin{cases} \ell \cdot (m'+1)/q^{\kappa'} \leq 2^{-\kappa} \\ m'\kappa' \geq M \log_q 2 \end{cases}$$

Now these equations are satisfied by setting

$$\begin{cases} \kappa' = \left\lceil \log_q \ell + \log_q(M \log_q 2) + \kappa \log_q 2 \right\rceil \\ m' = \left\lceil M \log_q 2/\kappa' \right\rceil \end{cases}$$

The overhead of the scheme is thus equal to

$$\begin{aligned} m \log q - M &= (m'\kappa' + 2\kappa') \log q - M \\ &\leq (M \log_q 2 + 3\kappa') \log q - M \\ &= O(\kappa') \\ &= O(\log M + \kappa). \end{aligned}$$

This concludes the proof.

□

3.3.6 With Universal Hash Functions

We present here a second way to implement Construction 3.3.1, where we keep the AMD codes of Lemma 3.3.2, but use linear secret-sharing schemes based on universal hash functions and highly list-decodable codes.

We first define the codes that will be used in our construction:

Theorem 3.3.5 (**List-decodability of Folded Algebraic Geometric Codes** [38])**.** *For any real numbers $0 < R < 1$ and $\epsilon > 0$, and for any large enough integer $m > 0$ (depending on R and ϵ) there exist a constant prime power q and an infinite family of m-folded $\mathbb{F}_q$-linear codes $\{\mathcal{C}_n\}$, such that the rate of $\mathcal{C}_n$ is R, and $\mathcal{C}_n$ is efficiently $(\tau n, \ell)$-list decodable with $\tau = 1 - R - \epsilon$ and $\ell = poly(n)$.*

Furthermore, such list decoding can be realized by an algorithm with running time $poly(n, m)$.

We then obtain following result:

Proposition 3.3.6. *For any real number $0 < \tau < 1/2$ and integer $\kappa > 0$ there exists an infinite family $\{\mathtt{SSS}_n\}$ of efficient n-player secret-sharing schemes with (τn)-privacy and $(\tau n, 2^{-\kappa})$-robustness. The secret has size $\Omega(n+\kappa)$ and the shares have size $O(1+\kappa/n)$ (each); this means that the share size is* constant *in n.*

Proof. Let $\{\mathcal{C}_n\}$ be the family of codes from Theorem 3.3.5, with R and ε to be determined; each $\mathcal{C}_n$ has parameters $[n, Rmn, d]_q$ for constant m and q, and is $(1-R-\varepsilon, \ell)$-list decodable for $\ell = \mathrm{poly}(n)$.

Now let $\mathcal{H}$ be a family of linear universal hash functions $\mathbb{F}_q^{Rmn} \to \mathbb{F}_q^{\rho mn}$ for a parameter $\rho > 0$ to be determined; notice that such a family exists thanks to Theorem 3.2.6.

By connecting C_n with a random $h \in \mathcal{H}$ as in Definition 3.1.1, we obtain a linear secret-sharing scheme $\mathtt{LSSS}_n$; for $\eta > 0$, we have that the scheme enjoys τn-privacy (with positive probability over the choice of h) as long as $R \geq \rho + \eta + \tau + H(\tau)/(m \cdot \log(q))$.

By writing $\mathbb{F}_q^{\rho mn} \simeq \mathbb{F}_q^{\rho' mn\kappa'} \times \mathbb{F}_q^{\kappa'} \times \mathbb{F}_q^{\kappa'}$, we can connect $\mathtt{LSSS}_n$ to the AMD code given by encoding function $\mathtt{AMD.Encode} : \mathbb{F}_q^{\rho' mn\kappa'} \to \mathbb{F}_q^{\rho' mn\kappa'} \times \mathbb{F}_q^{\kappa'} \times \mathbb{F}_q^{\kappa'}$ as in Lemma 3.3.2.

The resulting secret-sharing scheme $\mathtt{SSS}_n$ has the same privacy as $\mathtt{LSSS}_n$ and is $((1-R-\varepsilon)n\,, \ell(\rho' mn+1)/q^{\kappa'})$-robust; hence to get τn-privacy and $(\tau n, 2^{-\kappa})$-robustness we need to set

$$\begin{cases} R \geq \rho + \eta + \tau + H(\tau)/(m \cdot \log(q)) \\ 1 - R - \varepsilon \geq \tau \\ \ell \cdot \frac{\rho' mn}{q^{\kappa'}} \leq 2^{-\kappa} \end{cases}$$

i.e.

$$\begin{cases} 1 - \tau - \varepsilon \geq R \geq \rho + \eta + \tau + H(\tau)/(m \cdot \log(q)) \\ \kappa' \geq \kappa \log_q 2 + \log_q \ell + \log_q(\rho' mn + 1) \end{cases}$$

The first row has a solution R since $1 - 2\tau \geq \varepsilon + \rho + \eta + H(\tau)/(m \cdot \log(q))$ (for small enough ε, ρ and η, and up to enlarging m), while the second inequality is satisfied by setting

$$\kappa' := \left\lceil \kappa \log_q 2 + \log_q \ell + \log_q(\rho mn + 1) \right\rceil .$$

Moreover, the size of each share space is equal to $m \log q$; now we have that

$$m \log q = \Theta(m) = \Theta(1 + \kappa'/n) = \Theta(1 + \kappa/n).$$

Finally, the secret has size $\rho' mn \log q = n\Theta(m \log q)$; this concludes the proof.

□

Chapter 4

New Protocols for Secure Multi-Round Communication

We discuss in this chapter some generalizations of (Robust) Secret Sharing; we first discuss the application of Secret Sharing and Robust Secret Sharing to a scenario of one-way secure communication over parallel channels, which will form the common framework for the present and previous chapters.

Section 4.1 shows a generalization of this secure-communication setting, where messages can be transmitted in both ways; this model has become known as Secure Message Transmission. Based on the article [66], co-authored by Gilles Zémor, we use coding-theoretic techniques to construct a two-round protocol with perfect security, and show how our protocol has lower complexity and a more simple core structure compared to previous work.

In Section 4.2, we consider a more general scenario where an attacker has increased eavesdropping and tampering powers. We show how this scenario models the communication in a Secure Network Coding scenario; based on the articles [66] and [67] with Gilles Zémor, we then propose two secure protocols in this setting. The first protocol works in two rounds, and is an adaptation of the "vanilla" protocol of Section 4.1; the second one works in three rounds, and can also provide security in a multicast setting, i.e. in the presence of several receivers.

4.1 Perfectly Secure Message Transmission

We start by discussing in more details a direct application of Secret Sharing sketched in Chapter 2, Section 2.5. Suppose that two users Alice and Bob are connected by n parallel channels, and that an attacker Eve controls t of this channels (we will explain the meaning of "control" in the next few lines). Assume that Alice wishes to communicate a secret value s that she holds, and that she is only allowed to send a single symbol over each channel.

As a first adversarial model, assume that Eve acquires the symbols transmitted over the channels under her control, and assume that she is able to *erase* them, meaning that Bob will receive an error message $\perp$ instead of the symbol. It is then readily seen that a secret-sharing scheme can be used to achieve security in this setting: Alice can compute a share vector $(s_1, \ldots, s_n)$ from the secret s and send each share to Bob over a distinct channel; if the scheme has t-privacy and $(n-t)$-reconstruction, we then achieve *privacy* (meaning that Eve obtains no information at all on the secret) and *reliability,* meaning that Bob is able to compute the secret message s from the received data, in spite of the symbols erased by Eve.

Similarly, if we assume that Eve can eavesdrop and *tamper*, meaning that she can replace any symbol transmitted over a channel that she controls with an arbitrary symbol from the same alphabet, then a (t, ε)-robust secret sharing scheme can be used in the same way to achieve privacy and reliability (although reliability is only guaranteed with probability $1 - \varepsilon$).

The model that has become known as Secure Message Transmission, introduced by Dolev et al. in [27], is a further generalization of the above setting, where we assume that the channels can also be used by Bob to convey data to Alice, and where Alice and Bob are free to transmit several symbols over each channel. The adversary Eve is able to eavesdrop and tamper on the channels under her control, and the goal is again for Alice to communicate a secret message to Bob with privacy and reliability. We stress the fact that, traditionally, perfect privacy is required, while reliability can either have a small error probability or be perfect. In the latter case, we speak of *Perfectly* Secure Message Transmission or *PSMT,* which we focus on in this section.

We now give a formal definition of a PSMT protocol. Notice that in contrast with Secret Sharing, which we defined in "abstract" terms, PSMT is here formalized in terms of this secure-communication scenario, as an abstract definition would be quite cumbersome. We stress the fact, however, that in view of our mathematical formalization of interactive protocols (cf. Chapter 2, Section 2.4), the definition we give here is mathematically rigorous.

Definition 4.1.1 (PSMT). *A* Perfectly Secure Message Transmission *proto-*

col is specified by positive integers $n, t \leq n$ and a finite, non-empty set F. The protocol involves two users, Alice and Bob, that are connected by n "channels", and an adversary Eve that "controls" t of these channels.

Alice and Bob can send elements of F to each other over each channel; whenever a symbol is transmitted over a channel under her control, Eve acquires that symbol and is allowed to replace it with a symbol from F of her choice.

We assume that Alice holds secret values $s^{(1)}, \ldots, s^{(\ell)} \in F$; a PSMT protocol specifies which symbols Alice and Bob should communicate to each other, and is deemed secure *if the following two conditions hold for any choice of $s^{(1)}, \ldots, s^{(\ell)} \in F$:*

- Privacy: *the data eavesdropped by Eve has a distribution which does not depend on the secrets $s^{(1)}, \ldots, s^{(\ell)}$.*
- Reliability: *Bob is always able to recover $s^{(1)}, \ldots, s^{(\ell)}$ from the data he sent and received.*

We contribute to Perfectly Secure Message Transmission by constructing a two-round protocol with better efficiency compared to previous work and a more intuitive and powerful core structure.

The following subsections are organized as follows. In Section 4.1.1 we give some more background on PSMT, discuss what are the state-of-the-art protocols in the field and how our protocols compare to them. In Section 4.1.2, we sketch the core structure of our protocols, before starting to discuss the details; Section 4.1.3 presents the basic communication tools that are going to be needed, and Section 4.1.4 presents another fundamental concept for our protocols, the pseudo-basis or syndrome-spanning subset. In Section 4.1.5 we present a first protocol, prove its security and discuss its efficiency; finally, in Section 4.1.6 we show how to improve this protocol to reach better efficiency.

4.1.1 An Overview of PSMT

Two factors influence whether PSMT is possible and how difficult it is to achieve, namely the number t of channels corrupted and controlled by Eve, and the number r of transmission rounds, where a transmission round is a phase involving only one-way communication (either from Alice to Bob, or from Bob to Alice).

It was shown in Dolev et al.'s original paper [27] that for $r = 1$, i.e. when communication is only allowed from Alice to Bob, PSMT is possible if and

only if $n \geq 3t + 1$; notice that in this setting, PSMT is essentially equivalent to Robust Secret Sharing, which indeed is easily seen to be possible with no error probability only if $n \geq 3t + 1$ (cf. Chapter 3, Section 3.3.1).

It was also shown in [27] that for $r \geq 2$, i.e. when communication can be performed in two or more rounds, PSMT is possible if and only if $n \geq 2t + 1$, meaning that two-ways communication strengthen security, although only a very inefficient way to do this was proposed. A number of subsequent efforts were made to improve PSMT protocols, notably in the most difficult case, namely for $r = 2$ rounds and when $n = 2t + 1$. The following two quantities, called *communication complexity* and *transmission rate*, were introduced and give a good measure of the efficiency of a PSMT protocol. They are defined as follows:

$$\text{Communication complexity} := \quad \text{total number of bits transmitted to communicate a single-bit secret,}$$

$$\text{Transmission rate} := \frac{\text{total number of bits transmitted}}{\text{bit-size of the secret}}.$$

Focusing exclusively on the case $n = 2t + 1$, Dolev et al. [27] presented a PSMT protocol for $r = 3$ with transmission rate $O\left(n^5\right)$; for $r = 2$ a protocol was presented with non-polynomial rate.

Sayeed and Abu-Amara [57] were the first to propose a two-round protocol with a polynomial transmission rate of $O\left(n^3\right)$. They also achieved communication complexity of $O\left(n^3 \log n\right)$. Further work by Agarwal et al. [1] improved the transmission rate to $O(n)$ meeting, up to a multiplicative constant, the lower bound of [68]. However, this involved exponential-time algorithms for the participants in the protocol. The current state-of-the art protocol is due to Kurosawa and Suzuki [46, 45]; it achieves $O(n)$ transmission rate with a polynomial-time effort from the participants. All these protocols do not do better than $O\left(n^3 \log n\right)$ for the communication complexity.

We contribute to this topic in the following ways. We present a constructive protocol for which only straightforward computations are required of the participants, that achieves the improved communication complexity of $O\left(n^2 \log n\right)$. In passing, we give an affirmative answer to an open problem of Kurosawa and Suzuki (at the end of their paper [45]) that asks whether it is possible to achieve the optimal transmission rate $O(n)$ for a secret of size less than $O(n^2 \log n)$ bits. We do this for a secret of $O(n \log n)$ bits.

Moreover, our solution is conceptually simpler than previous protocols: two-round PSMT involves Bob initiating the protocol by first sending an array of symbols (x_{ij}) over the n parallel channels, where the first index i means

that symbol x_{ij} is sent over the i-th channel. All previous proposals relied on arrays (x_{ij}) with a lot of structure, with linear relations between symbols that run both along horizontal (constant j) and vertical (constant i) lines. In contrast, we work with an array (x_{ij}) consisting of completely independent rows $\mathbf{x}^{(j)} = (x_{1j}, x_{2j}, \ldots, x_{nj})$ that are simply randomly chosen words of a given Reed-Solomon code. In its simplest, non-optimized form, the PSMT protocol we present only involves simple syndrome computations from Alice, and one-time padding the secrets it wishes to transfer with the image of linear forms applied to corrupted versions of the codewords $\mathbf{x}^{(j)}$ it has received from Bob.

In its optimized form, the protocol achieves a transmission rate $5n + o(n)$, compared to the previous record of $6n + o(n)$ of [35] obtained by painstakingly optimizing the $25n + o(n)$ transmission rate of [45].

4.1.2 An Overview of our Protocol

The protocol is specified by the number $n = 2t + 1$ of channels between Alice and Bob and the number ℓ of secret messages to be communicated; we assume that the messages lie in a finite field $\mathbb{F}_q$. The basic communication tool is given by a randomized secret-sharing scheme built via Massey's blueprint; according to the formulation of Chapter 3, this is given by a pair $(\mathcal{C}, f)$, where $\mathcal{C}$ is a linear block code of parameters $[n, t+1, t+1]$ over $\mathbb{F}_q$ and $f : \mathcal{C} \to \mathbb{F}_q$ is a surjective, $\mathbb{F}_q$-linear map. $\mathcal{C}$ and f have the property that the knowledge of t symbols of any of the codewords $\mathbf{x} \in \mathcal{C}$ leaves $f(\mathbf{x})$ completely undetermined; $\mathcal{C}$ can be a Reed-Solomon code.

Since we require at most two rounds of communication, Bob starts the procedure; he chooses a certain number of random and independent codewords $\mathbf{x} \in \mathcal{C}$, and communicates them to Alice by sending the i-th symbol of each codeword over the i-th channel. This is a first major difference from previous papers, notably [45], where codewords are communicated in a more complicated "horizontal-and-vertical" fashion.

As a result of this first round of communication, Alice receives a corrupted version $\mathbf{y} = \mathbf{x} + \mathbf{e}$ for each codeword $\mathbf{x}$ sent by Bob. As in previous PSMT protocols, Alice then proceeds by broadcast, meaning every symbol she physically sends to Bob, she sends n times, once over every channel i. In this way privacy is sacrificed, since Eve can read everything Alice sends, but reliability is ensured, since Bob recovers every transmitted symbol by majority decoding.

A secret message consisting of a single symbol $s \in \mathbb{F}_q$ is encoded by Alice as $s + f(\mathbf{y})$ for some received vector $\mathbf{y}$. In other words, s is one-time padded with the quantity $f(\mathbf{y})$ and this is broadcast to Bob. Notice that at this point,

revealing $s+f(\mathbf{y})$ to Eve gives her zero information on s; this is because she can have intercepted at most t symbols of the codeword $\mathbf{x}$: therefore the element $f(\mathbf{x})$ is completely unknown to her by the above property of $\mathcal{C}$ and f (i.e., the privacy of Massey's scheme), and the mask $f(\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{e})$ is unknown to her as well.

Now broadcasting the quantity $s + f(\mathbf{y})$ is not enough by itself to convey the secret s to Bob, because Bob also does not have enough information to recover the mask $f(\mathbf{y})$. To make the protocol work, Alice needs to give Bob extra information that tells Eve nothing she doesn't already know.

This extra information comes in two parts. The first part is simply the syndrome $\sigma(\mathbf{y}) = \mathbf{H}\mathbf{y}^T$ of $\mathbf{y}$, where $\mathbf{H}$ is a parity-check matrix of $\mathcal{C}$; notice that this data is indeed useless to Eve, who already knows it given that $\mathbf{H}\mathbf{y}^T = \mathbf{H}\mathbf{x}^T + \mathbf{H}\mathbf{e}^T = \mathbf{H}\mathbf{e}^T$ where $\mathbf{e}$ is chosen by herself.

The second part makes use of the fact that during the first phase, Bob has not sent a single codeword $\mathbf{x}$ to Alice, but a batch of codewords $\mathcal{X}$ and Alice has received a set $\mathcal{Y}$ of vectors made up of the corrupted versions $\mathbf{y} = \mathbf{x} + \mathbf{e}$ of the codewords $\mathbf{x}$. Alice will sacrifice a chosen subset of these vectors $\mathbf{y}$ and reveal them to Bob and Eve by broadcast. Note that this does not yield any information on the unrevealed vectors $\mathbf{y}$ since Bob has chosen the codewords $\mathbf{x}$ of $\mathcal{X}$ randomly and independently. At this point we apply an idea that originates in [45]: the chosen revealed subset of $\mathcal{Y}$ is called in [45] a *pseudo-basis* of $\mathcal{Y}$. To compute a pseudo-basis of $\mathcal{Y}$, Alice simply computes all syndromes $\sigma(\mathbf{y})$ for $\mathbf{y} \in \mathcal{Y}$, and chooses a minimal subset of $\mathcal{Y}$ whose syndromes generate linearly all syndromes $\sigma(\mathbf{y})$ for $\mathbf{y} \in \mathcal{Y}$. A pseudo-basis of $\mathcal{Y}$ could alternatively be called a *syndrome-spanning* subset of $\mathcal{Y}$. Now elementary coding-theory arguments imply that a pseudo-basis of $\mathcal{Y}$ allows Bob, *for any non-revealed* $\mathbf{y} = \mathbf{x}+\mathbf{e}$, to recover the error $\mathbf{e}$ from the syndrome $\sigma(\mathbf{y}) = \sigma(\mathbf{e})$ (Proposition 4.1.4).

We shall present optimized variants that achieve the communication complexity and transmission rate claimed in Section 4.1.1. Our final protocol involves two additional ideas; the first involves a more efficient broadcasting scheme than pure repetition: this idea was also used by Kurosawa and Suzuki. The second idea is new and involves using a decoding algorithm for the code $\mathcal{C}$.

4.1.3 Private and Reliable Communication Tools

We present here two communication tools that are used by Alice and Bob in our protocol: a private communication tool and a broadcast procedure.

The private communication tool is a randomized secret-sharing scheme constructing via Massey's paradigm (cf. Chapter 2, Section 2.6). We define the

scheme and analyze it in the style of Definition 3.1.1, and show how it can be used by Alice and Bob for private communication.

Lemma 4.1.1. *For any positive integers n and $1 \leq t \leq n$ and any prime power $q > n$ there exist an $\mathbb{F}_q$-linear MDS code $\mathcal{C}$ of parameters $[n, t+1, n-t]$ and a linear function $f : \mathcal{C} \to \mathbb{F}_q$ such that the following holds. If $\mathbf{x} \in_\$ \mathcal{C}$ is uniformly distributed, then $f(\mathbf{x})$ is uniformly distributed over $\mathbb{F}_q$ and is independent of $\mathbf{x}_\mathcal{A}$ for any $\mathcal{A} \subseteq [n]$ with $|\mathcal{A}| = t$.*

This means that if a random $\mathbf{x} \in \mathcal{C}$ is transmitted from Alice to Bob or vice versa, then $f(\mathbf{x})$ is uniformly distributed and independent of Eve's eavesdropped data.

Proof. Let $\mathcal{C}'$ be an MDS code of parameters $[n+1, t+1, n-t+1]$; notice that such a code exists for any n and $t \leq n$ (cf. Chapter 2, Section 2.2.2). Let $\mathcal{C}$ be the code obtained from $\mathcal{C}'$ by puncturing at its first coordinate, i.e.

$$\mathcal{C} := \left\{ \mathbf{x} \in \mathbb{F}_q^n : \exists\, x \in \mathbb{F}_q \text{ with } (x, \mathbf{x}) \in \mathcal{C}' \right\}$$

The minimum distance of $\mathcal{C}$ is at most one less than that of $\mathcal{C}'$, hence $\mathcal{C}$ is MDS of parameters $[n, t+1, n-t]$ as requested. The map $f : \mathcal{C} \to \mathbb{F}_q$ is given by $\mathbf{x} \mapsto x : (x, \mathbf{x}) \in \mathcal{C}'$; notice that f can be explicitly constructed by setting $f(\mathbf{x}) = -1/\alpha \cdot \mathbf{h}\mathbf{x}^T$, where $\alpha \in \mathbb{F}_q \setminus \{0\}$ and $\mathbf{h} \in \mathbb{F}_q^n$ are such that $(\alpha, \mathbf{h})$ is a row of a parity-check matrix of $\mathcal{C}'$.

Now fix an arbitrary subset $\mathcal{A} \subseteq [n]$ with $|\mathcal{A}| \leq t$; we need to prove that $f(\mathbf{x})$ and $\mathbf{x}_\mathcal{A}$ are independent for $\mathbf{x} \in_\$ \mathcal{C}$ uniformly distributed. Thanks to Proposition 3.1.2, it suffices to show that for any $x \in \mathbb{F}_q$ there exists $\mathbf{y} = (x, \mathbf{x}) \in \mathcal{C}'$ with $\mathbf{y}_\mathcal{A} = \mathbf{0}$.

Now since $\{0\} \cup \mathcal{A}$ is an information set for $\mathcal{C}'$ (where we count coordinates in $\mathbb{F}_q^{n+1}$ from 0 to n), the claim holds. This concludes the proof.

□

Furthermore, under the assumption that $n = 2t + 1$, Alice and Bob can communicate with perfect reliability via the following straightforward method:

Lemma 4.1.2 (Broadcast). *Alice and Bob can* broadcast *any symbol $x \in \mathbb{F}_q$ by sending it over all the channels. Since Eve controls $t < n/2$ of them, the receiver will be able to recover x with a simple majority choice. Broadcast thus guarantees reliability by sacrificing privacy.*

4.1.4 Pseudo-Bases or Syndrome-Spanning Subsets

The second fundamental building block of our paper is the notion of *pseudo-basis*, introduced by Kurosawa and Suzuki [45]. The concept stems from the following intuition: assume that Bob communicates a single codeword $\mathbf{x}$ of an MDS code $\mathcal{C}$ to Alice by sending each of its n symbols over the corresponding channel. Eve intercepts t of these symbols, thus $\mathcal{C}$ must have dimension at least $t+1$ if we want to prevent her from learning $\mathbf{x}$; but this means that the minimum distance of $\mathcal{C}$ cannot exceed $n+1-(t+1) = t+1$, which is not enough for Alice to correct an arbitrary pattern of up to t errors that Eve can introduce.

If, however, we repeat the process for several different $\mathbf{x}^{(i)}$, then Alice and Bob have an important advantage: they know that all the errors introduced by Eve *always lie in the same subset of t coordinates*. Kurosawa and Suzuki propose the following strategy to exploit this knowledge: Alice can compute a pseudo-basis (a subset with special properties) of the received vectors; she can then transmit it to Bob, who will use this special structure of the errors to recover them from their syndromes.

The key is the following simple lemma:

Lemma 4.1.3. *Let $\mathcal{C}$ be a linear code of parameters $[n,k,d]_q$, and let $\mathbf{H}$ be a parity-check matrix of $\mathcal{C}$; let E be a linear subspace of vectors of $\mathbb{F}_q^n$ such that the Hamming weight $w_{\mathrm{H}}(\mathbf{e})$ of $\mathbf{e}$ satisfies $w_{\mathrm{H}}(\mathbf{e}) < d$ for any $\mathbf{e} \in E$.*

We then have that the following map is injective:

$$\begin{aligned} \sigma_{|E} : E &\to \mathbb{F}_q^{n-k} \\ \mathbf{e} &\mapsto \mathbf{H}\mathbf{e}^T \end{aligned}$$

Proof. Simply notice that $\ker\left(\sigma_{|E}\right) = \{\mathbf{0}\}$: indeed, $\ker\left(\sigma_{|E}\right) \subseteq \mathcal{C}$; but by assumption all elements of E have weight smaller than d, so that $\ker\left(\sigma_{|E}\right) = \{\mathbf{0}\}$.

□

We can now introduce the concept of pseudo-basis; for the rest of this section, we assume that a linear code $\mathcal{C}$ of parameters $[n,k,d]_q$ has been chosen, together with a parity-check matrix $\mathbf{H}$ and associated syndrome map σ.

Definition 4.1.2 (Pseudo-Basis [45]). *Let $\mathcal{Y}$ be a set of vectors of $\mathbb{F}_q^n$; a* pseudo-basis *of $\mathcal{Y}$ is a subset $\mathcal{W} \subseteq \mathcal{Y}$ such that $\sigma(\mathcal{W})$ is a basis of the syndrome subspace $\langle \sigma(\mathcal{Y}) \rangle$.*

Notice that a pseudo-basis has thus cardinality at most $n-k$, and that it can be computed in time polynomial in n.

The following property formalizes the data that Bob can acquire after he obtains a pseudo-basis of the words received by Alice:

Proposition 4.1.4 ([45]). *Let r be a positive integer, and let $\mathcal{X}, \mathcal{E}, \mathcal{Y}$ be three subsets:*

- $\mathcal{X} := \left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(r)}\right\}$ *a set of codewords of $\mathcal{C}$,*
- $\mathcal{E} := \left\{\mathbf{e}^{(1)}, \ldots, \mathbf{e}^{(r)}\right\}$ *a set of error vectors such that* $\left|\bigcup \left(\text{support}\left(\mathbf{e}^{(j)}\right) : j = 1, \ldots, r\right)\right| < d$,
- $\mathcal{Y} := \left\{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(r)}\right\} \subseteq \mathbb{F}_q^n$ *with* $\mathbf{y}^{(j)} = \mathbf{x}^{(j)} + \mathbf{e}^{(j)}$ *for every* $j = 1, \ldots, r$.

Then, given knowledge of $\mathcal{X}$ and a pseudo-basis of $\mathcal{Y}$, we have that $\mathbf{e}^{(j)}$ can be computed from its syndrome $\sigma(\mathbf{e}^{(j)})$ for any $1 \leq j \leq r$.

Proof. The hypothesis on the supports of the elements of $\mathcal{E}$ implies that the subspace $\langle \mathcal{E} \rangle$ satisfies the hypothesis of Lemma 4.1.3 and the syndrome function is therefore injective on $\langle \mathcal{E} \rangle$. Given the pseudo-basis $\left\{\mathbf{y}^{(i)} : i \in I\right\}$, we can decompose any syndrome $\sigma(\mathbf{e}^{(j)})$ as

$$\begin{aligned}\sigma(\mathbf{e}^{(j)}) = \sigma(\mathbf{y}^{(j)}) &= \sum_{i \in I} \lambda_i \sigma(\mathbf{y}^{(i)}) \\ &= \sum_{i \in I} \lambda_i \sigma(\mathbf{e}^{(i)}) \\ &= \sigma\left(\sum_{i \in I} \lambda_i \mathbf{e}^{(i)}\right)\end{aligned}$$

which yields

$$\mathbf{e}^{(j)} = \sum_{i \in I} \lambda_i \mathbf{e}^{(i)} = \sum_{i \in I} \lambda_i (\mathbf{y}^{(i)} - \mathbf{x}^{(i)})$$

by injectivity of σ on $\langle \mathcal{E} \rangle$ (cf. Lemma 4.1.3).

□

Remark 4.1.1. Since the syndrome map induces a one-to-one mapping from $\langle \mathcal{E} \rangle$ to $\sigma(\langle \mathcal{E} \rangle)$, we also have that $\left\{\mathbf{y}^{(i)} : i \in I\right\}$ is a pseudo-basis of $\mathcal{Y}$ if and only if $\left\{\mathbf{e}^{(i)} : i \in I\right\}$ is a basis of $\langle \mathcal{E} \rangle$.

The reader should now have a clear picture of how the pseudo-basis will be used to obtain shared randomness: Bob will select a few codewords $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(r)}$ in an MDS code of distance at least $t+1$, then communicate them to Alice by sending the i-th symbol of each codeword over channel i; Alice will be able to compute a pseudo-basis of the received words, a clearly non-expensive computation, then communicate it to Bob. Bob will then be able to determine any error introduced by Eve just from its syndrome as just showed in Proposition 4.1.4.

The following section gives all the details.

4.1.5 A First Protocol

We now present the complete version of our first communication protocol, following the blueprint of Section 4.1.2.

Protocol 4.1.1. *The protocol works in two rounds, and allows Alice to communicate ℓ secret elements $s^{(1)}, \cdots, s^{(\ell)}$ of $\mathbb{F}_q$ to Bob, where q is an arbitrary prime power with $q > n$. The protocol uses the MDS code $\mathcal{C}$ of parameters $[n, t+1, t+1]_q$ and the linear function $f : \mathcal{C} \to \mathbb{F}_q$ of Lemma 4.1.1 for private communication, and the broadcast technique of Lemma 4.1.2 for reliable communication.*

- Round 1: *Bob chooses $t+\ell$ uniformly random and independent codewords $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t+\ell)}$ of $\mathcal{C}$ and communicates them to Alice by sending the i-th symbol of each codeword over the i-th channel.*
- Round 2: *Alice receives the corrupted versions $\mathbf{y}^{(1)} = \mathbf{x}^{(1)} + \mathbf{e}^{(1)}, \cdots, \mathbf{y}^{(t+\ell)} = \mathbf{x}^{(t+\ell)} + \mathbf{e}^{(t+\ell)}$; she then proceeds with the following actions:*
  - *(i) She computes a pseudo-basis $\left(\mathbf{y}^{(i)} : i \in I\right)$ for $I \subset \{1, \cdots, t+\ell\}$ of the received values and broadcasts to Bob $\left(i, \mathbf{y}^{(i)} : i \in I\right)$.*
  - *(ii) She then considers the first ℓ words that do* not *belong to the pseudo-basis; to ease the notation, we will re-name them $\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(\ell)}$. For each secret $s^{(j)}$ to be communicated she broadcasts to Bob the following two elements:*
    - *$\mathbf{H}\left(\mathbf{y}^{(j)}\right)^T$, the syndrome of $\mathbf{y}^{(j)}$;*
    - *$s^{(j)} + f\left(\mathbf{y}^{(j)}\right)$.*

Proposition 4.1.4 now guarantees that for any j with $1 \leq j \leq \ell$ Bob can compute the error vector $\mathbf{e}^{(j)}$ and hence reconstruct $\mathbf{y}^{(j)} = \mathbf{x}^{(j)} + \mathbf{e}^{(j)}$ from his knowledge of $\mathbf{x}^{(j)}$. He can therefore open the mask $f\left(\mathbf{y}^{(j)}\right)$ and obtain the secret $s^{(j)}$.

Proposition 4.1.5. *Protocol 4.1.1 allows for secure communication of ℓ elements of $\mathbb{F}_q$ against an adversary Eve controlling any subset of t channels.*

Proof. As a first remark, notice that since the pseudo-basis has cardinality at most t as remarked in Definition 4.1.2, Alice has enough words to mask her ℓ secret messages, since the total number of words is equal to $t+\ell$. We can now prove that the protocol is private and reliable:

- *Privacy:* we need to prove that the data eavesdropped by Eve has a distribution independent of the secret messages $s^{(1)}, \ldots, s^{(\ell)}$. Denote by $\mathcal{A} \subseteq \{1, \ldots, n\}$ the indices of the channels controlled by Eve; then Eve acquires the following data during an execution of the protocol:

  I. $\mathbf{x}_{\mathcal{A}}^{(1)}, \ldots, \mathbf{x}_{\mathcal{A}}^{(t+\ell)}$;

  II. $\left(\mathbf{y}^{(i)} : i \in I\right)$, the pseudo-basis;

  III. $\mathbf{H}\left(\mathbf{y}^{(j)}\right)^T$ for any $j \notin I$

  IV. $s^{(1)} + f\left(\mathbf{y}^{(j_1)}\right), \ldots, s^{(\ell)} + f\left(\mathbf{y}^{(j_\ell)}\right)$ where $j_i \notin I$ for any i.

  First recall that the elements $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t+\ell)}$ are chosen independently of each other and of the secrets, which means that the data of point II can be ignored; similarly, since $\mathbf{H}\left(\mathbf{y}^{(j)}\right)^T = \mathbf{H}\left(\mathbf{x}^{(j)} + \mathbf{e}^{(j)}\right)^T = \mathbf{H}\left(\mathbf{e}^{(j)}\right)^T$, the data of point III only depends on the errors introduced by Eve during round 1. This means that it suffices to prove that $(\mathbf{x}_{\mathcal{A}}^{(j)},\ f(\mathbf{y}^{(j)}) + s^{(j)} : j)$ has a distribution which does not depend on the secrets $s^{(1)}, \ldots, s^{(\ell)}$.

  By Lemma 4.1.1, $s^{(i)} + f(\mathbf{y}^{(j_i)})$ is uniformly distributed and independent of $\mathbf{x}_{\mathcal{A}}^{(j_i)}$; hence $(\mathbf{x}_{\mathcal{A}}^{(j_i)}, s^{(i)} + f(\mathbf{y}^{(j_i)}) : i)$ is uniformly distributed over $\mathcal{C}_{\mathcal{A}}^{t+\ell} \times \mathbb{F}_q^{\ell}$. In particular, its distribution does not depend on the choice of s, so that privacy holds.

- *Reliability:* Eve can disrupt the communication only during Round 1, since Round 2 only uses broadcasts. Proposition 4.1.4 then ensures that Bob can recover the vectors $\mathbf{y}^{(j)}$ from their syndromes and the corresponding codeword $\mathbf{x}^{(j)}$. From there he can compute and remove the mask $f\left(\mathbf{y}^{(j)}\right)$ without error.

□

We now compute the communication complexity and transmission rate of this first protocol, underlining the most expensive parts:

Communication complexity: we can set $\ell := 1$.

- Step I requires transmitting $t + 1$ codewords, thus requiring a total of $O\left(n^2\right)$ symbols to be transmitted.
- Step II-(i) requires broadcasting up to t words of $\mathbb{F}_q^n$, thus giving a total of $O\left(n^3\right)$ symbols to be transmitted.
- Finally, step II-(ii) requires broadcasting a total of $t+1$ symbols (a size-t syndrome and the masked secret), thus giving a total of $O\left(n^2\right)$ elements to be transmitted.

Hence since we can assume that $q = O(n)$, we get a total communication complexity of

$$O\left(n^3 \log n\right)$$

bits to be transmitted to communicate a single-bit secret.

Tranfer rate: optimal rate is achieved for $\ell = \Omega\left(n\right)$.

- Step I requires transmitting $t + \ell$ codewords, for a total of $O\left(n^2 + n\ell\right)$ symbols.
- Step II-(i) remains unchanged from the single-bit case, and thus requires transmitting $O\left(n^3\right)$ symbols.
- Finally, step II-(ii) requires broadcasting a total of $\ell(t + 1)$ symbols (ℓ size-t syndromes and the masked secrets), thus giving a total of $O\left(n^2\ell\right)$ symbols;

To sum up, the overall transmission rate is equal to

$$\frac{O\left(n^2 + n\ell + n^3 + n^2\ell\right)}{\ell} = O\left(n^2\right).$$

It is immediately seen that the main bottleneck for communication complexity is step II-(i), i.e. the communication of the pseudo-basis, while for transmission rate it is step II-(ii), i.e. the communication of the masked secrets and of the syndromes. We address these issues in the following section.

4.1.6 The Improvements to the Protocol

We show in this section how to improve communication complexity and transfer rate.

Generalized Broadcast Our improvements on the two bottlenecks showed in Section 4.1.5 rely on the fundamental technique of generalized broadcast, which has been highlighted in the paper by Kurosawa and Suzuki [45].

The intuition is the following: we want to choose a suitable code $\mathcal{C}_{\texttt{BCAST}}$ for perfectly reliable transmission, i.e. we require that if any word $\mathbf{x} \in \mathcal{C}_{\texttt{BCAST}}$ is communicated by sending each symbol $\mathbf{x}_i$ over the i-th channel, then $\mathbf{x}$ can always be recovered in spite of the errors introduced by Eve. In the general situation, since Eve can introduce up to t errors, $\mathcal{C}_{\texttt{BCAST}}$ must have minimum distance $2t + 1 = n$, and hence dimension 1; for instance, $\mathcal{C}_{\texttt{BCAST}}$ can be a repetition code, yielding the broadcast protocol of Section 4.1.3.

Now assume that at a certain point of the protocol, Bob gets to know the position of m coordinates under Eve's control; then the communication system between the two has been improved: instead of n channels with t errors, we have n channels with m erasures and $t - m$ errors (since Bob can ignore the received on the m channels under Eve's control that he has identified). We can thus expect that reliable communication between Alice and Bob (i.e., broadcast) can be performed at a lower cost by using a code with smaller distance and greater dimension; the following lemma formalizes this intuition.

Lemma 4.1.6 (**Generalized Broadcast**). *Let $m \leq t$ and let $\mathcal{C}_m$ be an MDS code of parameters $[n, m + 1, n - m]_q$; assume that Bob knows the location of m channels controlled by Eve. Then Alice can communicate with perfect reliability $m + 1$ symbols $x_1, \cdots, x_{m+1}$ of $\mathbb{F}_q$ to Bob in the following way: she first takes the codeword $\mathbf{c} \in \mathcal{C}_m$ which encodes $(x_1, \cdots, x_{m+1})$, then sends each symbol of $\mathbf{c}$ through the corresponding channel; Eve cannot prevent Bob from completely recovering the message.*

We refer to this procedure as m-generalized broadcast.

Proof. Notice that $\mathbf{c}$ is well-defined since $\mathcal{C}_m$ has dimension $m + 1$. Now since Bob knows the location of m channels that are under Eve's control, he can replace the symbols of $\mathbf{c}$ received via these channels with erasure marks $\perp$, and consider the truncated codeword $\tilde{\mathbf{c}}$ lacking these symbols. Now $\tilde{\mathbf{c}}$ belongs to the punctured code obtained from $\mathcal{C}_m$ by removing m coordinates, which has minimum distance $(n - m) - m \geq 2(t - m) + 1$; it can thus correct up to $t - m$ errors, which is exactly the maximum number of errors that Eve can introduce (since she controls at most $t - m$ of the remaining channels). Once

he has obtained the shortened codeword $\tilde{\mathbf{c}}$, he can then recover the complete one since $\mathcal{C}_m$ can correct from m erasures, given that it has minimum distance $n - m \geq m$.

□

Hence if Alice knows that Bob has identified at least m coordinates under Eve's control, she can divide the cost of a broadcast by a factor m (since the above method requires to transmit n symbols of $\mathbb{F}_q$ to communicate $m + 1$ symbols of $\mathbb{F}_q$).

In the following paragraphs we will make use of Lemma 4.1.6 to improve the efficiency of the protocol.

Improved Transmission of the Pseudo-Basis: a Warm-Up. We present here a new method of communicating the pseudo-basis, which is a straightforward implementation of the generalized broadcasting technique.

The key point is the following observation:

Lemma 4.1.7. *Let $\mathcal{W} = \left(\mathbf{y}^{(i)} : i \in I\right)$ be a pseudo-basis of the set of received vectors; then if Bob knows m elements of $\mathcal{W}$, he knows at least m channels that are under Eve's control.*

Proof. By subtracting the original codeword from an element of the pseudo-basis, Bob knows the corresponding error; furthermore, these errors form a basis of the entire error space (Remark 4.1.1). Now if Bob knows m elements of the pseudo-basis, he then knows m of these errors, which necessarily affect at least m coordinates since they are linearly independent. The claim then follows.

□

The sub-protocol consisting of the transmission of the pseudo-basis by Alice is simply the following:

Protocol 4.1.2. *Alice wishes to communicate to Bob a pseudo-basis $\mathcal{W}$ of cardinality w.*

For any $i = 1, \cdots, w$, she then uses $(i-1)$-generalized broadcast to communicate the i-th element of the pseudo-basis to Bob.

Lemmas 4.1.6 and 4.1.7 ensure that this technique is secure; we now compute its cost:

- Each element of the pseudo-basis is a vector of $\mathbb{F}_q^n$;
- using m-generalized broadcast to communicate n elements of $\mathbb{F}_q$ requires communicating $\left\lceil \frac{n}{m+1} \right\rceil n$ field elements;
- hence Protocol 4.1.2 requires communicating the following number of elements of $\mathbb{F}_q$:

$$\sum_{i=1}^{w} \left\lceil \frac{n}{i} \right\rceil n = O\left(n^2 \sum_{i=1}^{w} \frac{1}{i} \right) = O\left(n^2 \log n\right)$$

  which means that we have reduced to $O\left(n^2 \log^2 n\right)$ the total communication complexity.

This complexity is still one logarithmic factor short of our goal; we now show how a more advanced technique allows us to bring down the cost to $O\left(n^2\right)$ field elements.

Improved Transmission of the Pseudo-Basis: the Final Version. In this paragraph we show a more advanced technique to communicate the pseudo-basis. The key idea is the following: denote by w the size of the pseudo-basis; if Alice can find a received word $\mathbf{y}$ which is subject to an error of weight cw for some constant c and sends it to Bob, then Bob will learn the position of at least cw corrupted coordinates. Alice will thus be able to use cw-generalized broadcast as in Lemma 4.1.6 to communicate the elements of the pseudo-basis (which amount to wn symbols); since cw-generalized broadcast of a symbol has a cost of $O(n/cw)$, the total cost of communicating the pseudo-basis will thus be $(wn) \cdot O(n/cw) = O\left(n^2\right)$.

We thus devise an algorithm that allows Alice to find a word $\mathbf{y}$ subject to at least $m = \Omega(w)$ errors (for instance, such condition is met if $\mathbf{y}$ is subject to $\Omega(t)$ errors, since $w \leq t$). Notice that such a word $\mathbf{y}$ may not exist among the received words $\left\{\mathbf{y}^{(i)}\right\}$, therefore we will look for a linear combination of the $\mathbf{y}^{(i)}$ with this property.

As mentioned in Section 4.1.2, Alice will make extensive use of a decoding algorithm. Recall that a code of distance d can be uniquely decoded from up to $\lfloor (d-1)/2 \rfloor$ errors, and that in the case of Reed-Solomon codes, such decoding can be performed in time polynomial in n [49]; this means that for any Reed-Solomon code $\mathcal{C}$ there exists an algorithm that takes as input

a word $\mathbf{y} \in \mathbb{F}_q^n$ and outputs a decomposition $\mathbf{y} = \mathbf{x} + \mathbf{e}$ with $\mathbf{x} \in \mathcal{C}$ and $w_{\mathrm{H}}(\mathbf{e}) \leq \lfloor (d-1)/2 \rfloor$ (if such a decomposition does not exist, the algorithm outputs an error message $\perp$).

Protocol 4.1.3. *Alice has received the words $\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(r)}$ and has computed a pseudo-basis $\{\mathbf{y}^{(i)} : i \in I\}$ of them; denote by w its cardinality. Alice proceeds with the following actions:*

- *she uses Algorithm 4.1.1 below to find a "special word" $\mathbf{y}$, with coefficients $(\mu_i : i \in I)$ such that $\mathbf{y} = \sum_{i \in I} \mu_i \mathbf{y}^{(i)}$. She then communicates to Bob the triplet $\big(I, (\mu_i : i \in I), \mathbf{y}\big)$ by using ordinary broadcast.*
- *Finally, she communicates the pseudo-basis of the received values by using m-generalized broadcast, where $m := \min(w, t/3)$.*

Before describing the algorithm formally and proving its validity, we sketch the idea. Alice has computed a pseudo-basis $\{\mathbf{y}^{(i)} : i \in I\}$. For $i \in I$, she applies the decoding algorithm to $\mathbf{y}^{(i)} = \mathbf{x}^{(i)} + \mathbf{e}^{(i)}$. If the decoding algorithm fails, it means that $\mathbf{y}^{(i)}$ is at a large Hamming distance from any codeword, in particular from Bob's codeword $\mathbf{x}^{(i)}$, and the single $\mathbf{y}^{(i)}$ is the required linear combination. If the decoding algorithm succeeds for every i, Alice obtains decompositions

$$\mathbf{y}^{(i)} = \tilde{\mathbf{x}}^{(i)} + \tilde{\mathbf{e}}^{(i)}$$

where $\tilde{\mathbf{x}}^{(i)}$ is some codeword. Alice must be careful, because she has no guarantee that the codeword $\tilde{\mathbf{x}}^{(i)}$ coincides with Bob's codeword $\mathbf{x}^{(i)}$, and hence that $\tilde{\mathbf{e}}^{(i)}$ coincides with Eve's error vector $\mathbf{e}^{(i)}$. What Alice then does is look for a linear combination $\sum_i \mu_i \tilde{\mathbf{e}}^{(i)}$ that has Hamming weight at least $t/3$ and at most $2t/3$. If she is able to find one, then a simple Hamming distance argument guarantees that the corresponding linear combination of Eve's original errors $\sum_i \mu_i \mathbf{e}^{(i)}$ also has Hamming weight at least $t/3$. If Alice is unable to find such a linear combination, then she falls back on constructing one that has weight not more than $2t/3$ and at least the cardinality w of the pseudo-basis; this will yield an alternative form of the desired result. We now describe this formally.

Algorithm 4.1.1. *Alice has a pseudo-basis* $\left(\mathbf{y}^{(i)} : i = 1, \cdots, w\right)$ *(indices have been changed to simplify the notation); the algorithm allows Alice to identify a word* $\mathbf{y}$ *subject to at least* $m := \min(w, t/3)$ *errors introduced by Eve.*

In the following steps, whenever we say that the output of the algorithm is a word $\mathbf{y}^{(i)}$*, we implicitly assume that the algorithm also outputs the index* i*; more generally, whenever the algorithm outputs a linear combination* $\sum_i \mu_i \mathbf{y}^{(i)}$ *of the words in the pseudo-basis, we assume that it also outputs the coefficient vector* $(\mu_1, \cdots, \mu_w)$ *of the linear combination.*

1. *Alice uses a unique-decoding algorithm to decode the elements of the pseudo-basis; if the algorithm fails for a given word* $\mathbf{y}^{(i)}$ *(i.e., it outputs an error message* $\perp$*), then Algorithm 4.1.1 stops and outputs* $\mathbf{y}^{(i)}$*.*

2. *If the decoding algorithm worked for every* i*, Alice gets a decomposition* $\mathbf{y}^{(i)} = \tilde{\mathbf{x}}^{(i)} + \tilde{\mathbf{e}}^{(i)}$ *with* $\tilde{\mathbf{x}}^{(i)} \in \mathcal{C}$ *and* $w_{\mathrm{H}}\left(\tilde{\mathbf{e}}^{(i)}\right) \leq t/2$ *for every* i*; notice that it is not guaranteed that the* $\tilde{\mathbf{x}}^{(i)}$ *coincide with the codewords* $\mathbf{x}^{(i)}$ *originally chosen by Bob.*

   If any of the $\tilde{\mathbf{e}}^{(i)}$ *has weight greater than* $t/3$*, the algorithm stops and outputs* $\mathbf{y}^{(i)}$*.*

3. *Define* $\tilde{\mathbf{f}}^{(1)} := \tilde{\mathbf{e}}^{(1)}$ *and* $\tilde{\mathbf{y}}^{(1)} := \mathbf{y}^{(1)}$*. For any* $i = 2, \cdots, w$*, proceed with the following actions:*

   - *let* $\lambda^{(i)}$ *be a non-zero element of* $\mathbb{F}_q$ *such that* $\tilde{\mathbf{f}}_j^{(i-1)} + \lambda^{(i)} \tilde{\mathbf{e}}_j^{(i)} \neq 0$ *for any coordinate* $j \in \{1, 2, \ldots, n\}$ *for which* $\tilde{\mathbf{f}}_j^{(i-1)} \neq 0$*.*
   - *let* $\tilde{\mathbf{f}}^{(i)} := \tilde{\mathbf{f}}^{(i-1)} + \lambda^{(i)} \tilde{\mathbf{e}}^{(i)}$ *and* $\tilde{\mathbf{y}}^{(i)} := \tilde{\mathbf{y}}^{(i-1)} + \lambda^{(i)} \mathbf{y}^{(i)}$*; if* $w_{\mathrm{H}}\left(\tilde{\mathbf{f}}^{(i)}\right) > t/3$*, stop and output* $\tilde{\mathbf{y}}^{(i)}$*.*

4. *Output* $\tilde{\mathbf{y}}^{(w)}$*.*

We can now prove that this algorithm allows Alice to find the desired codeword, which naturally implies that Protocol 4.1.3 indeed allows for reliable communication of the pseudo-basis:

Proposition 4.1.8. *Algorithm 4.1.1 allows Alice to find a word* $\mathbf{y}$ *subject to an error introduced by Eve of weight at least* $m := \min(w, t/3)$*.*

Proof. The following observation is the key point of the algorithm:

Lemma 4.1.9. *Let* $\mathbf{y} = \mathbf{x} + \mathbf{e} = \tilde{\mathbf{x}} + \tilde{\mathbf{e}}$ *for* $\mathbf{x}, \tilde{\mathbf{x}} \in \mathcal{C}$. *Then if* $\tilde{\mathbf{e}}$ *satisfies* $w_{\mathrm{H}}(\tilde{\mathbf{e}}) \leq 2t/3$, *we have that* $w_{\mathrm{H}}(\mathbf{e}) \geq \min\{w_{\mathrm{H}}(\tilde{\mathbf{e}}), t/3\}$.

Proof. The claim is trivial if $\mathbf{e} = \tilde{\mathbf{e}}$; hence assume that $\mathbf{e} \neq \tilde{\mathbf{e}}$. Notice that $\mathbf{e} - \tilde{\mathbf{e}} = \tilde{\mathbf{x}} - \mathbf{x}$; hence since $d_{\min}(\mathcal{C}) = t + 1$, we have that

$$t + 1 \leq w_{\mathrm{H}}(\mathbf{e} - \tilde{\mathbf{e}}) \leq w_{\mathrm{H}}(\mathbf{e}) + w_{\mathrm{H}}(\tilde{\mathbf{e}}) \leq w_{\mathrm{H}}(\mathbf{e}) + \frac{2t}{3}$$

Hence we have that $w_{\mathrm{H}}(\mathbf{e}) \geq t/3$, so that the claim is proved.

□

We now analyze the algorithm step-by-step:

1. if decoding fails for a word $\mathbf{y}^{(i)}$, then it is guaranteed that the error introduced by Eve on it has weight bigger than $t/2 > m$ (otherwise, the unique decoding algorithm would succeed since $d_{\min}(\mathcal{C}) = t + 1$).
2. since by assumption $w_{\mathrm{H}}\left(\tilde{\mathbf{e}}^{(i)}\right) \leq t/2 \leq 2t/3$, if we also have $t/3 \leq w_{\mathrm{H}}\left(\tilde{\mathbf{e}}^{(i)}\right)$, then thanks to Lemma 4.1.9 the output $\mathbf{y}^{(i)}$ is of the desired type.
3. Since the algorithm did not abort at step 2, all elements $\tilde{\mathbf{e}}^{(i)}$ have weight at most $t/3$.

   First notice that if the algorithm did not produce $\tilde{\mathbf{f}}^{(i-1)}$ as output, then $\tilde{\mathbf{f}}^{(i)}$ is well-defined: indeed, we have that $w_{\mathrm{H}}\left(\tilde{\mathbf{f}}^{(i-1)}\right) \leq t/3$; this means that $\lambda^{(i)}$ is well-defined, since it is an element of $\mathbb{F}_q$ that has to be different from 0 and from at most $t/3 < n - 1$ elements.

   Now if the algorithm outputs $\tilde{\mathbf{f}}^{(i)}$, then necessarily $w_{\mathrm{H}}\left(\tilde{\mathbf{f}}^{(i-1)}\right) \leq t/3$ (otherwise the algorithm would have stopped before computing $\tilde{\mathbf{f}}^{(i)}$); furthermore, by assumption we have that $w_{\mathrm{H}}\left(\tilde{\mathbf{e}}^{(i)}\right) \leq t/3$, so that $w_{\mathrm{H}}\left(\tilde{\mathbf{f}}^{(i)}\right) \leq 2t/3$ and we can apply Lemma 4.1.9, so that the output is of the desired type.
4. Notice that for any $i = 1, \cdots, w$, we have that $\tilde{\mathbf{f}}^{(i)}$ has maximal weight among elements of the vector space $\langle \tilde{\mathbf{e}}^{(1)}, \cdots, \tilde{\mathbf{e}}^{(i)} \rangle$ (the condition on $\lambda^{(i)}$ ensures that this condition is met at each step). Hence since the elements $\left\{\tilde{\mathbf{e}}^{(1)}, \cdots, \tilde{\mathbf{e}}^{(w)}\right\}$ are linearly independent (because their syndromes are linearly independent, since $(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(w)})$ is a pseudo-basis), we have that $w_{\mathrm{H}}\left(\tilde{\mathbf{f}}^{(i)}\right) \geq i$ for any i.

In particular, we have that $w_{\mathrm{H}}\left(\tilde{\mathbf{f}}^{(w)}\right) \geq w$; hence since $w_{\mathrm{H}}\left(\tilde{\mathbf{f}}^{(w)}\right) \leq 2t/3$ as remarked above, we have that the output $\tilde{\mathbf{y}}^{(w)}$ is of the desired type.

□

Remark 4.1.2. Protocol 4.1.3 requires Alice to use ordinary broadcast to communicate a single vector of $\mathbb{F}_q^n$ (hence transmitting n^2 elements of $\mathbb{F}_q$), then to use m-generalized broadcast with $m \geq \min\{w, t/3\}$ to communicate $w \leq t$ vectors of $\mathbb{F}_q^n$ (hence transmitting at most $3n^2$ elements of $\mathbb{F}_q$). We thus get a total of at most $4n^2$ elements of $\mathbb{F}_q$ to be transmitted.

Furthermore, Algorithm 4.1.1 has running time polynomial in n, as long as the code $\mathcal{C}$ has a unique-decoding algorithm of polynomial running time as well. As already remarked, such algorithms exist for instance for Reed-Solomon codes.

We then study the second bottleneck of Protocol 4.1.1.

The Improved Communication of the Masked Secrets We present here the second key improvement to the protocol: after the pseudo-basis is communicated, we devise a way to lower the cost of transmitting to Bob the masked secrets and the information to open the masks. We aim at a cost linear in the number ℓ of secrets to be transmitted (while it was quadratic in Protocol 4.1.1). As in the previous paragraph, Alice makes use of a unique-decoding algorithm.

Protocol 4.1.4. *We assume that Alice wishes to communicate ℓ secret elements $s^{(1)}, \cdots, s^{(\ell)}$ of $\mathbb{F}_q$ to Bob, and that ℓ codewords $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(\ell)}$ of $\mathcal{C}$ have been sent by Bob to Alice (who has received $\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(\ell)}$) and have not been disclosed in other phases. The protocol is performed once the pseudo-basis has been communicated to Bob; we thus assume that Bob knows the global support $\mathcal{S} := \cup_i \mathrm{support}\left(\mathbf{e}^{(i)}\right)$ of the errors affecting the elements $\mathbf{y}^{(i)}$ (cf. Remark 4.1.1).*

- *Alice uses a unique-decoding algorithm to decode $\mathbf{y}^{(i)}$, so that for every i she obtains (if decoding was successful) a decomposition $\mathbf{y}^{(i)} = \tilde{\mathbf{x}}^{(i)} + \tilde{\mathbf{e}}^{(i)}$ with $\tilde{\mathbf{x}}^{(i)} \in \mathcal{C}$ and $w_{\mathrm{H}}\left(\tilde{\mathbf{e}}^{(i)}\right) \leq t/2$.*

  For every $i = 1, \cdots, \ell$ she then communicates the following elements to Bob:

  - *the syndrome $\mathbf{H}\left(\mathbf{y}^{(i)}\right)^T$ via $t/2$-generalized broadcast;*

– *the elements* $z_1^{(i)}, z_2^{(i)}$ *of* $\mathbb{F}_q$ *by ordinary broadcast, where*

$$z_1^{(i)} := \quad s^{(i)} + f\left(\mathbf{y}^{(i)}\right)$$
$$z_2^{(i)} := \begin{cases} s^{(i)} + f\left(\tilde{\mathbf{x}}^{(i)}\right) & \text{if decoding succeeded,} \\ 0 & \text{otherwise.} \end{cases}$$

- *Bob can then obtain each secret* $s^{(i)}$ *in a different way depending on the size of the global support* $\mathcal{S}$ *of the errors:*
  - *if* $|\mathcal{S}| \geq t/2$*, he uses the knowledge of the syndrome of* $\mathbf{y}^{(i)}$ *and of the support of the error to compute* $\mathbf{y}^{(i)}$*, so that he can compute* $z_1^{(i)} - f\left(\mathbf{y}^{(i)}\right)$ *as well.*
  - *if* $|\mathcal{S}| < t/2$*, he ignores the syndrome that has been communicated to him, and computes* $z_2^{(i)} - f\left(\mathbf{x}^{(i)}\right)$.

Proposition 4.1.10. *The above protocol allows for private and reliable communication of* ℓ *elements of* $\mathbb{F}_q$.

Proof. We check privacy and reliability.

Privacy: we can prove that the data acquired by Eve is independent from the secrets as in Proposition 4.1.5; the only relevant difference is that Eve may learn the value of $z_2^{(i)}$ for some i. By dropping indexes to simplify notation, let $z_2 := z_2^{(i)}$; we then have that $z_2 = s + f(\tilde{\mathbf{x}}) = z_1 - f(\tilde{\mathbf{e}})$. It is thus sufficient to show that $f(\tilde{\mathbf{e}})$ is a function of the other data possessed by Eve; we prove this claim in the following lemma:

Lemma 4.1.11. *Let* $\mathbf{x}$ *be a codeword sent by Bob to Alice, and let* $\mathbf{y} = \mathbf{x} + \mathbf{e}$ *be the received vector. Then Eve knows whether* $\mathbf{y}$ *can be decoded (i.e.* $\mathbf{y} = \tilde{\mathbf{x}} + \tilde{\mathbf{e}}$ *as above) or not; furthermore, if* $\mathbf{y}$ *can be decoded, then she knows* $\tilde{\mathbf{e}}$.

Proof. By definition, $\tilde{\mathbf{e}}$ is a vector of minimum weight (and of weight at most $t/2$) such that $\mathbf{y} - \tilde{\mathbf{e}}$ belongs to $\mathcal{C}$; notice that the last condition is equivalent to require that $\mathbf{e} - \tilde{\mathbf{e}}$ belongs to $\mathcal{C}$. Now these requirements uniquely determine $\tilde{\mathbf{e}}$: indeed, if by contradiction $\mathbf{e} - \mathbf{e}' \in \mathcal{C}$ for another $\mathbf{e}'$, then $\mathbf{e}' - \tilde{\mathbf{e}}$ would belong to $\mathcal{C}$, a contradiction since $w_{\mathrm{H}}(\mathbf{e}' - \tilde{\mathbf{e}}) \leq t/2 + t/2 < d_{\min}(\mathcal{C})$.

Hence $\tilde{\mathbf{e}}$ is uniquely determined by $\mathbf{e}$ and $\mathcal{C}$: Eve can thus compute it from the data in her possession. Notice that, in particular, she knows whether $\tilde{\mathbf{e}}$ exists or not, i.e. whether decoding of $\mathbf{y}$ is possible or not.

□

Reliability: we have two possible cases:

- if $|\mathcal{S}| \geq t/2$, then Bob is able to acquire the syndrome $\mathbf{H}\mathbf{y}^T$ of $\mathbf{y}$ via $t/2$-generalized broadcast (cf. Lemma 4.1.6); thus as remarked in Proposition 4.1.5, he can recover $\mathbf{y}$ and open the mask to get the secret.
- if $|\mathcal{S}| < t/2$, then Bob knows that Alice has correctly decoded $\mathbf{y}$, since Eve introduced less than $d_{\min}/2$ errors; thus $\tilde{\mathbf{x}} = \mathbf{x}$ so that $z_2 - f(\mathbf{x}) = (s + f(\tilde{\mathbf{x}})) - f(\mathbf{x}) = s$.

  Notice that in this case Bob will have failed to decode the $t/2$-generalized broadcast but he will simply ignore the elements received in this way.

□

Remark 4.1.3. Notice that we could further improve the efficiency of this protocol by requiring Alice to use w-generalized broadcast (instead of regular one) to communicate the elements $z_1^{(i)}$ and $z_2^{(i)}$, where w is the size of the pseudo-basis; this, however, would not reduce the order of magnitude of the total cost.

The Improved Protocol The improved protocol simply implements the new techniques to communicate the pseudo-basis and the masked secrets.

Protocol 4.1.5. *The protocol works in two rounds, and allows Alice to communicate ℓ secret elements $s^{(1)}, \cdots, s^{(\ell)}$ of $\mathbb{F}_q$ to Bob, where q is an arbitrary prime power with $q > n$. The protocol uses Massey's scheme (cf. Lemma 4.1.1) for private communication, i.e. an MDS code $\mathcal{C}$ of parameters $[n, t+1, n-t]$ together with a linear function $f : \mathcal{C} \to \mathbb{F}_q$, and the broadcast techniques of Lemma 4.1.2 and 4.1.6 for reliable communication.*

- Round 1: *Bob chooses $t + \ell + 1$ uniformly random and independent codewords $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t+\ell+1)}$ of $\mathcal{C}$ and sends them over the channels to Alice.*
- Round 2: *Alice receives the corrupted versions $\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(t+\ell+1)}$, and she computes a pseudo-basis $\left\{\mathbf{y}^{(i)} : i \in I\right\}$ of the received values; she then proceeds with the following actions:*

(i) She uses Protocol 4.1.3 to communicate the pseudo-basis to Bob.

(ii) She then uses the remaining words to communicate to Bob the masked secrets and the data to retrieve them as in the first part of Protocol 4.1.4.

Upon receiving the pseudo-basis, Bob proceeds to compute the global support $\mathcal{S}$ of the error space; he can then obtain each secret $s^{(i)}$ as specified in the corresponding part of Protocol 4.1.4.

Notice that privacy and reliability of the protocol follow from the previous discussions; we now analyze the complexity of the protocol:

Communication complexity: we can set $\ell := 1$.

- Round 1 requires transmitting $t+2$ vectors of $\mathbb{F}_q^n$, thus requiring a total of $O\left(n^2\right)$ symbols to be transmitted.
- Round 2-(i) requires transmitting $O\left(n^2\right)$ elements of $\mathbb{F}_q$ as shown in Remark 4.1.2.
- Finally, Round 2-(ii) requires using $t/2$-generalized broadcast to communicate n symbols, and standard broadcast to communicate 2 symbols, thus giving a total of $O(n)$ elements to be transmitted.

Hence since we can assume that $q = O(n)$, we get a total communication complexity of

$$O\left(n^2 \log n\right)$$

bits to be transmitted to communicate a single-bit secret.

Transfer rate: optimal rate is achieved for $\ell = \Omega\left(n\right)$.

- Round 1 requires transmitting $t+\ell+1 = \ell+O(n)$ codewords, for a total of $n\ell + O(n^2)$ symbols.
- Round 2-(i) remains unchanged from the single-bit case, and thus requires transmitting $O\left(n^2\right)$ symbols.

- Finally, Round 2-(ii) uses $t/2$-generalized broadcast to communicate ℓt elements of $\mathbb{F}_q$ and standard broadcast to communicate 2ℓ elements of $\mathbb{F}_q$, so that the overall cost is equal to $4n\ell$ symbols to be transmitted.

To sum up, the overall transmission rate is equal to

$$\frac{5n\ell + O\left(n^2\right)}{\ell} = 5n + O\left(n^2/\ell\right)$$

Furthermore, by using Reed-Solomon codes (instead of arbitrary MDS ones), we then have that Protocol 4.1.5 has computational cost polynomial in n for both Alice and Bob.

4.1.7 Concluding Remarks and Open Problems

We have presented a two-round PSMT protocol that has polynomial computational cost for both sender and receiver, and that achieves transmission rate linear in n and communication complexity in $O\left(n^2 \log n\right)$. As proved in [68], the transfer rate is asymptotically optimal; furthermore, our protocol has a low multiplicative constant of 5. Conversely, it remains open whether the $O\left(n^2 \log n\right)$ communication complexity is optimal or not; the only known lower bound on this parameter is still $O(n)$, as the one for transfer rate [68]. It seems to us that a communication complexity lower than $O\left(n^2\right)$ is unlikely to be achievable, at least not without a completely different approach to the problem.

4.2 Generalization of PSMT to Linear Combinations of Errors and Eavesdropped Data

We discuss in this subsection a generalization of Perfectly Secure Message Transmission issued from Network Coding. We begin by examining PSMT from a slightly more abstract point of view: if we forget about the n channels connecting Alice and Bob and only look at the input and output of the involved parties, we can see PSMT as a scenario where Alice and Bob send vectors of length n to each other, and where an adversary Eve controls a subset $\mathcal{A} \subseteq \{1, \ldots, n\}$ of t coordinates, meaning that every time a string $\mathbf{x}$ is transmitted, she acquires $\mathbf{x}_i$ for $i \in \mathcal{A}$ and she can replace $\mathbf{x}_i$ with a symbol of her choice.

The scenario we consider now gives greater power to Eve, as she can eavesdrop linear combinations of the transmitted symbols and inject linear combinations of errors from a set of her choice. The formal definition is as follows:

Definition 4.2.1 (**Generalized Model**). *The protocol is specified by positive integers m, n, t, r and a finite field $\mathbb{F}_q$. The protocol involves two users, Alice and Bob, and an adversary Eve, who chooses t "eavesdropping vectors" $\boldsymbol{\lambda}^{(1)}, \ldots, \boldsymbol{\lambda}^{(t)} \in \mathbb{F}_q^n$ and t "tampering vectors" $\boldsymbol{\mu}^{(1)}, \ldots, \boldsymbol{\mu}^{(t)} \in \mathbb{F}_q^n$.*

Alice and Bob can send to each other vectors $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ where $\mathbf{x}_i \in \mathbb{F}_q^m$; whenever $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is transmitted (either from Alice to Bob or from Bob to Alice), the following happens:

- *Eve learns the value of*

$$\boldsymbol{\lambda}^{(i)}\mathbf{x}^T = \boldsymbol{\lambda}_1^{(i)}\mathbf{x}_1 + \cdots + \boldsymbol{\lambda}_n^{(i)}\mathbf{x}_n$$

  for every $i = 1, \ldots, t$.

- *Eve selects t columns vectors $\boldsymbol{\Delta}^{(1)}, \ldots, \boldsymbol{\Delta}^{(t)} \in \mathbb{F}_q^m$, depending on all the data she acquired so far; the intended receiver gets the message $\mathbf{y} = \mathbf{x} + \mathbf{e}$, where*

$$\mathbf{e} = \sum_{i=1}^{t} \boldsymbol{\Delta}^{(i)} \otimes \boldsymbol{\mu}^{(i)} = \sum_{i=1}^{t} \left(\boldsymbol{\Delta}^{(i)}\boldsymbol{\mu}_1^{(i)}, \ldots, \boldsymbol{\Delta}^{(i)}\boldsymbol{\mu}_n^{(i)} \right) .$$

We assume that Alice holds secret messages $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(\ell)} \in \mathbb{F}_q^m$; a protocol that specifies which vectors Alice and Bob should communicate with each other is deemed secure *if the following two properties are satisfied for any choice of secrets and of eavesdropping and tampering vectors:*

- Privacy: *the data eavesdropped by Eve has a distribution which does not depend on the secrets $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(\ell)}$.*

- Reliability: *Bob is always able to recover $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(\ell)}$ from the data he sent and received.*

Notice that standard PSMT can be seen as a more restrictive version of this model, where Eve is forced to choose vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ of weight 1, and where moreover $\boldsymbol{\lambda}^{(i)} = \boldsymbol{\mu}^{(i)}$ for every i.

The following subsections are organized as follows. First, we show in Section 4.2.1 that the generalization of Definition 4.2.1 is not gratuitous, since such a setting models interaction in Secure Network Coding scenarios; we thus give an overview of the topic and see discuss what the the state-of-the art protocols achieve.

We then contribute to the topic in two ways. First, we show in Section 4.2.2 that Protocol 4.1.1 can be adapted to provide security in the setting of Definition 4.2.1; more precisely, the only relevant modification we have to perform is the following: instead of classical codes, which can correct from errors with bounded Hamming weight, we use *rank* codes which can correct from errors with bounded rank; notice that such are the errors introduced by the adversary in Definition 4.2.1.

Then, in Section 4.2.3 we detail a three-round protocol that also works in a setting with multiple receivers (i.e., multiple "Bobs"); again, we make extensive use of the coding-theoretic techniques for PSMT presented in Section 4.1.

4.2.1 Motivation: Secure Network Coding

We discuss in this section the notion of Secure Network Coding, and show how it can be modeled by Definition 4.2.1.

A *network* is a directed, acyclic and connected multigraph $\mathcal{G}$ with *source* nodes (having in-degree 0) and destination nodes (with out-degree 0); for simplicity, we will only consider networks with a single source node. The *connectivity* (or *min cut*) C of a network $\mathcal{G}$ is defined as the minimum number of edges whose removal disconnects the source node from the destination nodes.

A network can be used to transmit information from the source to the destination nodes: in *Network Routing,* this is done by simply allowing each node to read the data received via the incoming edges and send it via the outbound edges (the source node will simply produce and send data). In contrast to this approach, *(Linear) Network Coding* allows nodes to perform operations on the received data, and to send the results over the outbound edges.

More precisely, the source node will produce messages $\mathbf{x} := (\mathbf{x}_1, \cdots, \mathbf{x}_n)$, where each $\mathbf{x}_i$ belongs to the alphabet $\mathbb{F}_q^m$ for a finite field $\mathbb{F}_q$; elements of $\mathbb{F}_q^m$ are called *packets*. Each node will compute linear combinations over $\mathbb{F}_q$ of the packets received via the input edges (or of the inputs $\mathbf{x}_i$ in the case of the source node) and send the results on the outbound edges; these linear combinations form the *network code*. We assume that the code is *feasible*, i.e. that each destination node can compute $\mathbf{x}$ as a linear function of the received packets.

It is well-known that a feasible network code for $\mathcal{G}$ exists if q is sufficiently large and $n \leq C$ (where C denotes the connectivity of the network), while no feasible network code exists if $n > C$ [2, 47, 43].

Secure Network Coding adds an adversarial component to this setting: for the sake of clarity, assume that the source node is controlled by a *sender* (who

decides which message $\mathbf{x}$ the node will produce) and that each destination node is controlled by a *receiver,* that can read the data received by the node. A first adversarial model for this situation was introduced by Cai and Yeung [10], where an adversary Eve can eavesdrop on t edges of the network (see also [30] and [29]); subsequently, Silva and Kschischang [63] introduce a more general adversarial model, where the adversary Eve gains full control of t edges of her choice, meaning she can read the symbols transmitted over these edges and replace them by symbols of her choice.

In case there is a single receiver, we can identify the sender with Alice and the receiver with Bob. It is then readily seen that the communication from Alice to Bob is affected by the adversary precisely as in Definition 4.2.1: each edge i under Eve's control carries a linear combination $\boldsymbol{\lambda}_1^{(i)}\mathbf{x}_1 + \cdots + \boldsymbol{\lambda}_n^{(i)}\mathbf{x}_n$ of the input packets $\mathbf{x}_1, \ldots, \mathbf{x}_n$, where the coefficients $\boldsymbol{\lambda}_j^{(i)}$ depend on the network code; similarly, any error $\boldsymbol{\Delta}^{(i)}$ that Eve injects in edge i propagate to the networks, so that eventually the output value would be affected by a linear combination $\sum_{i=1}^{t} \boldsymbol{\Delta}^{(i)} \otimes \boldsymbol{\mu}^{(i)}$ of the errors, where again the coefficients $\boldsymbol{\mu}_j^{(i)}$ depend on the network code. As a worst-case scenario, we may assume that Eve is free to select which subset of edges to control; this is modeled by letting her choose the eavesdropping and tampering vectors in Definition 4.2.1.

Furthermore, if the network also allows for communication from the receiver to the sender, then the adversary again affects the communication as in Definition 4.2.1. One has to be careful, however, since our generalized adversary is "symmetric", i.e. his eavesdropping and tampering vectors do not change for the communication from Alice to Bob and from Bob to Alice; this is not necessarily the case for a Network-Coding scenario. Nevertheless, the protocols we present actually do not require this symmetry of the adversarial powers, and can thus be used to provide security in a Network-Coding scenario, as long as communication is also possible from the receiver to the sender.

Hence the protocols that we present in the following sections provide the following contribution to Secure Network Coding:

- the generalization of Protocol 4.1.1 can be used to provide security for a Network Coding scenario with a single receiver, as long as communication is also possible from the receiver to the sender; notice that though Network Coding was originally introduced in a multicast scenario [2, 47], it has since been proved useful in single sender – single receiver scenarios as well [48, 70].
- the three-round protocol we present also works in a multicast scenario.

We stress the fact that just as the most recent works on Secure Network

Coding [63], both our protocols are *universal,* namely they are defined and secure regardless of the network code.

We remark that Jaggi et al. [42] studied a similar case, where the adversary controls vertices instead of edges of the network; their protocol lets the adversary inject up to $n/2$ errors, but with a weaker notion of security in that it must drop the privacy requirement and the reconstruction process admits a positive error probability.

How feedback improves security. Until very recently, existing work on Secure Network Coding assumes that information can only flow from sender to receiver. Notably, the work of Silva and Kschischang [63] presents a one-round protocol that is secure as long as $t < n/3$; this is in fact optimal if no feedback is allowed.

On the other hand, both our protocols are secure for any $t < n/2$, thanks to the possibility to convey data in both senses. Notice that two-ways communication has strengthened resistance against the adversary also in alternative information-theoretic scenarios, such as PSMT (which we covered in Section 4.1) and Secret Key Agreement [52] (where the sender and a single receiver communicate over a noisy channel in the presence of a wiretapper).

We give here some intuition on how two-ways communication strengthens resistance in our case. A common approach for communication in Secure Network Coding is to use rank-metric codes, a variant of block codes that can correct from errors of limited rank; the properties of rank-metric codes mirrors those of classical codes: for instance, we can speak of minimum distance d, a code can be corrected from errors of rank at most $\lfloor (d-1)/2 \rfloor$, and the Singleton bound still holds.

Assume that communication is only possible from the sender to the receivers, and assume that a codeword $\mathbf{x} \in \mathcal{C}$ is sent by the sender to the receivers, for a given (rank-metric) code $\mathcal{C}$. Then to ensure that the receivers can recover $\mathbf{x}$, $\mathcal{C}$ must have minimum rank-distance at least $2t+1$; furthermore, to prevent Eve from recovering $\mathbf{x}$, $\mathcal{C}$ must have dimension at least $t+1$. Thus $d_{\min}(\mathcal{C}) + \dim(\mathcal{C}) \geq 3t+2$; by the Singleton bound, we thus obtain $3t < n$.

Now if feedback is allowed, the situation improves: the feedback information can contain critical information on the errors introduced by the adversary (namely, the pseudo-basis of the received values); this basically allows us to recover the support of the errors and hence only requires the code $\mathcal{C}$ to have minimum distance at least $t+1$ for correction. We thus obtain security for any $t < n/2$.

4.2.2 The Two-Round Protocol

We present in this section a secure two-round protocol in the setting of Definition 4.2.1; as we have discussed above, this is achieved by simply adapting Protocol 4.1.1 to a rank-metric setting. We start by adapting the machinery needed for Protocol 4.1.1:

Rank-Metric Codes for Communication. We discuss here a special type of codes, defined under the *rank metric* [24], [31]. These have been extensively used in Secure Network Coding [62], [63].

Definition 4.2.2 (**Rank-Metric Code**). *Let m, n be positive integers, and let q be a prime power. Given the $\mathbb{F}_q$-linear space $\mathbb{F}_q^{m\times n}$, we can define the* rank distance *between its elements by letting $d_R(\mathbf{x}, \mathbf{y}) := \mathrm{rank}(\mathbf{y} - \mathbf{x})$.*

A rank-metric *code $\mathcal{C}$ is a non-empty subset of $\mathbb{F}_q^{m\times n}$ with induced rank distance; by identifying the field $\mathbb{F}_{q^m}$ with $\mathbb{F}_q^m$, we can view $\mathcal{C}$ as a code over $\mathbb{F}_{q^m}^n$, and require it to be* linear *over $\mathbb{F}_{q^m}$; we can hence speak of* block length *and* dimension *of such a code (as in the Hamming case) and of* minimum (rank) distance*, and combine these parameters into the triplet $[n, k, d]_{q^m}$.*

Just as the Hamming case, we have that a rank-metric code of minimum distance d can correct from errors of rank-weight at most $\lfloor (d-1)/2 \rfloor$.

The equivalent concept of MDS in this setting is called *Maximum Rank-Distance*: a rank-metric code is Maximum Rank-Distance (or *MRD* for short) if $k + d = n + 1$; an MRD code of arbitrary dimension k and length n exists if and only if $m \geq n$ [24]. More precisely, for any q and any m, n, k with $m \geq n$, we can construct a *Gabidulin code* [31] of parameters $[n, k, n - k + 1]_{q^m}$.

As in the Hamming case, we can express a rank-metric code $\mathcal{C}$ in term of a *parity-check* matrix $\mathbf{H}$, i.e.

$$\mathcal{C} = \left\{ \mathbf{x} \in \mathbb{F}_{q^m}^n \,:\, \mathbf{H}\mathbf{x}^T = \mathbf{0} \right\}.$$

Furthermore, we can define the associated *syndrome* map $\sigma : \mathbf{w} \mapsto \mathbf{H}\mathbf{w}^T$; clearly, we have that $\mathcal{C} = \ker(\sigma)$.

We will use the following fundamental property to transition between Hamming and rank-metric codes for communication:

Proposition 4.2.1 ([62]). *Let $\mathcal{C}$ be a MRD code of parameters $[n, k, \dots]$ over $\mathbb{F}_{q^m}$; let $\mathbf{H}$ be a parity-check matrix of $\mathcal{C}$ (so that $\mathbf{H}$ is an $(n-k)$-by-n matrix with coefficients over $\mathbb{F}_{q^m}$) and let $\mathbf{B}$ be a full-rank k-by-n matrix with coefficients over $\mathbb{F}_q$. Then the square matrix*

$$\left(\frac{\mathbf{H}}{\mathbf{B}}\right)$$

is nonsingular over $\mathbb{F}_{q^m}$*.*

As a first building block, we construct a rank-metric equivalent of Lemma 4.1.1: we show how to define a rank-metric code $\mathcal{C}$ and a vector $\mathbf{h}$ such that the adversary has no information of the value of $\mathbf{h}\mathbf{x}^T$ if a random $\mathbf{x} \in \mathcal{C}$ is transmitted between Alice and Bob. Here the key point is that the vector $\mathbf{h}$ has entries in $\mathbb{F}_{q^m}$ rather than in $\mathbb{F}_q$.

Lemma 4.2.2. *For any prime power q and any positive integers n, $m \geq n+1$ and $t < n$ there exists an MRD code $\mathcal{C}$ of parameters $[n, t+1, n-t]_{q^m}$ and a vector $\mathbf{h} \in \mathbb{F}_{q^m}^n$ such that the following holds for any t eavesdropping vectors $\boldsymbol{\lambda}^{(1)}, \cdots, \boldsymbol{\lambda}^{(t)} \in \mathbb{F}_q^n$:*

if $\mathbf{x} \in_{\$} \mathcal{C}$ is uniformly distributed, then $\mathbf{h}\mathbf{x}^T$ is uniformly distributed over $\mathbb{F}_q^m$ and is independent of $\boldsymbol{\lambda}^{(1)}\mathbf{x}^T, \ldots, \boldsymbol{\lambda}^{(t)}\mathbf{x}^T$.

This means that if a random $\mathbf{x} \in \mathcal{C}$ is transmitted from Alice to Bob or vice versa, then $\mathbf{h}\mathbf{x}^T$ is uniformly distributed and independent of Eve's eavesdropped data.

Proof. We construct $\mathcal{C}$ and $\mathbf{h}$ by adapting the blueprint of Lemma 4.1.1 to the rank-metric setting: hence we first let $\mathcal{C}'$ be an MRD code of parameters $[n+1, t+1, n-t+1]$, and let $\mathcal{C}$ be the code obtained by puncturing $\mathcal{C}'$ at the first coordinate, i.e.

$$\mathcal{C} := \left\{\mathbf{x} \in \mathbb{F}_{q^m}^n : \exists\, x \in \mathbb{F}_{q^m} \text{ with } (x, \mathbf{x}) \in \mathcal{C}'\right\}$$

Again, since puncturing can decrease the minimum rank distance by at most 1, $\mathcal{C}$ is MRD of parameters $[n, t+1, n-t]$ as requested.

We then define $\mathbf{h}$ by selecting a parity-check matrix $\mathbf{H}'$ of $\mathcal{C}'$, and choosing a row thereof of the form

$$(1, \mathbf{h}) \in \mathbb{F}_{q^m}^{n+1} \text{ with } \mathbf{h} \in \mathbb{F}_{q^m}^n.$$

Notice that such a row exists (up to multiplying $\mathbf{H}'$ with a non-zero scalar of $\mathbb{F}_{q^m}$) since $\mathcal{C}'$ has minimum distance $n - t + 1 > 1$. Now assuming without loss of generality that the vectors $\boldsymbol{\lambda}^{(1)}, \ldots, \boldsymbol{\lambda}^{(t)}$ are $\mathbb{F}_q$-linearly independent, we then have that the square matrix

$$\mathbf{M} := \left(\begin{array}{c|c} \multicolumn{2}{c}{\mathbf{H}'} \\ \hline 0 & \boldsymbol{\lambda}^{(1)} \\ \vdots & \vdots \\ 0 & \boldsymbol{\lambda}^{(t)} \\ \hline -1 & \mathbf{0} \end{array}\right)$$

is non-singular thanks to Proposition 4.2.1. This means that for any $\mathbf{b} \in \mathbb{F}_{q^m}^t$ and any $s \in \mathbb{F}_{q^m}$, there exists a unique $\hat{\mathbf{x}} \in \mathcal{C}$ with $\boldsymbol{\lambda}^{(1)}\mathbf{x}^T = \mathbf{b}_1, \ldots, \boldsymbol{\lambda}^{(t)}\mathbf{x}^T = \mathbf{b}_t$ and $\mathbf{h} \cdot \mathbf{x}^T = s$.

In other words, we have that

$$p\left(\left(\begin{array}{c} \boldsymbol{\lambda}^{(1)} \\ \vdots \\ \boldsymbol{\lambda}^{(t)} \end{array}\right) \cdot \mathbf{x}^T = \mathbf{b},\, \mathbf{h}\mathbf{x}^T = s\right) = 1/|\mathcal{C}| \text{ for any } \mathbf{b} \text{ and } s\,.$$

By using the law of total probability, it is thus easily seen that the claim holds.

□

In a symmetric fashion, we now present a rank-metric version of the broadcast protocol, hence allowing for reliable communication of messages, although with no guarantee of privacy.

Lemma 4.2.3. *Given any prime power q and positive integers n, m, t with $n \leq m$ and $n \geq 2t+1$, let $\mathcal{C}_{\texttt{BCAST}}$ be an MRD code of parameters $[n, 1, n]_{q^m}$. Then if an arbitrary $\mathbf{x} \in \mathcal{C}_{\texttt{BCAST}}$ is transmitted between the players, the receiver can always recover $\mathbf{x}$ from the received message $\mathbf{y}$ by computing the closest codeword to $\mathbf{y}$.*

Proof. By Definition 4.2.1, the receiver obtains a message $\mathbf{y}$ with $\mathbf{y} = \mathbf{x} + \mathbf{e}$, where the error $\mathbf{e}$ introduced by Eve is of rank at most t. Hence the original codeword $\mathbf{x}$ can be recovered since $\mathcal{C}_{\texttt{BCAST}}$ has rank-distance $n \geq 2t+1$.

□

Syndrome-spanning subsets. We show that the machinery of the pseudo-basis, or syndrome-spanning subset, can be adapted to the rank-metric case; we begin with the equivalent of Lemma 4.1.3:

Lemma 4.2.4. *Let $\mathcal{C}$ be a rank-metric code of parameters $[n, k, d]_{q^m}$; let $\mathbf{H}$ be a parity-check matrix of $\mathcal{C}$, and let $W \leq \mathbb{F}_{q^m}^n$ be an $\mathbb{F}_{q^m}$-linear subspace with*

the property that each $\mathbf{w} \in W$ *is of rank at most* $d-1$ *over* $\mathbb{F}_q$*. Then the syndrome map* $\sigma : \mathbf{w} \mapsto \mathbf{H}\mathbf{w}^T$ *is injective on* W*.*

Proof. Assume that $\sigma(\mathbf{w}) = \mathbf{0}$ for some $\mathbf{w}$ in W. By definition, we then have that $\mathbf{w}$ belongs to $\mathcal{C}$; hence since $\mathcal{C}$ has minimum rank-distance d and since $\mathbf{w}$ has rank at most $d-1$ by assumption, we have that $\mathbf{w} = \mathbf{0}$.

This means that $\ker\left(\sigma_{|W}\right) = \{\mathbf{0}\}$, so that $\sigma_{|W}$ is injective.

□

We then show how the concept of pseudo-basis and its main property can be transposed to the rank-metric case; for the rest of this section, we assume that a rank-metric code $\mathcal{C}$ of parameters $[n, k, d]_{q^m}$ has been chosen, together with a parity-check matrix $\mathbf{H}$ and associated syndrome map σ.

Definition 4.2.3. *Let* $\mathcal{Y}$ *be a set of matrices of* $\mathbb{F}_q^{m\times n}$*; a* pseudo-basis *of* $\mathcal{Y}$ *is a subset* $\mathcal{W} \subseteq \mathcal{Y}$ *such that* $\sigma(\mathcal{W})$ *is a basis of the syndrome subspace* $\langle\sigma(\mathcal{Y})\rangle$*.*

Notice that a pseudo-basis has cardinality at most $n-k$*, and that it can be computed in time polynomial in* n*.*

Proposition 4.2.5. *Let* r *be a positive integer, and let* $\mathcal{C}$ *be a linear rank-metric code of parameters* $[n, k, d]_{q^m}$*, and let* $\mathcal{X}$*,* $\mathcal{E}$*,* $\mathcal{Y}$ *be three subsets:*

- $\mathcal{X} := \left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(r)}\right\}$ *a set of codewords of* $\mathcal{C}$*,*
- $\mathcal{E} := \left\{\mathbf{e}^{(1)}, \ldots, \mathbf{e}^{(r)}\right\}$ *a set of error matrices such that rank*$(\mathbf{e}) \leq d-1$ *for all* $\mathbf{e} \in \langle\mathcal{E}\rangle_{\mathbb{F}_{q^m}}$*,*
- $\mathcal{Y} := \left\{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(r)}\right\} \subseteq (\mathbb{F}_q^m)^n$ *with* $\mathbf{y}^{(j)} = \mathbf{x}^{(j)} + \mathbf{e}^{(j)}$ *for every* j*.*

Then, given knowledge of $\mathcal{X}$ *and a pseudo-basis of* $\mathcal{Y}$*, we have that* $\mathbf{e}^{(j)}$ *can be computed from its syndrome* $\sigma(\mathbf{e}^{(j)})$ *for any* $1 \leq j \leq r$*.*

Proof. The hypothesis on the rank of the elements of $\langle\mathcal{E}\rangle$ implies that $\langle\mathcal{E}\rangle$ satisfies the hypothesis of Lemma 4.2.4 and the syndrome function is therefore injective on $\langle\mathcal{E}\rangle$. Given the pseudo-basis $\left\{\mathbf{y}^{(i)} : i \in I\right\}$, we can decompose any syndrome $\sigma(\mathbf{e}^{(j)})$ as

$$\begin{aligned}\sigma(\mathbf{e}^{(j)}) &= \sum_{i\in I} \lambda_i \sigma(\mathbf{y}^{(i)}) = \sum_{i\in I} \lambda_i \sigma(\mathbf{e}^{(i)}) \\ &= \sigma\left(\sum_{i\in I} \lambda_i \mathbf{e}^{(i)}\right)\end{aligned}$$

which yields

$$\mathbf{e}^{(j)} = \sum_{i \in I} \lambda_i \mathbf{e}^{(i)} = \sum_{i \in I} \lambda_i \left(\mathbf{y}^{(i)} - \mathbf{x}^{(i)} \right)$$

by injectivity of σ on E.

□

The Protocol. We define a two-round protocol for private and reliable communication in the setting of Definition 4.2.1, obtained by adapting Protocol 4.1.1 to the rank-metric case.

Protocol 4.2.1. *The protocol works in two rounds, and allows Alice to communicate ℓ secret elements $s^{(1)}, \cdots, s^{(\ell)}$ of $\mathbb{F}_q^m$ to Bob; we assume that $m > n$ and that $t < n/2$, where t denotes the number of eavesdropping and tampering vectors of the adversary.*

The protocol uses a pair $(\mathcal{C}, \mathbf{h})$ as in Lemma 4.2.2 for private communication, and an MRD code $\mathcal{C}_{\texttt{BCAST}}$ for reliable communication as in Lemma 4.2.3.

- Round 1: *Bob chooses $t+\ell$ uniformly random and independent codewords $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t+\ell)}$ of $\mathcal{C}$ and communicates them to Alice.*
- Round 2: *Alice receives the corrupted versions $\mathbf{y}^{(1)} = \mathbf{x}^{(1)} + \mathbf{e}^{(1)}, \ldots, \mathbf{y}^{(t+\ell)} = \mathbf{x}^{(t+\ell)} + \mathbf{e}^{(t+\ell)}$; she then proceeds with the following actions:*
  - *(i) She computes a pseudo-basis $\left\{ \mathbf{y}^{(i)} : i \in I \right\}$ for $I \subset \{1, \ldots, t+\ell\}$ of the received values, and uses $\mathcal{C}_{\texttt{BCAST}}$ to reliably communicate to Bob $\left(i, \mathbf{y}^{(i)} : i \in I \right)$.*
  - *(ii) She then considers the first ℓ words that do* not *belong to the pseudo-basis; to ease the notation, we will re-name them $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(\ell)}$. For each secret $s^{(j)}$ to be communicated she broadcasts to Bob the following two elements:*
    - *$\mathbf{H} \left(\mathbf{y}^{(j)} \right)^T$, the syndrome of $\mathbf{y}^{(j)}$;*
    - *$s^{(j)} + \mathbf{h} \left(\mathbf{y}^{(j)} \right)^T$.*

Proposition 4.2.5 guarantees that for any j, $1 \leq j \leq \ell$, Bob can compute the error vector $\mathbf{e}^{(j)}$ and hence reconstruct $\mathbf{y}^{(j)} = \mathbf{x}^{(j)} + \mathbf{e}^{(j)}$ from his knowledge of $\mathbf{x}^{(j)}$. He can therefore open the mask $\mathbf{h} \left(\mathbf{y}^{(j)} \right)^T$ and obtain the secret $s^{(j)}$.

Thanks to the generalization of the machinery of PSMT described above, the security of Protocol 4.2.1 can be easily proved by adapting the proof of Proposition 4.1.5. We thus obtain the following proposition:

Proposition 4.2.6. *Protocol 4.2.1 allows for private and reliable communication of ℓ elements of $\mathbb{F}_q^m$ in the setting of Definition 4.2.1.*

We now analyze the complexity of Protocol 4.2.1. First notice that its computational cost is clearly polynomial in n, m and q; we thus focus on the communication cost:

Communication complexity: we can set $\ell := 1$; furthermore, since MRD codes exist for any q and any $m > n$, we can assume that $q = 2$ and $m = O(n)$.

- Round 1 requires transmitting $t + 1$ codewords over the channels, thus requiring a total of $O\left(n^3\right)$ bits to be transmitted.
- Round 2-(i) requires broadcasting up to t vectors of $(\mathbb{F}_q^m)^n$, thus giving a total of $O\left(n^4\right)$ bits to be transmitted.
- Finally, Round 2-(ii) requires broadcasting a total of $t+1$ elements of $\mathbb{F}_q^m$ (a size-t syndrome and the masked secret), thus giving a total of $O\left(n^3\right)$ bits to be transmitted.

We thus obtain a total communication complexity of

$$O\left(n^4\right)$$

bits to be transmitted to communicate a single-bit secret.

Tranfer rate: optimal rate is achieved for $\ell = \Omega\left(n\right)$.

- Round 1 requires transmitting $t+\ell$ codewords, for a total of $O\left(n^2 + n\ell\right)$ elements of $\mathbb{F}_q^m$.
- Round 2-(i) remains unchanged from the single-bit case, and requires transmitting $O\left(n^3\right)$ elements of $\mathbb{F}_q^m$.
- Finally, Round 2-(ii) requires broadcasting a total of $\ell(t+1)$ elements of $\mathbb{F}_q^m$ (ℓ size-t syndromes and the masked secrets), thus giving a total of $O\left(n^2\ell\right)$ elements of $\mathbb{F}_q^m$;

To sum up, the overall transmission rate is equal to

$$\frac{O\left(n^2 + n\ell + n^3 + n^2\ell\right)}{\ell} = O\left(n^2\right).$$

4.2.3 The Three-Round Protocol

We discuss here a three-round protocol that provides security in a setting where a sender communicates with several receivers, and where an adversary can affect each of these communications as in Definition 4.2.1. We assume that Eve chooses the *same* eavesdropping vectors for each receivers, while the tampering vectors may change for each of them; this setting models a network-coding scenario with a single sender and several receivers.

We make use of the same core constructions of the previous section: rank-metric codes and pseudo-basis; the protocol works in three phases: a first communication round from sender to receivers, a feedback step, and a final communication phase.

Protocol 4.2.2. *The protocol works in three rounds, and allows the sender to communicate ℓ secret elements $\mathbf{s}^{(1)}, \cdots, \mathbf{s}^{(\ell)}$ of $\mathbb{F}_q^m$ to r receivers. The protocol uses a pair $(\mathcal{C}, \mathbf{h})$ where $\mathcal{C}$ is an MRD code of parameters $[n, t+1, n-t]_{q^m}$ and $\mathbf{h}$ is a vector of $\mathbb{F}_{q^m}^n$ as in Lemma 4.2.2. An MRD code $\mathcal{C}_{\texttt{BCAST}}$ of parameters $[n, 1, n]$ is also taken as input to perform broadcast.*

- Round 1: *The sender chooses $\ell + rt$ uniformly random and independent codewords $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(\ell+rt)}$ of $\mathcal{C}$ and communicates them to the receivers.*
- Round 2: *Each receiver obtains the corrupted versions $\mathbf{y}^{(1)} = \mathbf{x}^{(1)} + \mathbf{e}^{(1)}, \cdots, \mathbf{y}^{(\ell+rt)} = \mathbf{x}^{(\ell+rt)} + \mathbf{e}^{(\ell+rt)}$ (where the errors may depend on the receiver); each receiver then computes a syndrome-spanning subset $\left\{\mathbf{y}^{(i)} : i \in I\right\}$ of the received vectors and communicates $\left(I, \{i, \mathbf{y}^{(i)}\} : i \in I\right)$ to the sender via broadcast.*
- Round 3: *For each receiver, the sender computes the basis of the error space given by $\left\{\mathbf{e}^{(i)} = \mathbf{y}^{(i)} - \mathbf{x}^{(i)} : i \in I\right\}$ and broadcasts it to the receivers (again, bases for different receivers may be different).*

  The sender then consider the first ℓ words that do not *belong to any basis; to ease the notation, we will re-name them $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(\ell)}$. For each secret $s^{(i)}$ to be communicated he broadcasts to the receivers the element $\mathbf{s}^{(i)} + \mathbf{h}\left(\mathbf{x}^{(i)}\right)^T$.*

At this point, each receiver can use the basis of the error space to recover the codewords $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(\ell)}$, which in turn he can use to recover the elements $\mathbf{h}\left(\mathbf{x}^{(1)}\right)^T, \cdots, \mathbf{h}\left(\mathbf{x}^{(\ell)}\right)^T$; he can hence compute the masks that have been broadcast and recover the secrets.

Proposition 4.2.7. *Protocol 4.2.2 allows for secure communication of ℓ elements of $\mathbb{F}_q^m$ against an adversary as described in Definition 4.2.1.*

Proof. First notice that the protocol is well-defined since each syndrome-spanning subset has cardinality at most t: hence there are at least ℓ codewords outside all syndrome-spanning subsets since the total number of the codewords is $\ell + rt$.

We can now prove the privacy and reliability of the protocol:

- *Privacy:* the data acquired by the adversary is the following:

  I. $\boldsymbol{\lambda}^{(i)} \left(\mathbf{x}^{(1)}\right)^T, \ldots, \boldsymbol{\lambda}^{(i)} \left(\mathbf{x}^{(\ell+rt)}\right)^T$ for any $i = 1, \ldots, t$;

  II. $\left(\mathbf{y}^{(i)} : i \in I\right)$, the pseudo-basis (for each receiver);

  III. $\left(\mathbf{e}^{(i)} : i \in I\right)$, the corresponding basis of the error space (for each receiver);

  IV. $\mathbf{s}^{(1)} + \mathbf{h}\left(\mathbf{x}^{(j_1)}\right)^T, \ldots, \mathbf{s}^{(\ell)} + \mathbf{h}\left(\mathbf{x}^{(j_\ell)}\right)^T$ where the indexes j_i mark elements outside any pseudo-basis.

  Now thanks to the discussion in Section 4.2.2, privacy can be demonstrated by adapting the proof of Proposition 4.1.5.

- *Reliability:* since the second and third round of communication are performed via broadcast, we have that the syndrome-spanning subsets, the bases of the error spaces and the masked values are correctly received; now each receiver can use the knowledge of a basis of the error space to decode from errors of rank-weight $d - 1 = t$ for codewords of $\mathcal{C}$ (cf. Lemma 4.2.4. He can thus recover the codewords $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(\ell)}$ as requested, and in turn the elements $\mathbf{h} \cdot \left(\mathbf{x}^{(1)}\right)^T, \cdots, \mathbf{h} \cdot \left(\mathbf{x}^{(\ell)}\right)^T$ which mask as one-time pads the secret elements.

□

The computational cost of Protocol 4.2.2 is easily seen to be polynomial in n, m and q for the sender and for all receivers; furthermore, the protocol requires transmitting $O\left(n\ell + n^2 rt\right)$ elements of $\mathbb{F}_q^m$ to communicate ℓ secret elements of $\mathbb{F}_q$. This means that the communication complexity is equal to $O\left(n^4 r\right)$ bits and that the transfer rate is equal to $O\left(n\right)$.

Unicast and Multicast. We conclude this chapter by discussing how suitable our protocols are for multicast.

Protocol 4.2.1 has been presented in a unicast scenario, i.e. where a single receiver Bob is present. It would be possible to adapt it to a multicast scenario, but as far as we can see such a generalization would have little justification.

Indeed, assume that a sender is connected to r receivers in a network-coding situation, where an adversary controls t wires of the network; as discussed in Section 4.2.1, the communication between the sender and each receiver is then modeled by Definition 4.2.1, assuming that the connectivity C of the network is at least n. Under this assumption, by Menger's theorem each destination node is connected by n disjoint paths to the source node [8]; hence we could achieve secure and reliable communication by simply executing a Perfectly Secure Message Transmission protocol for each receiver.

The point is that this straightforward approach would be (roughly) as efficient as adapting Protocol 4.2.1; hence in such a general scenario, there seems to be little justification to perform such an adaptation.

Conversely, we have presented Protocol 4.2.2 (the three-round one) in a multicast scenario. This is because Protocol 4.2.2 actually outperforms the straightforward approach sketched above: indeed, with this approach a Perfectly Secure Message Transmission protocol is executed for each receiver. Now for an arbitrary network these executions cannot be done in parallel, since the connection paths could overlap. This implies that such a scheme, relying on the best known PSMT protocol, would be r times more expensive than Protocol 4.2.2 for a growing secret message size, where r denotes the number of receivers; recall in fact that Protocol 4.2.2 has a transfer rate of $O(n)$, independent of r. In fact, given the lower bounds on the complexity of PSMT [68], for $r > 2$ our protocol has smaller complexity than anything that can possibly be achieved with routing and PSMT.

In short, our three-step protocol preserves the fundamental property of network coding: complexity does not depend on the number of receivers.

Chapter 5

Improvements to the SPDZ Multi-Party Computation Protocol

This chapter is devoted to Secure Multi-Party Computation, or MPC for short, a scenario where several users wish to jointly evaluate a function on some input values without giving away the privacy of these inputs.

More specifically, we focus on the so-called "SPDZ" protocol by Damgård et al., named after the last names of its authors; our contribution is based on our article [65] with Serge Fehr. The chapter is organized as follows: Section 5.1 provides an introduction on Secure Multi-Party Computation and the SPDZ protocol, and summarizes our contribution to the topic; Section 5.2 gives an overview of the SPDZ protocol, highlighting the problem of the non-identifiable abort. Finally, Section 5.3 presents our protocol, which enhances the original SPDZ with dishonest players detection.

5.1 An Introduction to Secure Multi-Party Computation and the SPDZ Protocol

MPC, introduced by Yao [73] in 1982, assumes that several parties $P_1, \ldots, P_n$, usually referred to as *players,* hold private inputs $x_1, \ldots, x_n$ respectively; it is

further assumed that an adversary gains control of some of the players (who are then called *corrupted*). The adversary can be *passive,* meaning that he only reads the data acquired by corrupted parties, or *active,* meaning that he can decide the actions of corrupted players.

Informally stated, the players aim at computing the value $\varphi(x_1, \ldots, x_n)$ of a function φ on their inputs, while guaranteeing the correctness of the output and while keeping their inputs private. If we imagine that players can appeal to a trusted mediator, then the problem can be easily solved: players would simply send their inputs to the mediator via secure channels, the mediator would compute the value of $\varphi(x_1, \ldots, x_n)$ and communicate it back to all the players. The goal of an MPC protocol is to allows players to achieve the same outcome without any external mediator.

Whether MPC is possible or not depends on several parameters and on the exact security requirements of the protocol: for instance, relevant factors are the number of corrupted players, their cheating capabilities (whether they can deviate from the instructions of the protocol or not), their computing power, and the communication model.

Since the initial theoretical possibility results for MPC in the late eighties [73, 32, 6, 13], much effort has been put into reducing the communication and computation complexity of MPC, an important issue for the topic. We are now at a stage where MPC is at the verge of being practical.

One of the currently known protocols that is (close to) efficient enough for practical deployment is the so-called SPDZ protocol by Damgård et al. [23], and its variations from [22]. The efficiency of the SPDZ protocol is due to a clever mix of cryptographic operations, which can mostly be pushed into a preprocessing phase, and very efficient information-theoretic techniques.[1]

The SPDZ MPC protocol offers security against a dishonest majority, i.e., there is no bound on the number of corrupted players the protocol can tolerate: even if all but one of the players are corrupt, that one single party is still protected, meaning that his input will remain private and that he will not be lead to accept a false output. A downside of such protocols with security against a dishonest majority is that they are inherently susceptible to a "denial-of-service" attack: even *one single dishonest party* can enforce the protocol to fail, meaning that the honest parties have to abort the computation without learning the outcome, whereas the cheating party may actually learn it. Furthermore, the SPDZ MPC protocol is such that the cheating party who launches the attack remains covert: the (honest) parties know that there is a cheater among them that caused the protocol to fail, but they have no way to

[1] See Section 5.2.1 for a brief discussion over information-theoretic and computational security.

identify the culprit. As such, a single party can prevent the SPDZ protocol from doing its job.

Identifiable vs Non-identifiable Abort. We feel that such an non-identifiable abort, where the honest parties cannot identify the cheating party that caused the abort, is a drawback for practical deployment. In real-life scenarios, there are many reasons for why a party may be tempted to enforce the protocol to fail: he may know or suspect that he is not going to like the outcome, he may gain an advantage by learning the outcome but preventing the others from learning it, he may want to sabotage the computation out of malevolence, etc. And of course, if that party does not have to fear any consequence because he knows that he will not be caught, there is little incentive for him not to cheat. Furthermore, once a non-identifiable abort does take place, the affected honest parties are stuck – there is nothing they can do: they cannot call anyone to account, and re-trying the computation is (almost) useless, because the cheating party can just re-do the attack.

In contrast to this is the concept of *identifiable abort*, where it is required that, as a consequence of launching a denial-of-service attack, the cheating party will be identified as being the culprit. Obviously, for a protocol that offers identifiable abort, there is less incentive for a party to cheat and enforce the protocol to fail, because he knows that he will be caught and have to deal with the consequences. Furthermore, even if an abort does occur, the affected honest parties have room for further actions: not only can they call the cheating party to account, they can also re-do the computation with the culprit excluded, and this way they can still obtain the outcome of the computation eventually.[2]

We point out that non-identifiable abort is no issue in case of two-party computation: if the protocol fails then it is clear to the honest party that the other party must be cheating.

Our Contribution. We propose a new version of the SPDZ protocol that supports *identifiable* abort: if the protocol aborts then at least one dishonest party will be identified as having cheated. We emphasize that the challenge lies in adding identifiability to SPDZ without increasing its complexity too much; in particular, we want the protocol to run (almost) as fast as the original version in case parties do not misbehave (too much). This is what our protocol achieves.

[2] One has to be careful with this "solution" though: collaborating dishonest parties that remained passive during the first run may now adjust their inputs, given that they have learned the output from the first (failed) run.

- In case no cheating takes place, i.e., all the players behave honesty, our protocol is essentially as efficient as the original SPDZ protocol: namely, it has an asymptotic communication complexity of $O(n)$ point-to-point communications per gate and an asymptotical computational cost of $O(n)$ field operations per gate.

  We perform extra broadcasts compared to the original SPDZ protocol, but since their number is independent of the circuit size, this can be neglected for large enough circuits.

- In case cheating does take place, but to an extent that the protocol can handle it and does not abort, our protocol is slower by a factor at most 2, hence still with an asymptotic complexity of $O(n)$ per gate for both communication and computation.

  Again, the extra broadcasts can be neglected.

- In case cheating takes place and the protocol does abort (with identification), we distinguish between the following two cases (which case occurs depends on the kind of cheating):

  - *Identification with no agreement:* Every honest player has identified at least one player as a cheater, but there may not be agreement among the honest players about the list of cheaters.[3] In this case, our protocol is slower still by a factor 2 only.
  - *Identification with agreement:* There is common agreement among the honest players about at least one player being a cheater. In this case, our protocol may take substantially longer to identify the cheater, namely in this case the number of cryptographic operations to be performed grows with the size of the circuit.

Thus, the only case when our version is significantly slower than the original SPDZ protocol is when a dishonest player cheats so bluntly that he is publicly recognized as being a cheater. However, in many practical scenarios, there seems to be little gain for a dishonest player in slowing down the protocol at the cost of being publicly caught as a cheater, and thus having to face the consequences. Therefore, in typical scenarios, our protocol is similarly efficient as the original SPDZ protocol but, in contrast to the original version, it discourages dishonest players from enforcing the protocol to abort.

Related Work. Cheater detection is achieved by early MPC protocols such as [32], and by other protocols that are based on the paradigm that players

[3] But every player that is identified by an honest player to be a cheater *is* a cheater; thus, this case can only occur if there is more than one cheater.

prove in zero-knowledge that they followed the protocol instructions honestly. However, the high communication complexity of these protocols make them unsuitable for practical deployment.

On the other hand, recent MPC protocols (in a so-called offline/online model) are designed to have very high efficiency, like the protocols from the SPDZ family [23, 22], which feature a very attractive asymptotic communication and computational complexity of $O(n)$ per multiplication gate (for the online phase). However, these protocols do not offer cheater detection. An earlier protocol by Bendlin et al. [7] offers a very weak form of cheater detection: namely, at least one honest player will identify a dishonest one, but other honest players may have no clue on the identity of cheating parties; the protocol has a computational complexity of $O(n^2)$ per multiplication gate.

The work by Ishai et al. [41] is the first to rigorously define and discuss the notion of cheater detection (in the universal-composability model of Canetti [11]); the article presents a general compiler that adds cheater detection to any semi-honest MPC protocol in the preprocessing model.

A very recent protocol, due to Baum et al. [4], builds up on the Bendlin et al. approach and achieves full cheater detection with a communication and computational complexity of $O(n^2)$ per multiplication gate; this also improves on the best protocol obtained by means of the techniques by Ishai et al.

The goal of our work is to develop a MPC protocol that is "strictly stronger" than SPDZ, in that when not under attack it has the same running time than SPDZ, and when under attack it either gives away cheaters or the protocol can handle the attack and still has the same (asymptotic) running time than SPDZ. This is achieved by our protocol, but is not achieved by any of the above. Indeed, in case no severe cheating takes place, our protocol is at most a factor 2 slower than SPDZ, hence achieving a communication and computation complexity of $O(n)$ per multiplication gate. If cheating does take place to the extent that the protocol aborts, than either we obtain a weaker notion of cheater detection ("identification with no agreement") at the same cost, or we obtain the same notion ("identification with agreement"), but with an overhead in local computations.

5.2 The Standard SPDZ Protocol

We discuss in this section the original SPDZ protocol, thus providing the starting point for our construction. We begin by recalling the notions of information-theoretic and computational security (Section 5.2.1). In Section 5.2.2, we then present the setting and goal of SPDZ, highlighting how it is divided

into two phases; these are then discussed in the following two sections. Finally, Section 5.2.5 gives an idea on how the security of the second phase is guaranteed, and highlights the problem of the non-identifiable abort.

5.2.1 A Brief Discussion on Information-Theoretic and Computational Security

Before describing the details of the SPDZ protocol, we give discuss the two notions that are needed to model its security.

All the previous chapter of this dissertation presented protocols with *information-theoretic security:* namely, no assumptions are made on the computing power of the adversary, neither in space (i.e., the adversary is allowed to have unlimited memory) nor in time (i.e., he is allowed to run algorithms with arbitrary long running time).

In practical scenarios, however, it is safe to assume that the adversary is limited, either in space, or in time, or in both; several cryptographic protocols are secure only if this type of assumption is made (and under the postulate that some computational problem is unfeasible to solve with limited space/time computing power). We speak in this case of *computational security.*

In SPDZ, both these notions of security are found. However, computational security is (almost) only found in the preprocessing phase; hence since our focus is on the online phase, the informal definition of computational security given above is sufficient for our purposes.

5.2.2 Setting and Goal of SPDZ.

The function to be computed. The SPDZ protocol involves n players $P_1, \ldots, P_n$; each player P_i holds some inputs $x^{(1)}, \ldots, x^{(r)}$, where each $x^{(j)}$ lies in a finite field $\mathbb{F}_q$ and r depends on the player P_i.

The function to be evaluated at the input values is described in terms of an *arithmetic circuit* C over $\mathbb{F}_q$. An arithmetic circuit is a type of graph that describes the behaviour of a function in a step-by-step manner, and it is given by a finite, directed, acyclic and connected multigraph. The number of *input* gates (i.e., of vertices with in-degree 0) of the graph is equal to the total number of input values held by players; each of these gates is uniquely labeled with the identifier of an input value. We assume for simplicity that there exists a single *output* gate (with out-degree 0), and that its in-degree is equal to 1. Finally, all other gates have in-degree either 1 (which are labeled with an element from $\mathbb{F}_q$ and can either be *addition-with-constant* or *multiplication-by-constant* gates)

or 2 (which can be *sum* or *multiplication* gates).

By simply following the topology of the circuit, we obtain a polynomial function of the inputs held by players: the starting point is given by the input gates. Subsequently, sum and multiplication gates can be "evaluated" by adding or multiplying the two inbound values, and sending the result via the outbound edge to the following gate; multiplication-by-constant gates simply multiply the inbound value with the given constant in $\mathbb{F}_q$, and send the result to the following gate.

Eventually, the output gate will yield the result of a computation, which is the value of a polynomial function over $\mathbb{F}_q$ evaluated at the inputs held by players.

As hinted in Section 5.1, the goal of SPDZ is to allow players to evaluate the circuit C on their inputs while keeping privacy: more precisely, any coalition of up to $n-1$ dishonest players should learn nothing about the input value of the remaining player aside from what can be deduced from the outcome of the computation; and no honest player should accept an incorrect output. We speak of *privacy* and *correctness;* notice that we cannot achieve *fairness,* i.e. we cannot guarantee that honest players will learn the correct output of the computation.

We will further discuss security in Section 5.2.5; we now focus on further detailing the SPDZ protocol.

The communication model. A critical point for our discussion is how the communication between players is realized. More precisely, we assume that players can communicate with each other via secure point-to-point channels and via broadcast.

(Secure) point-to-point channels allow each player P_i to communicate with any other player P_j with perfect reliability and privacy, meaning that P_j will always correctly receive any message sent by P_i, while any other player will have no information on these messages.

Broadcast, on the other hand, allows each player P_i to communicate a message to all other players with perfect privacy *and* with consistency, meaning that all players will receive the same message. A player that broadcasts a value is modeled by an interactive algorithm `Alg` with the property that its output is the same for all players. The key point, of course, is that dishonest players are bound as well by the consistency property of broadcast: namely, we model dishonest players by replacing `Alg` with an arbitrary algorithm $\texttt{Alg}'$, subject nevertheless to the property that the output of $\texttt{Alg}'$ is the same for all players.

From a practical point of view, constructing a broadcast protocol from point-

to-point channels is expensive. For this reason, when computing the complexity of our protocol we keep a separate counter for point-to-point communication and for broadcasts, assuming that the latter will be more expensive. Notice anyway that providing a concrete instantiation of a broadcast protocol would be beyond the scope of this dissertation.

The commitment scheme. This technique is used to solve the following problem: assume that at a given point of the protocol, each player P_i has to reveal a value z_i that he holds; if this is simply done in turns, a malicious player could then "adjust" z_i depending on the previously revealed values. This may happen, for instance, when players want to reconstruct a shared value $[z]$: a malicious player could announce a suitably chosen false share so that the players will reconstruct a value $\tilde{z}$ of his choice.

A *commitment scheme* is an interactive protocol that allows players to solve this problem; we give here an informal definition of the variant we are interested in. Such a scheme allows each player P_i to produce a *commitment* $\texttt{Commit}(z_i)$ to a value z_i that he holds; later on, he can be asked to publicly *open* this commitment. The *binding* property of the scheme means that $\texttt{Commit}(z)$ can never open to z' for $z' \neq z$, while the *hiding* property means that it is computationally unfeasible to obtain any information on z from $\texttt{Commit}(z)$.

Thus in a situation when each player P_i has to reveal a value z_i, we can ask for each player in turn to broadcast a commitment $\texttt{Commit}(z_i)$, and then require all players to open their commitments. Thus players cannot adjust their value of z_i depending on the previously announced values.

We will see in Section 5.2.3 how the original SPDZ protocol is endowed with a commitment scheme.

The main ingredient: additive sharings. Following a standard paradigm for MPC over arithmetic circuits [5], SPDZ requires players to share their input values; each gate is then processed on the shares, and finally the shares of the output are reconstructed at the end of the circuit. We start by detailing the formulation of this sharing, before describing in more details how SPDZ works.

- A $[\cdot]$-*sharing* of a value $z \in \mathbb{F}_q$ is an additive sharing of z, meaning that each player P_i holds a random *share* $z_i \in \mathbb{F}_q$ subject to $\sum_i z_i = z$. This is denoted by $[z] = (z_1, \cdots, z_n)$.

Furthermore, to ensure correctness, every shared value is accompanied by a sharing of an authentication tag for the shared value; this is formalized as follows:

- For an arbitrary but fixed $\alpha \in \mathbb{F}_q$, a $\langle\cdot\rangle_\alpha$-*sharing* of z consists of $[\cdot]$-sharings of z and of $\alpha \cdot z$, i.e., $\langle z\rangle_\alpha = \big([z], [\alpha \cdot z]\big)$. The element α is called the *global key*, and αz is called the *tag* of z and usually denoted by $\gamma(z)$. If α is clear from the context, we may write $\langle\cdot\rangle$ instead of $\langle\cdot\rangle_\alpha$.

We say that a sharing $[z]$ or a sharing $\langle z\rangle_\alpha = \big([z], [\gamma(z)]\big)$ is *privately opened* to a player P_i if each player P_j sends his share z_j to P_i via a point-to-point channel and P_i computes $z := \sum_j z_j$. We say that a sharing is *publicly* opened if it is privately opened to a designated "king player" P_k, and then P_k sends the reconstructed value z to all the players via point-to-point channels.[4]

Note that (for a fixed global key α) a $\langle\cdot\rangle_\alpha$-sharing is linear, in the sense that linear combinations can be computed *on the shares*:

$$\begin{aligned}\langle z + w\rangle = \langle z\rangle + \langle w\rangle &:= \big([z_i + w_i]_{i=1,\cdots,n},\ [\gamma(z)_i + \gamma(w)_i]_{i=1,\cdots,n}\big)\\ \langle \lambda z\rangle = \lambda \cdot \langle z\rangle &:= \big([\lambda z_i]_{i=1,\cdots,n},\ [\lambda\gamma(z)_i]_{i=1,\cdots,n}\big)\,.\end{aligned}$$

Furthermore, if α is $[\cdot]$-shared then the same holds for addition with a constant:

$$\langle \lambda + z\rangle = \quad \lambda + \langle z\rangle \quad := \big([\lambda + z_1, z_2, \cdots, z_n], [\lambda\alpha_1 + \gamma(z)_1, \cdots, \lambda\alpha_n + \gamma(z)_n]\big)\,.$$

Finally, a triple $\big(\langle a\rangle_\alpha, \langle b\rangle_\alpha, \langle c\rangle_\alpha\big)$ is called a *multiplication triple* if it consists of three $\langle\cdot\rangle_\alpha$-shared random values a, b, c subject to $ab = c$.

SPDZ is divided into a *offline* (or *pre-processing*) phase, and an *online* phase. The idea is to push most of the (somewhat) expensive cryptographic techniques into the offline phase (which can be executed *before* the inputs to the computation – or even the actual computation – are known), and rely mainly on very efficient information-theoretic techniques in the online phase. We describe these two phases in the following sections.

5.2.3 The Pre-Processing Phase

The Pre-processing phase produces the following *shared randomness:*

- a $[\cdot]$-sharing $[\alpha]$ of a random and unknown global key α,
- a list of $\langle\cdot\rangle_\alpha$-sharings $\langle r\rangle_\alpha$ of random and unknown values r, and

[4] We emphasize that, by definition, these private and public openings do *not* involve any checking of the correctness of z by means of its tag; this will have to be done on top.

- a list of multiplication triples $\left(\langle a\rangle_\alpha, \langle b\rangle_\alpha, \langle c\rangle_\alpha\right)$ with random and unknown $a, b, c = ab$.

Concretely, these elements are produced via a *(somewhat) additive-* and *(somewhat) multiplicative-homomorphic* encryption scheme `Enc`; formally defining `Enc` and its properties would be beyond the scope of this thesis, but we discuss here some properties which are relevant for our construction. `Enc` takes as input a "plaintext" value z and some randomness ρ_z, and produces a cyphertext $e_z = \texttt{Enc}(z, \rho_z)$. The key point is that if even though `Enc` is injective on its first coordinate (i.e. $\texttt{Enc}(z, \rho_z) \neq \texttt{Enc}(z', \rho'_z)$ as long as $z \neq z'$), the value of a cyphertext e_z yields no information on the plaintext z for a computationally-bounded party. This has the following two consequences:

- `Enc` can be used as a commitment scheme (cf. the previous section): indeed, each player P_i can compute a commitment on some value z_i that he holds by selecting a uniformly random ρ_{z_i} and then setting $\texttt{Commit}(z_i) := \texttt{Enc}(z_i, \rho_{z_i})$; opening this commitment requires P_i to broadcast both z_i and ρ_{z_i}, so that players can check that indeed $\texttt{Commit}(z_i) = \texttt{Enc}(z_i, \rho_{z_i})$.

  Notice that the binding property of the scheme follows from the fact that `Enc` is injective on its first coordinate, while he hiding property follows from the the fact that `Enc` is an encryption function.

- As a sort of "side product" of the generation of the above sharings with the help of `Enc`, the following is given at the end of the offline phase for every $[\cdot]$-sharing $[z] = (z_1, \ldots, z_n)$ that occurs as (first or second) component of a $\langle\cdot\rangle_\alpha$-sharing (as well as for the $[\cdot]$-sharing $[\alpha]$).

  The commitment $e_{z_i} := \texttt{Enc}(z_i, \rho_{z_i})$ of each share z_i is *public,* and player P_i knows the corresponding randomness ρ_{z_i}. Recall that `Enc` is additively-homomorphic, so that linear combinations (and addition with constants) can be computed on the commitments; this will form the basis of our commitment check (cf. Section 5.3.6).

5.2.4 The Online Phase

The actual computation takes place in the online phase. By using the sharings produced in the offline phase as a resource, the online phase can be executed to a large extent by means of very efficient information-theoretic techniques – the number of cryptographic operations (broadcasts, commitments) needed is independent of the circuit size. Concretely, the online phase is composed of the following gadgets.

- *Input sharing:* For each input x held by a player P_i, a fresh (meaning: yet unused) sharing $\langle r \rangle_\alpha$ from the offline phase is selected and privately opened to P_i. P_i then sends $\varepsilon := x - r$ to all the players, and altogether the players can then compute a sharing of x as $\langle x \rangle_\alpha = \varepsilon + \langle r \rangle_\alpha$.

- *Distributed addition (and multiplication/addition with constants):* For each addition gate in the circuit with shared inputs $\langle z \rangle_\alpha$ and $\langle y \rangle_\alpha$, a sharing of $z+y$ is computed (non-interactively) as $\langle z+y \rangle_\alpha = \langle z \rangle_\alpha + \langle y \rangle_\alpha$. Correspondingly for multiplication/addition with a constant.

- *Distributed multiplication:* For each multiplication gate in the circuit with shared inputs $\langle z \rangle_\alpha$ and $\langle y \rangle_\alpha$, a sharing of $z \cdot y$ is computed (interactively) by means of the multiplication subprotocol below, which consumes one fresh multiplication triple from the offline phase.

- *Output reconstruction*: For each shared output value $\langle z \rangle_\alpha$, the players publicly reconstruct z.

- *Tag checking:* For a shared value $\langle z \rangle_\alpha = ([z], [\gamma(z)])$ that has been publicly opened, the players can check the correctness of z as follows. Every player P_i computes $y_i := \gamma(z)_i - z \cdot \alpha_i$ and broadcasts a commitment of y_i, and then every player opens the commitment and the players compute $y := \sum_i y_i = \gamma(z) - z \cdot \alpha$. If $y = 0$ then z is declared to be correct; otherwise, it is declared incorrect and the protocol is aborted.

Multiplication subprotocol
A fresh multiplication triple $\big(\langle a \rangle, \langle b \rangle, \langle c \rangle\big)$ is selected, and the following is performed.

1. The players compute $\langle \varepsilon \rangle := \langle z - a \rangle$ and $\langle \delta \rangle := \langle y - b \rangle$.
2. The sharings $\langle \varepsilon \rangle$ and $\langle \delta \rangle$ are publicly opened:
   - $\langle \varepsilon \rangle$ and $\langle \delta \rangle$ are privately opened to a designated king player P_k, and
   - P_k sends ε and δ to the others player via the point-to-point channels.
3. The players compute $\langle z \cdot y \rangle := \langle c \rangle + \varepsilon \langle b \rangle + \delta \langle a \rangle + \varepsilon\delta$.

5.2.5 The Security of SPDZ and Its Cost

We give here an overview of how SPDZ achieves security, highlighting the problem of the lack of cheaters detection; our goal is to give a high-level description of this aspect of SPDZ, and the reader should refer to the original paper(s) for a formal discussion.

Security is achieved in a rather straightforward way: once the output of the circuit has been publicly reconstructed, its tags are checked; the same goes for any other value that has been opened during the protocol (e.g., the ε's and δ's of multiplication subprotocols). The SPDZ protocol offers security against a *static* adversary, namely an adversary that chooses which players to control before the execution of the protocol (although the online phase actually does not require this, and is thus secure against an *adaptive* adversary).

The idea is that even if the adversary controls $n-1$ players, the key α is a uniformly random element of $\mathbb{F}_q$ from his point of view; hence an incorrect value will fail the tag check except with probability $1/q$.

A downside of the protocol is that even a single dishonest player can easily enforce the protocol to abort, e.g., by submitting an incorrect share for a sharing $\langle z \rangle_\alpha$ that is publicly opened and then checked; the check will recognize (with high probability) that the reconstructed value z is incorrect, and so the protocol will abort, but there is no way for the honest players to find out *who* submitted an incorrect shares. We thus speak of *non-identifiable abort.*

Hence, any such dishonest player gets away with it, and hence there is no incentive for a dishonest player not to cheat, should it give him any advantage or satisfaction whatsoever. Our goal is to add dishonest players detection to the SPDZ protocol, while keeping essentially the same complexity.

Complexity of SPDZ. We give here a brief overview on the complexity of the online phase of SPDZ; we assume for simplicity that each player has a constant (in n and $|C|$) number of inputs to be shared.

It can then easily be proved that the online phase of SPDZ has a total computation cost of $O\left(n \cdot |C| + n^3\right)$ elementary field operations over $\mathbb{F}_q$, and a communication cost of $O\left(n \cdot |C| + n^3\right)$ elements of $\mathbb{F}_q$ to be communicated via point-to-point channels.

5.3 Adding Cheater Detection

We propose a new version of the SPDZ protocol that supports *identifiable* abort: if the protocol aborts then at least one dishonest party will be identified as having cheated. We emphasize that the challenge lies in adding identifiability to SPDZ without increasing its complexity too much; in particular, we want the protocol to run (almost) as fast as the original version in case parties do not misbehave (too much).

We organize our discussion as follows. In Section 5.3.1 we provide an overview of our protocol. Section 5.3.2 discuss an important building block of our construction, namely the sub-protocol that ensures correct execution of (part of) the circuit; in Section 5.3.4 we discuss how to secure the input-sharing and output-reconstruction phases, and all pieces are put together in Section 5.3.5, which proves the security of our protocol and computes its complexity. Finally, in Section 5.3.6 we briefly discuss how to use the commitments to in-agreement identify a dishonest player.

5.3.1 An Overview of The New Protocol

We explain on a high level how our protocol works. First, notice that there are four distinct ways for dishonest players to disrupt the protocol execution (and enforce an abort in the original SPDZ protocol):

- During the input sharing phase, dishonest players could send incorrect shares of r to P_i, or P_i could send inconsistent values ε to the players.[5]
- During the multiplication step, dishonest players could send incorrect shares of ε and δ to the king player.
- During the multiplication step, a dishonest king player could send false and/or inconsistent values for ε and δ.
- In the output reconstruction phase, dishonest players could announce false shares of the output.

We will focus on the two possible attacks in the multiplication step, since our techniques to deal with those can easily be used to also deal with the attacks in the input sharing and output reconstruction phases.

[5] Note that there is no issue of ε being *incorrect* since any ε corresponds to a possible input for P_i.

As pointed out above, the players have two "checking mechanisms" available in order to verify the correctness of a reconstructed value z:

- they can use the *tag* $\gamma(z)$ of z to check the correctness of z, and
- they can use the *commitments* to check the correctness of the shares z_i.

The first technique is very efficient but cannot be used to identify *who* submitted a false share in case of an incorrect z; this can be done by the latter, but that one is computationally more expensive, and so we want to avoid it as much as possible and use it only as a "last resort".

Now, a first and straightforward approach to achieve cheater detection but use the computationally expensive techniques only as a last resort, seems as follows: first, use the "cheap" tag checks to verify the correctness of every reconstructed value (as in the original SPDZ protocol), and then resort to the commitments if and only if an error is detected, in order to find out who claimed an incorrect value.

Unfortunately, this does not work; the reason is that only the king player knows the shares of, say, ε. As such, if ε claimed by the king player turns out to be incorrect, there is no way for an honest player to distinguish the case of a dishonest player P_i who has sent an incorrect share ε_i to the king player, from the case of a dishonest king player who pretends that he has received an incorrect share ε_i from P_i. There is no way such a dispute can be resolved, even with the help of the commitments – except if these shares are broadcast from the start, but that would greatly increase the complexity of the protocol.

To deal with such a situation, we use an idea from *dispute control*: we re-do (part of) the computation in such a way that this particular dispute cannot occur anymore (essentially by choosing a fresh king player). Since the number of disputes is bounded, this means that there is a limit on how often something needs to be re-done, and setting the parameters right ensures that this merely gives a factor-2 blowup.

On the other hand, if a dishonest player P_i keeps on claiming an incorrect share for, say, ε, even when the players are asked to *broadcast* their shares because a fault was detected, then the players can use the (computationally expensive) commitments to find the incorrect share, and the honest players will unanimously identify P_i as cheater.

The overall structure of the computation phase of our protocol is thus as follows:

Set-up: The circuit C is divided into consecutive blocks, each comprising ca.

$|C|/n$ gates (where "consecutive" here means that C can be evaluated in a block-by-block manner).

Furthermore, a list $\mathcal{L}_{\texttt{suspects}}$ of *suspect* players is initialized as the empty set.

Computation: Sequentially, for each block the following is done:

I. A king player $P_k \notin \mathcal{L}_{\texttt{suspects}}$ is selected, and the computation is done as in the normal SPDZ protocol by repeatedly invoking the multiplication sub-protocol and doing local computations.

II. Once the block has been processed, a checking protocol `BlockCheck` is invoked that verifies the correctness of the computation. `BlockCheck` has three possible outcomes:

- *Success:* The block has been correctly processed. In this case, the players simply move to the next block.
- *Fail with Conflict:* The block has *not* been correctly processed, and P_k accuses some player(s) of faulty behaviour. In this case, P_k and all accused players are added to $\mathcal{L}_{\texttt{suspects}}$, and the players go back to step I and re-do the computation with a "fresh" $P_k \notin \mathcal{L}_{\texttt{suspects}}$. Should $\mathcal{L}_{\texttt{suspects}}$ now consist of *all* players then the protocol aborts; in this case, every honest player has identified at least one dishonest player.
- *Fail with Agreement:* The block has *not* been correctly processed, and it is guaranteed that some player has *broadcast* an incorrect share during the run of `BlockCheck`. In this case, the players make use of the commitments to unanimously identify the cheating player.

The following sections give all the details.

5.3.2 The Checking Protocol `BlockCheck`

We now provide a formal discussion of the check-phase mentioned in the previous section; what it will do is check the correctness and consistency of all the ε's and δ's that were announced by the king player during the multiplication subprotocols in the block to be checked.

We assume that t multiplication opening values $\langle z^{(1)} \rangle, \cdots, \langle z^{(t)} \rangle$ have been publicly opened via a king player P_k, and we will use the following notation: for each shared value $\langle z^{(h)} \rangle$,

- each player P_j has sent $z_j^{(h)}$ to P_k;

– $\tilde{z}_j^{(h)}$ denotes the value received by P_k from P_j (so if P_j is honest, $\tilde{z}_j^{(h)} = z_j^{(h)}$);

– P_k has computed and sent to each P_j the value $z^{(h)}$;

– $\tilde{z}^{(h)}(j)$ denotes the value received by each P_j from P_k (so if P_k is honest, $\tilde{z}^{(h)}(j) = z^{(h)}$).

We first give an informal overview on how `BlockCheck` works:

- *Step 1:* players run a subroutine `Rand` to produce a random element e, and they compute the linear combination $\langle z \rangle := \sum_h e^h \langle z^{(h)} \rangle$.

  Let $\tilde{z}_i$ and $\tilde{z}$ be the respective linear combinations of $\tilde{z}_i^{(1)}, \cdots, \tilde{z}_i^{(t)}$ and of $\tilde{z}^{(1)}, \cdots, \tilde{z}^{(t)}$.

- *Step 2: Public Reconstruction.* Each player broadcasts his share of $\langle z \rangle$, upon which the king player P_k broadcasts a list of players that he accuses of inconsistent behaviour; if he does so, `BlockCheck` outputs the message "Fail with Conflict" and the list of accused players plus P_k.

  If P_k has not accused anybody, then each player can broadcast an accusation against P_k, stating that the value $\tilde{z}$ that he received is different from z (which is now public, since its shares have been broadcast). If that is the case, then once again we are in the "Fail-with-Conflict" case: `BlockCheck` outputs the corresponding error message and the list of players accusing P_k plus P_k himself.

- *Step 3: Tag Checking.* If no accusations have been produced in Step 2, then players check the tags of $\langle z \rangle = ([z], [\gamma(z)])$; this is done running a subroutine `ZeroTest` on $[\gamma(z)] - \tilde{z}[\alpha]$, which outputs $\top$ if it is a sharing of 0, and $\bot$ if it is not (except with small probability).

  `BlockCheck` outputs the message "Success" if the tag check succeeds, and "Fail with Agreement" if it fails.

The idea behind security is that if the tag check fails, then a dishonest player P_i must have broadcast a false share z_i during step 2, *or* (as we will see) he has broadcast some false share as part of the execution of `ZeroTest`; in either case, he has broadcast a linear combination (with coefficients that may depend on the $\tilde{z}^{(i)}$) of values he is committed to, by means of the commitments from the pre-processing phase and by the linearity of all computations. P_i can then be *publicly identified* as cheater by simply asking the players to open the commitments to the claimed and broadcast values (cf. Section 5.3.6). We note that checking the commitments causes a significant overhead to the efficiency

of the protocol because the players need to perform computations on a large number of commitments, proportional to the size of the circuit; in return, however, it publicly identifies a cheating player.

We now give all the details. First, we formally define the subprotocol `Rand`, which assumes that players have access to a commitment scheme (as in standard SPDZ - cf Sections 5.2.2 and 5.2.3).

`Rand`:

The protocol is used to generate a random seed $e \in_\$ \mathbb{F}_q$.

(i) Each player P_j selects a uniformly random $e_j \in_\$ \mathbb{F}_q$ and broadcasts a commitment $\texttt{Commit}(e_j)$ to it;

(ii) all the commitments are then opened, so that all players obtain $e_1, \cdots, e_n$;

(iii) the output of `Rand` is the value $e := \sum_{j=1}^{n} e_j$.

We now show that any error that occurred during the opening of the values $\langle z^{(1)} \rangle, \cdots, \langle z^{(t)} \rangle$ will affect their linear combination as well (with high probability):

Lemma 5.3.1 (**Security of `Rand`**). *Let e be a seed generated by `Rand`; consider the following linear combination with coefficients given by the powers of e:*

$$\langle z \rangle := \sum_{h=1}^{t} e^h \cdot \langle z^{(h)} \rangle, \quad \tilde{z}(j) := \sum_{h=1}^{t} e^h \cdot \tilde{z}^{(h)}(j) \text{ for any } j = 1, \cdots, n.$$

Assume that for a given index $h \in \{1, \cdots, t\}$ the value received by a given player P_j is incorrect, i.e. $\tilde{z}^{(h)}(j) \neq z^{(h)}$; then $\tilde{z}(j) \neq z$ except with probability t/q.

Similarly, if the values received by two players P_j and P_i for an index h are different (i.e. $\tilde{z}^{(h)}(j) \neq \tilde{z}^{(h)}(i)$), then the same will hold for the corresponding linear combinations, i.e. $\tilde{z}(j) \neq \tilde{z}(i)$ except with probability t/q.

Proof. Define $f(x) \in \mathbb{F}_q[x]$ to be

$$f(x) := \sum_h \left(z^{(h)} - \tilde{z}^{(h)}(j) \right) x^h;$$

following the definition of `Rand`, we have that $\tilde{z}(j) = z$ if and only if $f(e) = 0$, where f is non-zero polynomial of degree at most t. But e is a random element of $\mathbb{F}_q$ independent of the coefficients $z^{(h)} - \tilde{z}^{(h)}(j)$, so that

$$p(f(e) = 0) = \frac{|\text{roots of } f|}{|\mathbb{F}_q|} \leq \frac{t}{q}.$$

This proves the first part of the claim; the second part can be proved with a simple adaptation of this argument.

□

We then discuss the "public opening and conflict" phase, performed via the sub-protocol `PublicOpening`:

`PublicOpening`:

The protocol takes as input a shared value $[z]$ and the index k of the king player P_k; following the notation set at the beginning of the section, we assume that each P_j holds $z_j, \tilde{z}(j)$ and that P_k holds $(\tilde{z}_j : j = 1, \cdots, n)$.

Initialize the boolean value $\mathbf{b}$ to $\top$ and the list L to the empty set $\emptyset$.

(i) For each $j = 1, \cdots, n$, player P_j broadcast z_j and P_k broadcast $\tilde{z}_j$; if the two values do not coincide, set $\mathbf{b} = \bot$ and $L \leftarrow L \cup \{P_j\}$.

(ii) If $\mathbf{b} = \bot$, the protocol stops and output $(\bot, L)$.

(iii) Players set $\tilde{z} := \sum_j z_j$; for each $j = 1, \cdots, n$, player P_j broadcasts $\tilde{z}(j)$. If this value is different from $\tilde{z}$, set $\mathbf{b} = \bot$ and $L \leftarrow L \cup \{P_j\}$.

(iv) The protocol outputs $(\mathbf{b}, L, \tilde{z})$.

The following lemma is a direct consequence of the definition of the algorithm, and states that the public opening routine is correct and sound:

Lemma 5.3.2 (**Security of** `PublicOpening`)**.** *Let* $(\mathbf{b}, L, \tilde{z})$ *be the output of* `PublicOpening` $([z])$ *with king player* P_k*; we then have the following properties:*

- (correctness) *if* $\langle z \rangle$ *has been correctly reconstructed and players follow the instructions of the protocol, then* $\mathbf{b} = \top$ *and* $L = \emptyset$*.*

- (soundness) *if* $\tilde{z}(j) \neq \tilde{z}(i)$ *for some honest players* P_j *and* P_i*, then* $\mathbf{b} = \perp$ *and* $L \neq \emptyset$.

  Furthermore, in this case either P_k *or all players in* L *are dishonest.*

The last step consists in checking the tags of the value $\langle z \rangle = ([z], [\gamma(z)])$, which is performed by using the subroutine `ZeroTest` to check that the value $[\gamma(z)] - \tilde{z}[\alpha]$ opens to 0.

Notice that the players cannot just do a public reconstruction of the sharing $[\gamma(z)] - \tilde{z}[\alpha]$ to check whether it is a sharing of zero, because in case it is not, the value of $\gamma(z) - \tilde{z}\alpha$ reveals information on α. That is why the slightly more involved subroutine `ZeroTest` is invoked, which publicly reconstructs a *random multiple* of $[\gamma(z)] - \tilde{z}[\alpha]$.

Now the security of `ZeroTest` relies on the secrecy of the global key α, which we measure as follows:

Definition 5.3.1 (**Guessing Probability**). *We measure the secrecy of a random variable* $x \in_\$ \mathbb{F}_q$ *(typically, the global key* α*) as follows: let* v *denote the adversary's* view *at a given point in the online protocol, i.e. the random variable modeling all the data acquired by the adversary at that point.*[6]

Then, the adversary's (average) guessing probability *of* x *is given by*

$$p_{\text{guess}}(x|v) := \sum_{\hat{v}} p(v = \hat{v}) \cdot \max_{\hat{x}} p(x = \hat{x} | v = \hat{v}) \,.$$

We then introduce the following definition to model the information on x possessed by the adversary:

Definition 5.3.2 (**List Distribution**). *Given an abstract pair of random variables* (x, v)*, we say that the distribution of* x *given* v *is* a list of size m *if there exists a (conditional) distribution* $p(\ell|v)$*, where the range of* ℓ *consists of lists of* m *elements in the range of* x*, such that the following two properties hold for the joint distribution* $p(x, v, \ell) := p(x, v) \cdot p(\ell|v)$*:*

(I) $p(x \in \ell) \leq \max_{\hat{\ell} \in Im(\ell)} p(x \in \hat{\ell})$*;*

(II) $p(x|v = \hat{v}, \ell = \hat{\ell}, x \notin \hat{\ell}) = p(x|x \notin \hat{\ell})$ *for every* $\hat{v}$*,* $\hat{\ell}$ *such that the formula is well-defined.*

[6] Here and below, when we make information-theoretic statements, we understand v to *not* include the encryptions/commitments of the honest parties shares etc. that were produced during the preprocessing phase. Adding these elements to the adverary's view of course renders the information-theoretic statements invalid, but has a negligible effect with respect to a computationally bounded adversary.

In a nutshell, we use the above definition to formalize the following situation: let v denote the adversary's view and $x := \alpha$; assume that the distribution of α given v is a list of size m. This means that the adversary has tried to guess the value of α for m times, and he has learned whether his guess was correct or not after each guess.

We now state the basic properties of `ZeroTest`, which will in turn imply the desired properties of the tag check; we assume that `ZeroTest` outputs a boolean value $\mathbf{b} \in \{\top, \bot\}$, marking whether the input opens to zero or not, and some extra data that will be omitted in the following lemma.

Lemma 5.3.3 (**Security of** `ZeroTest`)**.** *let* $\mathbf{b}$ *be the output of* `ZeroTest` $([x])$*; we then have the following properties:*

- Correctness: *if* $x = 0$ *and players follow the instructions of the protocol, then* $\mathbf{b} = \top$ *with probability 1.*
- Soundness: *consider the joint distribution* $p(x, v_0)$*, where* v_0 *denotes the adversary's view before the execution of* `ZeroTest`*. Then*

  $$p(\mathbf{b} = \top) \leq 1/q + p_{\text{guess}}(x|v_0).$$

  Furthermore, if $x = 0$ *but* $\mathbf{b} = \bot$*, then a dishonest player has broadcast an incorrect version of a value to which he is committed by means of a linear combination of the commitments produced in the preprocessing phase.*
- (privacy)*: Assume that* x *is uniformly distributed and that the distribution of* x *given* v_0 *is a list of size* m_0*. Then after the execution of* `ZeroTest`$([x])$*, the distribution of* x *given* v *is a list of guesses of size at most* $m := m_0 + 1$*, where* v *denotes the adversary's view after the execution of* `ZeroTest`*.*

Now that we have fixed the notation for the subroutines, we can state the definition of `BlockCheck` in a formal way:

`BlockCheck`:

The protocol takes as input a block and the index k of the king player P_k; denote by $\langle z^{(1)} \rangle, \cdots, \langle z^{(t)} \rangle$ the multiplication opening values of the block.

- *Step 1:* players execute `Rand` to get a random seed $e \in \mathbb{F}_q$, then compute the linear combination $\langle z \rangle := \sum_{h=1}^{t} e^h \langle z^{(h)} \rangle$.

- *Step 2:* run $\left(\mathbf{b}, L, \tilde{z}\right) \leftarrow$ `PublicOpening` $([z])$; if $\mathbf{b} = \perp$, `BlockCheck` stops and outputs the message "Fail with Conflict" together with the list $L \cup \{P_k\}$.
- *Step 3:* run `ZeroTest` $([\gamma(z)] - \tilde{z}[\alpha])$, and denote by $\mathbf{b}$ its output.

  If $\mathbf{b} = \top$, `BlockCheck` outputs the message "Success";

  if $\mathbf{b} = \perp$, it outputs the message "Fail with Agreeement".

We can now prove the following security properties of `BlockCheck`:

Proposition 5.3.4 (**Security of** `BlockCheck`)**.** `BlockCheck` *satisfies the following:*

- Correctness: *if all players behave honestly and hence all* $\tilde{z}^{(j)}$ *are correct and consistently announced by* P_k*, then* `BlockCheck` *outputs "Success" with probability 1.*
- Soundness: *if at least one of the* $\tilde{z}^{(j)}$ *is incorrect, i.e.* $\neq z^{(j)}$*, or inconsistently announced by* P_k*, then the following holds except with probability at most*

$$\delta = (2|C|/n + 1)/q + p_{\text{guess}}(\alpha|v)\,,$$

  where v *is the adversary's view before the execution of* `BlockCheck`*.* `BlockCheck` *outputs "Fail"; furthermore, if it outputs "Fail with Conflict", then either the king player* P_k *or all of the accusing players are dishonest (or both), and if it outputs "Fail with Agreement", then all* $\tilde{z}^{(j)}$ *have been consistently announced by* P_k*, and a dishonest player has broadcast as part of* `BlockCheck` *an incorrect version of a value to which he is committed by means of a linear combination (depending on the* $\tilde{z}^{(j)}$*'s) of the commitments produced in the pre-processing phase.*

Proof.
- Correctness: obvious by the correctness of `PublicOpening` and `ZeroTest`.
- Soundness: first assume that there exists an index h such that $\tilde{z}^{(h)}(j) \neq \tilde{z}^{(h)}(i)$ for two honest players P_j and P_i; then thanks to Lemma 5.3.1, $\tilde{z}(j) \neq \tilde{z}(i)$ except with probability t/q, where $t \leq 2|C|/n$ since each block contains at most $|C|/n$ gates. Hence thanks to Lemma 5.3.2, `PublicOpening` will output $(\perp, L)$ with $L \neq \emptyset$, so that by definition the output of `BlockCheck` will be "Fail with Conflict". The properties on L follow from Lemma 5.3.2.

Now assume that there is consistency among the values but at least one of them is incorrect, i.e. there is an h such that $\tilde{z}^{(h)} \neq z^{(h)}$. Then by Lemma 5.3.1, $\tilde{z} \neq z$ except with probability at most $2|C|/nq$; now assuming that `PublicOpening` did not output $\perp$ (in which case we end up in the "Fail-with-Conflict" case), `ZeroTest` is executed. The input of this subprotocol is equal to $[\gamma(z)] - \tilde{z}[\alpha] = (z - \tilde{z})[\alpha]$, so that by Lemma 5.3.3 $p(\texttt{ZeroCheck}((z-\tilde{z})[\alpha]) = \top) \leq 1/q + p_{\text{guess}}((z-\tilde{z})\alpha|v)$, where v denotes the adversary's view before the execution of of `ZeroTest`. Since $z \neq \tilde{z}$, we have that $p_{\text{guess}}((z - \tilde{z})\alpha|v) = p_{\text{guess}}(\alpha|v)$, so that `BlockCheck` will output "Fail with Agreement" except with probability δ.

Finally, notice that in the "Fail-with-Agreement" case we have two possibilities: the first one is that $\tilde{z} \neq z$, which means necessarily at least a dishonest player submitted a false share of z; now this share will be a linear combination of values he is committed to with coefficients that depend on the multiplication opening values used so far (hence, possibly including the $\tilde{z}^{(h)}$'s). Since shares need to be broadcast during `PublicOpening`, the dishonest player will thus be committed to a wrong public value as claimed. The second possibility is that $\tilde{z} = z$ but $\texttt{ZeroTest}([\gamma(z)] - \tilde{z}[\alpha]) = \perp$, in which case the claim follows by Lemma 5.3.3.

□

We discuss in the next section how the tag checking is executed.

5.3.3 The Tag Check

We discuss here the sub-routine `ZeroTest`, meant to check whether some shared value $[x]$ is equal to zero or not. A straightforward way to do this would be to simply open $[x]$; however, in the actual scenario this value will be equal to $[\gamma(z)] - \tilde{z}[\alpha]$ for some shared value $\langle z \rangle$, and the adversary could select any non-zero value Δz and let $\tilde{z} = z + \Delta z$, so that opening $[\gamma(z)] - \tilde{z}[\alpha] = \Delta z \cdot [\alpha]$ would actually let the adversary learn the global key α. This is no problem in a stand-alone execution of the protocol, since this action would lead to a commitment check (and thus to the identification of a cheater without any need for the key α); however, we find it desirable to avoid leakage of the secret key. In this way, α and and the pre-processing material can still be used in, for instance, the execution of another instance of the protocol. We thus devise a sub-routine that does not leak the value of α.

`ZeroTest`:

The protocol takes as input a shared value $[x]$.

(i) Players select a random shared value $\langle r \rangle$ and a fresh multiplication triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$.

(ii) Players compute $[rx]$ with multiplication triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ as described in Section 5.2.4, but with a different communication model: instead of sending their data to a king player that acts as a relay, they will broadcast a commitment to it, then open all the commitments before moving to the next round.

(iii) Each player P_j broadcasts a commitment $\texttt{Commit}((rx)_j)$ to his share of $[rx]$, then all commitments are opened, so that players obtain rx.

(iv) Output $\top$ if $rx = 0$, $\perp$ otherwise.

We first prove that the subprotocol is correct and sound:

Lemma 5.3.5 (Correctness and Soundness of `ZeroTest`). `ZeroTest` *satisfies the following properties:*

- Correctness: *if players follow the instructions of the protocol,* $\texttt{ZeroTest}([0]) = \top$ *with probability 1.*
- Soundness: *consider the joint distribution* $p(x, v_0)$*, where* v_0 *denotes the adversary's view before the execution of* `ZeroTest`*; then*

$$p(\texttt{ZeroTest}([x]) = \top) \leq 1/q + p_{\text{guess}}(x|v_0).$$

Furthermore, if $x = 0$ *but* $\mathbf{b} = \perp$*, then a dishonest player has broadcast an incorrect version of a value to which he is committed by means of a linear combination of the commitments produced in the pre-processing phase.*

Proof. - Correctness: trivially, `ZeroTest` will open $[r \cdot 0] = [0]$.

- Soundness: by definition of the protocol, the output of $\texttt{ZeroTest}([x])$ is equal to $\top$ if and only if $\mathbf{b} = 0$, where

$$\mathbf{b} := (r - \tilde{r})(x - \tilde{x}) - \tilde{y}$$

where r is a variable uniformly distributed and independent of $x, \tilde{x}, \tilde{r}$ and $\tilde{y}$, and the variables $\tilde{r}, \tilde{x}$ and $\tilde{y}$ are chosen by the adversary, and are thus determined by his current view (since we can assume without loss of generality that the adversary is deterministic).

By the law of total probability, we have the following equality:

$$p\big((r-\tilde{r})(x-\tilde{x})-\tilde{y}=0\big) = \sum_{\hat{v}_0} p(v_0=\hat{v}_0)\cdot p\big((r-\tilde{r}(\hat{v}_0))(x-\tilde{x}(\hat{v}_0)) = \tilde{y}(\hat{v}_0)|v_0=\hat{v}_0\big)$$

We focus on the element $p((r-\tilde{r}(\hat{v}_0))(x-\tilde{x}(\hat{v}_0)) = \tilde{y}(\hat{v}_0)|v_0=\hat{v}_0)$; we can assume that $\tilde{y}(\hat{v}_0)=0$, since this clearly yields the highest probability. Notice that

$$\begin{aligned} p\big((r-\tilde{r}(\hat{v}_0))(x-\tilde{x}(\hat{v}_0))=0|v_0=\hat{v}_0\big) &\leq p\big(r=\tilde{r}(\hat{v}_0)|v_0=\hat{v}_0\big) + p\big(x=\tilde{x}(\hat{v}_0)|v_0=\hat{v}_0\big) \\ &\leq 1/q + \max_{\hat{x}} p\big(x=\hat{x}|v_0=\hat{v}_0\big) \end{aligned}$$

This implies that

$$\begin{aligned} p\big((r-\tilde{r})(x-\tilde{x})-\tilde{y}=0\big) &\leq \sum_{\hat{v}_0} p(v_0=\hat{v}_0)\cdot\left(\frac{1}{q}+\max_{\hat{x}} p\big(x=\hat{x}|v_0=\hat{v}_0\big)\right) \\ &\leq 1/q + \sum_{\hat{v}_0} p(v_0=\hat{v}_0)\cdot\max_{\hat{x}} p\big(x=\hat{x}|v_0=\hat{v}_0\big) \\ &= 1/q + p_{\text{guess}}(x|v_0) \end{aligned}$$

Finally, notice that since `ZeroTest` only uses broadcast for communication, all incorrect values submitted by players are now *public;* hence if $x=0$ but $\mathbf{b}=\perp$, then a dishonest player must have submitted a false value during the multiplication of r and x, and thus he will be exposed when the commitments on these values are checked.

□

Finally, we need to discuss the privacy of `ZeroTest`; we first remark that Definition 5.3.2, formalizing our privacy notion, yields the following consequences:

Remark 5.3.1. Assume that the *uniform* distribution x is a list of size m given v; we then have the following properties:

(i) $p(x \in \ell) \leq m/q$ (immediate consequence of (I));

(ii) $p\big(x = y | x \notin \ell\big) \leq 1/(q-m)$ for any $y = y(v)$ (consequence of (II) via the law of total probability).

Furthermore, let r be a random variable independent of both v and x, and set $v' := (v, r)$. Then it trivially holds that if $p(x, v)$ satisfies the above definition, then so does $p(x, v')$.

Lemma 5.3.6 (**Privacy of** `ZeroTest`)**.** *Given an abstract pair of random variables (x, v_0), where v_0 denotes the adversary's view, assume that the* uniform *distribution of x given v_0 is a list of size m_0. Then after the execution of* `ZeroTest`$([x])$*, the distribution of x given v is a list of guesses of size at most $m := m_0 + 1$, where v denotes the adversary's view after the execution of* `ZeroTest`*.*

Proof. By looking at the instructions to compute and open $[xr]$ to P_i, we see that what the adversary can learn the following values (plus random sharings of them): $\gamma := x - a$, $\delta := r - b$ and $\pi := (r - \tilde{r})(x - \tilde{x})$, where a, b and r are jointly uniformly distributed and independent of each other and of $v, x, \tilde{x}, \tilde{r}$. $\tilde{x}$ and $\tilde{r}$ are chosen by the adversary, and are thus determined by his view (since we assume without loss of generality that the adversary is deterministic).

Now given the adversary's view v_0 before the execution of `ZeroTest`, the adversary's *current* view is equal to $(v_0, \gamma, \delta, \pi)$; notice that a and b are (jointly) random and independent of x, r, v_0 and π, and thus so are $\gamma = x - a$ and $\delta = r - b$, so that we may restrict the view to $v := (v_0, \pi)$ (cf. Remark 5.3.1)[7].

Now by inductive hypothesis, there exists a conditional distribution $p(\ell_0 | v_0)$ such that properties I and II hold for $p(x, v_0, \ell_0) := p(x, v_0) \cdot p(\ell_0 | v_0)$; in a natural way, we define the new distribution to be

$$p\big(\ell = (x_1, \cdots, x_{m_0}, x_m) | v\big) := p\big(\ell_0 = (x_1, \cdots, x_{m_0}) | v_0\big) \cdot p\big(\tilde{x}(v_0) = x_m | v_0\big).$$

Clearly, elements in the range of ℓ are lists of size $m = m_0 + 1$ of elements in the range of x.

We now prove that properties I and II hold for $p(x, v, \ell)$: first of all, notice that $p\left(x \in \ell\right) = p\left(x \in \ell_0\right) + p\left(x = \tilde{x}(v_0) | x \notin \ell_0\right) \cdot p\left(x \notin \ell_0\right)$; hence thanks to properties (i) and (ii) of the Remark 5.3.1 we have that

[7] For the same reason, we omit here the fact that the view also contain random sharings of $x - a$, $r - b$ and π.

$$\begin{aligned} p(x \in \ell) &= p(x \in \ell_0) + p\big(x = \tilde{x}(v_0) | x \notin \ell_0\big) \cdot p(x \notin \ell_0) \\ &\leq m_0/q + p\big(x = \tilde{x}(v_0) | x \notin \ell_0\big) \cdot (1 - m_0/q) \\ &\leq m_0/q + \frac{1}{q - m_0} \cdot \frac{q - m_0}{q} \\ &= (m_0 + 1)/q = m/q \end{aligned}$$

Hence property I holds; we can thus focus on property II. As a first step, notice that if $x \notin \hat{\ell}$, then in particular $x \neq \tilde{x}(\hat{v}_0)$; hence we can re-write $\pi = \hat{\pi}$ as $r = \tilde{r}(v_0) + \hat{\pi}/(x - \tilde{x}(\hat{v}_0))$. Hence since r is independent of x, v_0 and ℓ_0, we obtain the following equalities:

$$\begin{aligned} p\big(x | (v_0, \pi) = (\hat{v}_0, \hat{\pi}), \ell = \hat{\ell}, x \notin \hat{\ell}\big) &= p\big(x | v_0 = \hat{v}_0, \ell_0 = \hat{\ell}_0, x \notin \hat{\ell}_0, x \neq \tilde{x}(\hat{v}_0)\big) \\ &= p\big(x | x \notin \hat{\ell}_0, x \neq \tilde{x}(\hat{v}_0)\big) \end{aligned}$$

which means that property II holds.

□

We discuss in the next section how to securely share the inputs of the players and reconstruct the output of the circuit.

5.3.4 Secure Input Sharing and Output Reconstruction

We show in this section how to secure the input-sharing and output-reconstruction phases; we use the main ideas and techniques of the multiplication check.

We first describe in more detail how the input sharing is performed in the original SPDZ protocol: each shared value $\langle r \rangle$ produced in the pre-processing phase comes with another type of sharing, denoted by

$$[\![r]\!] := \Big([r], \big(\beta_i, \gamma(r)_1^i, \cdots, \gamma(r)_n^i\big)_{i=1,\cdots,n}\Big),$$

where each player P_i holds $r_i, \beta_i, \gamma(r)_1^i, \cdots, \gamma(r)_n^i$ and $r\beta_i = \sum_j \gamma(r)_i^j$ for any i; the idea is that each each player P_i holds the key β_i that can authenticate the other players' shares of r, given their tags $\gamma(r)_i^j$.

Now in classical SPDZ, whenever a player P_i holds input x, a random shared value $[\![r]\!]$ is selected; then each player P_j communicates r_j and $\gamma(r)_i^j$ to P_i,

who computes r and checks that $r\beta_i = \sum_j \gamma(r)_i^j$; P_i can then broadcast either an error message or the value $\varepsilon := x - r$. The input is then shared as $\langle r \rangle + \varepsilon$.

We add to this protocol our system of accusations; the commitment checks will be executed as a last resort.

`InputShare`:

The protocol is used to share an input x held by player P_i; a fresh king player P_k and a shared value $\langle r \rangle, [\![r]\!]$ are selected.

I. for each $j \neq i$, player P_j sends $(r_j, \gamma(r)_i^j)$ to P_k, who in turn communicates these elements to P_i.
   P_i then computes $y := r\beta_i - \sum_j \gamma(r)_i^j$.

II. If $y = 0$, P_i broadcasts $(\top, \varepsilon := x - r)$;
   players share x as $\langle r \rangle + \varepsilon$ and `InputShare` ends.

III. If $y \neq 0$, P_i broadcasts $\perp$.
   Then for each $j \neq i$, player P_j broadcasts $(r_j, \gamma(r)_i^j)$; the king player P_k broadcast a list L of players that he accuses of inconsistent behaviour.

   - if $L \neq \emptyset$, then `InputShare` outputs message "Fail with Conflict" together with the list $L \cup \{P_k\}$;
   - if $L = \emptyset$, then P_i can accuse P_k of inconsistent behaviour; if that is the case, `InputShare` output message "Fail with Conflict" together with the list $\{P_i, P_k\}$;
   - if $L = \emptyset$ but P_i does not accuse P_k of inconsistent behaviour, then `InputShare` outputs message "Fail with Agreement".

The following proposition follows from the definition of `InputShare` and proves that the protocol is secure:

Proposition 5.3.7 (**Security of** `InputShare`)**.** *Let x be an input held by player P_i, where we model x as a uniformly distributed random variable over $\mathbb{F}_q$. Then* `InputShare` *satisfies the following properties:*

- Correctness: *if players behave honestly,* `InputShare`(x) *produces no accusations and players obtain a $\langle \cdot \rangle$-sharing of x.*

- Soundness: *if $y \neq 0$, then* `InputShare` *outputs "Fail"; furthermore, if it outputs "Fail with Conflict", then either the king player or all of the accusing players are dishonest (or both), and if it outputs "Fail with Agreement", then all $(r_j, \gamma(r)_i^j)$ have been consistently announced by P_k. Then either one of these value is incorrect (and the corresponding player is committed to this value) or all values are correct and P_i is dishonest.*

- Privacy: *if P_i is honest, the adversary's guessing probability of x is equal to $1/q$.*

We now introduce an output-checking phase which makes use of the protocols introduced in the previous sections: it simply reconstructs the output, then checks its tag with `ZeroTest`. If an error is detected, it will be guaranteed that some players have broadcast a value they are committed to.

`OutputCheck`:

The protocol takes as input the shared value $\langle z \rangle$, output of the circuit.

I. Each player P_i broadcasts his share z_i of $[z]$.

II. Players set $\tilde{z} := \sum_i z_i$; they then run `ZeroTest` $([\gamma(z)] - \tilde{z}[\alpha])$. Denote by $\mathbf{b}$ its output;

   - if $\mathbf{b} = \top$, the protocol outputs message "Success" and the element $\tilde{z}$;
   - if $\mathbf{b} = \bot$, the protocol outputs message "Fail".

The following proposition proves that the protocol is correct and sound; its proof follows from Lemma 5.3.3

Proposition 5.3.8 (**Secuirty of** `OutputCheck`)**.** `OutputCheck` *satisfies the following properties:*

- Correctness: *if players submit the correct shares of $[z]$ and behave honestly during* `ZeroTest`*, then* `OutputCheck` *outputs "Success" and the correct value z;*

- Soundness: *assume that $\tilde{z} \neq z$ or that the adversary behaved dishonestly in the* `ZeroTest` *phase (which means that a dishonest player has broadcast an incorrect version of a value he is committed to). Then* `OutputCheck` *outputs "Fail" except with probability $1/q + p_{\text{guess}}(\alpha|v)$.*

In the concrete setting, this error probability will be equal to $1/q + 1/(q - 2n)$.

5.3.5 The Complete Protocol

In this section we formally define our protocol and argue its security. According to the outline given in Section 5.3.1, the protocols presented are combined as follows:

MPC Protocol

Set-up: the circuit C is divided into consecutive blocks, each comprising ca. $|C|/n$ gates (where "consecutive" here means that C can be evaluated in a block-by-block manner).

Furthermore, a list $\mathcal{L}_{\texttt{suspects}}$ of suspect players is initialized as the empty set.

Input Sharing: a king player $P_k \notin \mathcal{L}_{\texttt{suspects}}$ is selected, and each player P_i shares his input(s) via the subprotocol `InputShare`. This has three possible outcomes:

- *Success:* players share the next input with the same procedure, or move to the computation phase if all inputs have been shared.
- *Fail with Conflict:* `InputShare` outputs a list L of suspect players; set $\mathcal{L}_{\texttt{suspects}} \leftarrow \mathcal{L}_{\texttt{suspects}} \cup L$.

  If all players belong to $\mathcal{L}_{\texttt{suspects}}$, the protocol aborts; otherwise, a "fresh" king player P_k is selected, and players re-invoke `InputShare`.
- *Fail with Agreement:* the players make use of the commitments to unanimously identify the cheating player; in case all commitment checks are successfully executed, then player P_i is deemed dishonest and the protocols aborts.

Computation: Sequentially, for each block the following is done:

I. A king player $P_k \notin \mathcal{L}_{\texttt{suspects}}$ is selected, and the computation is done as in the normal SPDZ protocol by repeatedly invoking the multiplication sub-protocol and doing local computations.

II. Once the block has been processed, `BlockCheck` is invoked; this has three possible outcomes:

- *Success:* the players simply move to the next block.
- *Fail with Conflict:* `BlockCheck` outputs a list L of suspect players; set $\mathcal{L}_{\texttt{suspects}} \leftarrow \mathcal{L}_{\texttt{suspects}} \cup L$.
  If all players belong to $\mathcal{L}_{\texttt{suspects}}$, the protocol aborts; otherwise, a "fresh" king player P_k is selected, and players go back to step I.
- *Fail with Agreement:* it is guaranteed that some player has *broadcast* an incorrect share during the run of `BlockCheck`. In this case, the players make use of the commitments to unanimously identify the cheating player.

Output Reconstruction: At this point, the players have a well-defined sharing $\langle z \rangle$ of the output of the circuit; they thus run $\texttt{OutputCheck}(\langle z \rangle)$. This has two possible outcomes:

- *Success:* the output z is obtained by all players.
- *Fail:* it is guaranteed that some player has *broadcast* an incorrect share during the run of `OutputCheck`. In this case, the players make use of the commitments to unanimously identify the cheating player.

While the security of the input-sharing and output-reconstruction phases is quite straightforward, the analysis of the computation phase requires a more involved discussion.

Indeed, recall that the security of `BlockCheck` depends on the information over the global key α possessed by the adversary (cf. Proposition 5.3.4), and that this information may increase at every execution of `BlockCheck`. More precisely, we need to upper bound the adversary's guessing probability of α; as a first step, we upper bound the number of executions of `BlockCheck`:

Remark 5.3.2. During the MPC protocol, `BlockCheck` is run at most $2n$ times: indeed, notice that by definition the circuit C is divided into at most n blocks, so that `BlockCheck` is run at most n times excluding re-boots.

Now the number of re-boots of `BlockCheck` is easily seen to be at most n: indeed, a re-boot only occurs when `BlockCheck` outputs "Fail with Conflict"; now by the soundness property of `BlockCheck` (cf. Proposition 5.3.4), this will add at least a new player to $\mathcal{L}_{\texttt{suspects}}$, namely the previous king player, so that we can have at most n such reboots in total before $\mathcal{L}_{\texttt{suspects}}$ is "full" and

hence the MPC protocol aborts.

We can now upper bound the adversary's guessing probability of α;

Proposition 5.3.9. *Throughout the entire protocol, the adversary's guessing probability of α is bounded by*

$$p_{\text{guess}}(\alpha|v) \leq \frac{1}{q-2n} + \frac{2n}{q}.$$

Proof. It is easily seen that the adversary can increase his guessing probability only during the execution of `BlockCheck` or, more specifically, during `ZeroTest`. This, by definition of `BlockCheck`, is executed only when the value $\tilde{z}$ is consistent among players, so that its input is equal to $[x] := [\gamma(z)] - \tilde{z}[\alpha] = (z - \tilde{z})[\alpha]$. Hence we can assume as a worst-case scenario that $z \neq \tilde{z}$, so that the adversary's guessing probabilities of α and of x coincide.

Notice that at the beginning of the computation, the distribution of α given the adversary's view is a list of guesses of size 0 (cf.Definition 5.3.2); hence we can inductively apply Lemma 5.3.3, so that during the execution of the protocol the distribution of α given the adversary's view is a list of guesses of size at most $2n$ (recall that `BlockCheck`, and hence `ZeroTest`, is executed at most $2n$ times as shown in Remark 5.3.2).

Hence according to Definition 5.3.2, and given that α is uniformly distributed, there exists a distribution $p(\ell|v)$ with the following properties:

(I) $p(\alpha \in \ell) \leq 2n/q$;

(II) $\max_{\hat{\alpha},\hat{\ell}} p(\alpha = \hat{\alpha}|v = \hat{v}, \ell = \hat{\ell}, \alpha \notin \hat{\ell}) = 1/(q-m)$.

Now from this we can deduce the claimed upper bound on the guessing probability: indeed, by using the law of total probability with the events $(\alpha \in \ell)$ and $(\alpha \notin \ell)$, we obtain

$$
\begin{aligned}
p_{\text{guess}}(\alpha|v) &= \sum_{\hat{v}} p(v=\hat{v}) \cdot \max_{\hat{\alpha}} p(\alpha=\hat{\alpha}|v=\hat{v}) \\
&\leq \sum_{\hat{v}} p(v=\hat{v}) \cdot \left[\max_{\hat{\alpha}} p(\alpha=\hat{\alpha}|v=\hat{v}, \alpha \notin \ell) + p(\alpha \in \ell) \right] \\
&\leq \sum_{\hat{v}} p(v=\hat{v}) \cdot \left[\max_{\hat{\alpha},\hat{\ell}} p(\alpha=\hat{\alpha}|v=\hat{v}, \ell=\hat{\ell}, \alpha \notin \hat{\ell}) + p(\alpha \in \ell) \right] \\
&\leq \sum_{\hat{v}} p(v=\hat{v}) \cdot \left[\frac{1}{q-2n} + \frac{2n}{q} \right] \\
&= \frac{1}{q-2n} + \frac{2n}{q}
\end{aligned}
$$

□

The security of the computation phase is now straightforward: as a worst-case scenario, we will assume that the adversary controls all but one of the players. First notice that if the adversary decides to behave (semi)-honestly, then by the correctness of `BlockCheck` the protocol will reach the end of the circuit and `CommitCheck` will not be executed.

On the other hand, if the adversary misbehaves in (at least) one of the invocations of the multiplication subprotocol in one of the blocks, either by sending an incorrect share of $\langle\varepsilon\rangle$ or $\langle\delta\rangle$ to P_k, or by having dishonest P_k announce inconsistent values (or both), then this will be detected by `BlockCheck` that will announce "Fail with Conflict" or "Fail with Agreement", depending on the adversary's precise behavior.

In the case of a "Fail with Conflict", the incorrect data is dismissed and the block is rebooted with a fresh king player that is not in the list $\mathcal{L}_{\texttt{suspects}}$ of suspect players. As we discussed in Remark 5.3.2, every re-boot adds a new player to $\mathcal{L}_{\texttt{suspects}}$, namely the previous king player, so that we can have at most n such reboots in total before the protocol produces the correct output or before $\mathcal{L}_{\texttt{suspects}}$ is "full", and in that case the protocol stops and every honest player has correctly identified at least one dishonest player (because an honest player ends up in $\mathcal{L}_{\texttt{suspects}}$ only by accusing a dishonest player). Notice that there is no need to check the commitments in this case. On the other hand, if `BlockCheck` ends with a "Fail with Agreement", then it is guaranteed that a dishonest player has broadcast an incorrect version of a value he is committed to.

As for the overall error probability, by combining the soundness error of `BlockCheck` with the bound on p_{guess}, and observing that `BlockCheck` is in-

voked at most $2n$ times – as we have n blocks plus at most n reboots – we obtain an overall error probability of at most

$$\varepsilon = 2n \cdot \left(\frac{1}{q - 2n} + \frac{2|C|/n + 2n + 1}{q} \right)$$

To sum up, and using the corresponding security properties of the input-sharing and output-reconstruction phases (cf. Propositions 5.3.7 and 5.3.8), our new MPC protocol satisfies the following.

Theorem 5.3.10 (**Security of the MPC protocol**). *For any computationally bounded adversary that cannot break the encryptions / commitments used in the preprocessing phase, except with negligible probability, an execution of our protocol results in one of the following cases (depending on the adversary's strategy):*

I. Success: *the protocol reaches the end of the circuit and outputs the correct result to all players. In this case, no commitment checks are needed.*

II. Identification without agreement: *the protocol aborts, but each honest player has identified at least one dishonest player. Also in this case, no commitment checks are needed.*

III. Identification with agreement: *the protocol aborts, and at least a dishonest player has broadcast an incorrect version of a value he is committed to.*

Hence the honest players will be able to in-agreement identify at least one dishonest player.

Furthermore, in all cases, the adversary learns no information on the honest players' inputs, beyond the result of evaluating the circuit C on the inputs.

The Complexity of our Protocol. We discuss in this section the complexity of our protocol; we focus on the circuit-evaluation phase, which is the most expensive part of our protocol. The input sharing and the output reconstruction, moreover, can be analyzed in a similar fashion (i.e., in the general case they yield a complexity of the same order of magnitude as the original SPDZ, and exceed it only to unanimously identify a dishonest player).

We start by discussing the complexity of `BlockCheck`: first notice that since each block contains at most $|C|/n$ gates, there are at most $2|C|/n$ multiplication opening values to be checked in each block; we thus get the following complexity:

- $4n$ commitments need to be prepared via `Enc`, broadcast and opened (n to produce a random seed via `Rand`, and $3n$ during `ZeroTest`);
- the computational complexity of a block check is in $O\left(|C| + n^2\right)$ field operations (excluding computation on commitments), essentially given by the cost of computing the linear combination of the values to be checked;
- finally, the block check requires broadcasting $3n$ field elements for the dispute phase of `PublicOpening`. Notice that we do not use point-to-point communication.

We thus get the following complexity of processing and checking a single block:

- First, the gates of the block are evaluated as in standard SPDZ; this yields a complexity of $|C|/n \cdot O(n) = O(|C|)$ field operations (in total over all players) and the same number of field elements for point-to-point communication, and no broadcasts.
- At the end of the block, the subprotocol `BlockCheck` is executed; as we have seen, its computation complexity is of $O(|C| + n^2)$ field operations, while its communication complexity consists of no point-to-point communication, and $3n$ broadcasts of field elements and $4n$ of commitments and openings.

Now as we have seen, `BlockCheck` can lead to a "Fail-with-Agreement" abort, to a re-boot of the current block ("Fail with Conflict"), or simply to the processing of the following block ("Success"). As argued in Remark 5.3.2, we can have at most n reboots in total before the protocol aborts; as such, the overhead of the reboots causes at most a factor 2 overhead to the ordinary computation of the n blocks.

We thus get the following result summarizing the complexity of our protocol:

Proposition 5.3.11. *Our MPC protocol has the following complexity:*

- *Computation: $O\big(n|C| + n^3\big)$ field operations, plus preparing $8n^2$ commitments (as part of* `Rand` *and of* `ZeroTest`*);*
- *Communication: $O\big(n|C|\big)$ field elements for point-to-point communication plus $O\big(n^2\big)$ broadcasts.*[8]

[8] Note that we treat broadcast as a given primitive here; implementing it using the point-to-point communication and, say, digital signatures, causes some (communication and computation) overhead, but this overhead is independent of the circuit size.

Compared to the original SPDZ protocol, in case all players behave honestly, our protocol is as efficient as the original protocol, up to an additive overhead caused by an increased number of commitments and broadcasts[9], but this overhead is independent of the circuit size and thus negligible except for small circuits. In case of active cheating – unless a dishonest player cheats so bluntly that commitment checks are invoked and he will be publicly identified as being a cheater – the (computation and communication) complexity of our protocol is larger by a factor 2 only, plus the same kind of additive overhead that does not depend on the circuit size.

5.3.6 The Commitment Check

We briefly discuss here the "last resort" to detect a dishonest player: checking the commitments on his shares.

As we have seen in Section 5.2.3, for every shared value z that is $[\cdot]$-shared in the pre-processing phase each player P_i holds randomness ρ_{z_i} and the value $e_{z_i} := \texttt{Enc}(z_i, \rho_{z_i})$ has been broadcast. Now `Enc` is well-defined, namely $e_{z_i} \neq e_{\tilde{z}_i}$ for any choice of randomness: thus a player that has broadcast an incorrect share $\tilde{z}_i$ will be unable to produce randomness that matches the cyphertext e_{z_i}.

We first give a definition of the encryption check assuming that `Enc` is linear; we discuss the more general case, where the encryption scheme is only assumed to be somewhat-homomorphic, in the next paragraph. We make the remark that similar techniques and ideas have been used in a paper by Baum et al. [3], where commitments are employed to allow an external party to check the correctness of an execution of SPDZ. In [3], Pedersen commitments [55] are used for this auditing process (instead of the encryption scheme from SPDZ).

`CommitCheck` (linear case):

the protocol takes as input the index i of a player P_i and his share $z_i = \sum_{h=1}^{M} \lambda^{(h)} z_i^{(h)}$, where all $\left[z^{(h)}\right]$ are computed in the preprocessing phase; let $e_i^{(h)} := \texttt{Enc}\left(z_i^{(h)}, \rho_i^{(h)}\right)$ (these values are public, cf. Section 5.2.3). We assume that the coefficients $\lambda^{(h)}$ are public as well.

(i) Players set $e_i := \sum_{h=1}^{M} \lambda^{(h)} e_i^{(h)}$;

(ii) P_i computes and broadcasts $\rho_i := \sum_{h=1}^{M} \lambda^{(h)} \rho_i^{(h)}$;

[9] Plus that we have to do *real* broadcasts, whereas in the original SPDZ protocol without cheater detection it is good enough to do a simple consistency check and abort as soon as there is an inconsistency.

(iii) players set $e_i := \texttt{Enc}(z_i, \rho_i)$.

If $e_i = \texttt{Enc}(z_i, \rho_i)$, the protocol outputs $\top$; otherwise, it outputs $\perp$ and P_i is deemed dishonest.

Trivially, if P_i behaves honestly, `CommitCheck` will output $\top$; on the other hand, if the share $\tilde{z}_i$ he submitted is not correct, then the output will be $\perp$ since $e_{z_i} \neq \texttt{Enc}(\tilde{z}_i, \tilde{\rho}_i)$ for any randomness $\tilde{\rho}_i$.

Now thanks to Theorem 5.3.10, if the MPC protocol ends in a "Fail-with-Agreement" case, then it is guaranteed that a dishonest player P_i has broadcast a $\tilde{z}_i \neq z_i$; the values $\tilde{z}_1, \ldots, \tilde{z}_n$ to be checked are clear from the definition of the subprotocols we discussed. Thus a dishonest player will be publicly identified as such in a "Fail-with-Agreement" case.

The General Case. The security of `CommitCheck` is based on the assumption that `Enc` is a linear map. Now in the original SPDZ, a different assumption is made, namely that the encryption scheme is somewhat-homomorphic. This means that there exists a *decryption* protocol `Dec` such that

$$\texttt{Dec}\left(\sum_{h=1}^{M} \lambda^{(h)} \texttt{Enc}\left(z^{(h)}, \rho^{(h)}\right)\right) = \sum_{h=1}^{M} \lambda^{(h)} z^{(h)}$$

for all $M \leq M'$, where M' is a positive constant (notice that here we have dropped the indexes i to simplify the notation). We do not discuss here the security properties of this decryption protocol, as this would be beyond the scope of this dissertation; the detailed explanation can be found in [23].

Now if the share of player P_i to be checked is $z_i = \sum_h \lambda^{(h)} z_i^{(h)}$ for $M \leq M'$, then players only need to slightly modify `CommitCheck`: namely, they need to compute $\texttt{Dec}\left(\sum_h \lambda^{(h)} e_i^{(h)}\right)$ and then check that it is equal to z_i.

If, on the other hand, we have that $M > M'$, then the encryption scheme can no longer be used as a commitment scheme to prove the validity of z_i. In this case, we require players to switch to another *linear* commitment scheme `Commit`,[10] and then demonstrate equality of the committed values via a zero-knowledge proof (i.e., without leaking any other information besides the fact

[10]Here and in the following protocol, with a slight abuse of notation we denote by `Commit` both the commitment scheme and the algorithm that produces the commitment.

that the committed values coincide). More precisely, the protocol becomes the following:

`CommitCheck`:

the protocol takes as input the index i of a player P_i and his share $z_i = \sum_{h=1}^{M} \lambda^{(h)} z_i^{(h)}$, where all $\left[z^{(h)}\right]$ are computed in the preprocessing phase; let $e_i^{(h)} := \texttt{Enc}\left(z_i^{(h)}, \rho_i^{(h)}\right)$ (these values are public, cf. Section 5.2.3). We assume that the coefficients $\lambda^{(h)}$ are public as well, and that players have access to a linear commitment scheme `Commit`.

(i) P_i computes and broadcast a commitment $c^{(h)} := \texttt{Commit}(z_i^{(h)})$ for all h, and publicly proves in zero-knowledge that $c^{(h)}$ and $e^{(h)}$ open to the same value for any h;

(ii) players compute $c := \sum_{h=1}^{M} \lambda^{(h)} c^{(h)}$;

(iii) P_i opens the commitment c.

If c opens to z_i, the protocol outputs $\top$; otherwise, it outputs $\bot$ and P_i is deemed dishonest.

Bibliography

[1] Saurabh Agarwal, Ronald Cramer, and Robbert de Haan. Asymptotically optimal two-round perfectly secure message transmission. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 394–408, 2006.

[2] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, Jul 2000.

[3] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 175–196, 2014.

[4] Carsten Baum, Emmanuela Orsini, and Peter Scholl. Efficient secure multiparty computation with identifiable abort. *IACR Cryptology ePrint Archive*, 2016:187, 2016.

[5] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 420–432, 1991.

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, 1988. ACM.

[7] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT 2011*, pages 169–188. Springer, Heidelberg, 2011.

[8] C. Berge. *Graphs and Hypergraphs*. North-Holland mathematical library. Amsterdam, 1973.

[9] George R. Blakley. Safeguarding Cryptographic Keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, volume 48, pages 313–317, June 1979.

[10] Ning Cai and R.W. Yeung. Secure network coding. In *Information Theory, 2002. Proceedings. 2002 IEEE International Symposium on*, pages 323–, 2002.

[11] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.

[12] Alfonso Cevallos, Serge Fehr, Rafail Ostrovsky, and Yuval Rabani. Unconditionally-secure robust secret sharing with compact shares. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 7237 of *Lecture Notes in Computer Science*, pages 195–208. Springer, 2012.

[13] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 11–19, New York, NY, USA, 1988. ACM.

[14] Hao Chen, Ronald Cramer, Shafi Goldwasser, Robbert De Haan, and Vinod Vaikuntanathan. Secure computation from random error correcting codes. In *Advances in Cryptology-EUROCRYPT 2007*, pages 291–310. Springer, 2007.

[15] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 383–395, 1985.

[16] Ronald Cramer, Ivan Damgård, and Serge Fehr. On the cost of reconstructing a secret, or VSS with optimal reconstruction phase. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 503–523, 2001.

[17] Ronald Cramer, Ivan Damgård, and Serge Fehr. On the cost of reconstructing a secret, or vss with optimal reconstruction phase. In *CRYPTO*, pages 503–523, 2001.

[18] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on*

the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding, pages 316–334, 2000.

[19] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[20] Ronald Cramer, Ivan Bjerre Damgård, Nico Döttling, Serge Fehr, and Gabriele Spini. Linear secret sharing schemes from error correcting codes and universal hash functions. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 313–336, 2015.

[21] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 471–488, 2008.

[22] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, pages 1–18, 2013.

[23] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[24] Ph Delsarte. Bilinear forms over a finite field, with applications to coding theory. *Journal of Combinatorial Theory, Series A*, 25(3):226 – 241, 1978.

[25] Yvo Desmedt and Yair Frankel. Shared generation of authenticators and signatures (extended abstract). In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 457–469, 1991.

[26] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[27] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *J. ACM*, 40(1):17–47, January 1993.

[28] Erez Druk and Yuval Ishai. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In *ITCS*, pages 169–182, 2014.

[29] S.Y. El Rouayheb and E. Soljanin. On wiretap networks ii. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pages 551–555, June 2007.

[30] Jon Feldman, Tal Malkin, Rocco A. Servedio, and Cliff Stein. On the capacity of secure network coding. In *In Proc. 42nd Annual Allerton Conference on Communication, Control, and Computing*. Cambridge University Press, 2004.

[31] È. M. Gabidulin. Theory of codes with maximum rank distance. *Problemy Peredachi Informatsii*, 21(1):3–16, 1985.

[32] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.

[33] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304, 1985.

[34] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 89–98, 2006.

[35] Jacopo Griggio. *Perfectly Secure Message Transmission Protocols with Low Communication Overhead and Their Generalization.* 2012.

[36] Venkatesan Guruswami and Piotr Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005.

[37] Venkatesan Guruswami and Atri Rudra. Explicit capacity-achieving list-decodable codes. In *STOC*, pages 1–10, 2006.

[38] Venkatesan Guruswami and Chaoping Xing. Optimal rate list decoding of folded algebraic-geometric codes over constant-sized alphabets. In *SODA*, pages 1858–1866, 2014.

[39] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *STOC*, pages 433–442, 2008.

[40] Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 21–38, 2012.

[41] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO 2014*, pages 369–386. Springer, Heidelberg, 2014.

[42] Sidharth Jaggi, Michael Langberg, Sachin Katti, Tracey Ho, Dina Katabi, Muriel Médard, and Michelle Effros. Resilient network coding in the presence of byzantine adversaries. *IEEE Trans. Information Theory*, 54(6):2596–2603, 2008.

[43] R. Koetter and M. Medard. An algebraic approach to network coding. *Networking, IEEE/ACM Transactions on*, 11(5):782–795, Oct 2003.

[44] Hugo Krawczyk. Secret sharing made short. In *CRYPTO*, pages 136–146, 1993.

[45] K. Kurosawa and K. Suzuki. Truly efficient 2 -round perfectly secure message transmission scheme. *Information Theory, IEEE Transactions on*, 55(11):5223–5232, Nov 2009.

[46] Kaoru Kurosawa and Kazuhiro Suzuki. Truly efficient 2-round perfectly secure message transmission scheme. In *EUROCRYPT*, pages 324–340, 2008.

[47] S.-Y.R. Li, R.W. Yeung, and Ning Cai. Linear network coding. *Information Theory, IEEE Transactions on*, 49(2):371–381, Feb 2003.

[48] Desmond S. Lun, Muriel Médard, Ralf Koetter, and Michelle Effros. On coding for reliable communication over packet networks. *Physical Communication*, 1(1):3–20, 2008.

[49] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error Correcting Codes*. North-Holland mathematical library. North-Holland Publishing Company, 1977.

[50] Yishay Mansour, Noam Nisan, and Prasoon Tiwari. The computational complexity of universal hashing. In *Proceedings: Fifth Annual Structure in Complexity Theory Conference, Universitat Politècnica de Catalunya, Barcelona, Spain, July 8-11, 1990*, page 90, 1990.

[51] J. L. Massey. Some applications of coding theory in cryptography. In *Codes and Ciphers: Cryptography and Coding IV*, pages 33–47, 1995.

[52] Ueli Maurer. Secret key agreement by public discussion. *IEEE Transactions on Information Theory*, 39(3):733–742, May 1993.

[53] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, New York, NY, USA, 2005.

[54] Moni Naor and Avishai Wool. Access control and signatures via quorum secret sharing. *IEEE Trans. Parallel Distrib. Syst.*, 9(9):909–922, 1998.

[55] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 129–140, 1991.

[56] Michael O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 403–409, 1983.

[57] Hasan Md. Sayeed and Hosame Abu-Amara. Efficient perfectly secure message transmission in synchronous networks. *Information and Computation*, 126(1):53 – 61, 1996.

[58] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[59] Bhavani Shankar, Kannan Srinathan, and C. Pandu Rangan. Alternative protocols for generalized oblivious transfer. In *Distributed Computing and Networking, 9th International Conference, ICDCN 2008, Kolkata, India, January 5-8, 2008.*, pages 304–309, 2008.

[60] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.

[61] C. E. Shannon. Communication theory of secrecy systems*. *Bell System Technical Journal*, 28(4):656–715, 1949.

[62] D. Silva and F.R. Kschischang. Security for wiretap networks via rank-metric codes. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 176–180, July 2008.

[63] D. Silva and F.R. Kschischang. Universal secure network coding via rank-metric codes. *Information Theory, IEEE Transactions on*, 57(2):1124–1135, Feb 2011.

[64] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 388–397, 1995.

[65] Gabriele Spini and Serge Fehr. Cheater detection in SPDZ multiparty computation. In *Information Theoretic Security - 9th International Conference, ICITS 2016, Tacoma, WA, USA, August 9-12, 2016, Revised Selected Papers*, pages 151–176, 2016.

[66] Gabriele Spini and Gilles Zémor. Perfectly secure message transmission in two rounds. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, pages 286–304, 2016.

[67] Gabriele Spini and Gilles Zémor. Universally secure network coding with feedback. In *IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016*, pages 2339–2343, 2016.

[68] K. Srinathan, Arvind Narayanan, and C.Pandu Rangan. Optimal perfectly secure message transmission. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 545–561. Springer Berlin Heidelberg, 2004.

[69] Douglas Stinson. *Cryptography: Theory and Practice,Second Edition.* CRC/C&H, 2nd edition, 2002.

[70] Jay Kumar Sundararajan, Devavrat Shah, Muriel Médard, Szymon Jakubczak, Michael Mitzenmacher, and João Barros. Network coding meets TCP: theory and implementation. *Proceedings of the IEEE*, 99(3):490–512, 2011.

[71] Tamir Tassa. Generalized oblivious transfer by secret sharing. *Des. Codes Cryptography*, 58(1):11–21, 2011.

[72] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*, pages 53–70, 2011.

[73] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

Summary

This dissertation presents contributions to four areas in Cryptography, investigating in particular its connection to Coding Theory. Concretely, we make use of techniques from Coding Theory to construct and analyze cryptographic protocols with new and/or enhanced properties.

We first focus on *Secret Sharing,* an important topic which forms the common ground for most of the concepts discussed in this thesis. A secret-sharing scheme takes as input a secret value, and produces as output n shares in such a way that small enough sets of shares yield no information at all on the secret (*privacy*), while large enough sets of shares allow to recover the secret (*reconstruction*). Secret Sharing has many applications to Cryptography. For instance, secret-sharing schemes with additional properties, such as linearity and multiplicativity, form a fundamental building block for secure *Secure Multi-Party Computation* (*MPC*). In MPC, n parties who do not necessarily trust each other can to compute the value $f(x_1, \ldots, x_n)$ of a function f on mutually private inputs $x_1, \ldots, x_n$, while guaranteeing the correctness of the output and while keeping their respective inputs private from each other.

Furthermore, Secret Sharing has several direct applications. It was originally motivated by reliable and secure distributed storage of sensitive information, relaxing the single-point-of-failure problem of conventional storage methods. Moreover, a secret-sharing scheme can be used to solve the following special instance of *Perfectly Secure Message Transmission.* Assume that a sender Alice is connected to a receiver Bob via n distinct channels, some of which are controlled by an adversary Eve. By using Secret Sharing, it is easy to devise a scheme that allows Alice to communicate a secret message to Bob in such a way that Eve learns no information on the message by eavesdropping on the channels she controls, while Bob can receive the message even if Eve blocks the channels under her control.

Recovering data from incomplete information is common to both Secret Sharing and Coding Theory. Therefore, it is perhaps not surprising that the two

fields have known a long and fruitful interplay. In particular, Massey suggested the construction of linear secret-sharing schemes from linear error-correcting codes, and showed how to analyze properties of a secret-sharing scheme in terms of those of the underlying code. From this analysis, one can derive schemes whose reconstruction is governed by a parameter called *distance* of the underlying code, and whose privacy is governed in terms of *dual distance,* which depends on the *dual* of the code. There are examples of good codes that have good duals as well (i.e., codes that have large distance and large dual distance), e.g. Reed-Solomon or random linear codes. However, it is currently not known how to achieve these properties while at the same time providing very efficient (i.e., linear-time) encoding and decoding.

We circumvent this problem by introducing an alternative paradigm where the privacy of secret-sharing schemes no longer depends on the dual distance of the underlying code, but is controlled by the rate of the underlying code and by the parameters of a family of *linear universal hash functions.* This allows us to fully harness the potential of recent code constructions to obtain improved schemes; we exemplify this by means of two applications. First, we make use of linear-time encodable and decodable codes to obtain a family of secret-sharing schemes with asymptotically good reconstruction and privacy where both the computation of the shares and the reconstruction of the secret can be performed in linear time. Second, we make instead use of list-decodable codes to obtain robust secret-sharing schemes, i.e., schemes that can recover the secret even if some of the shares are incorrect, except with a small error probability. The family we present optimizes the trade-off between the extra data that needs to be appended to the share to achieve robustness and the error probability in the reconstruction, reaching the best possible value.

The next topic we study is *Perfectly Secure Message Transmission* or *PSMT* for short. As opposed to the non-interactive variant described above, we now imagine that Bob is also allowed to send messages over the n channels to Alice, where Eve corrupts and controls $t < n$ of the channels, meaning that she is able to read the data transmitted over the corrupted channels and replace them with symbols of her choice. The interesting point is that two-way communication improves security, namely it is possible to achieve perfect privacy and perfect correctness as long as $t < n/2$, while in the non-interactive case, this can only be achieved if $t < n/3$.

We present a new protocol working in two rounds and for any number $t < n/2$ of corrupted channels. The use of techniques based on syndrome computation allows for a conceptually simpler blueprint and for improved efficiency, dividing by a factor n the amount of data to be communicated to transmit a single-bit secret.

PSMT can be generalized to scenarios where Alice and Bob are connected by

more complex communication means. For instance, in *Secure Network Coding* Alice and Bob are connected by a network, i.e. a directed and acyclic graphs with a single source node (with no incoming wires) and a single sink node (with no outbound wires); Alice can feed input to the source node, Bob can read the data received by the sink node, while intermediate nodes compute linear combinations of the data received via the incoming wires and transmit it over the outbound wires. It is still assumed that an adversary Eve controls (i.e., taps and tampers with) some of the wires in the network.

So far, Secure Network Coding has only addressed one-way communication. In this dissertation, we consider, for the first time, two-way communication, and show that it enables to achieve security for any $t < C/2$ (rather than $t < C/3$ in the non-interactive case), where C is an invariant of the network topology known as *connectivity*. We present two protocols, adapted from our construction for PSMT; the first one works in two rounds, while the second one works in three, but can be adapted to a scenario where multiple receivers (instead of just one) are present. Both our protocols can tolerate any number $t < C/2$ of corrupted wires, and are secure regardless of the way internal nodes of the network transform the received data.

Finally, we focus on *Secure Multi-Party Computation* or *MPC*. As briefly discussed above, in MPC n parties known as *players* hold private inputs $x_1, \ldots, x_n$ respectively and aim at computing the value $f(x_1, \ldots, x_n)$ of a function f on their inputs, while guaranteeing the correctness of the output and while keeping their respective inputs private.

A recent construction is the so-called SPDZ protocol by Damgård et al., distinguished by its fast performance (and thereby its promising possibilities of becoming practical). A downside of the SPDZ protocol is that it is susceptible to a denial-of-service attack: even a single dishonest player can enforce the computation to fail, meaning that the honest parties have to *abort* the computation without learning the outcome, whereas the cheating party may actually learn it. Furthermore, the dishonest party can launch such an attack without being identified to be the cheater.

We enhance the SPDZ protocol to allow for cheater detection: a dishonest party that enforces the protocol to fail will be identified as being a cheater. As a consequence, in typical application scenarios, parties will actually have little incentive to cheat, and if cheating still takes place, the cheater can be identified and discarded and the computation can be re-done, until it succeeds.

The challenge lies in adding this cheater detection feature to the original protocol without increasing its complexity significantly. To achieve this, we introduce a carefully-designed dispute control system that allows players to proceed with the execution of the protocol in cases where the original SPDZ protocol

has to abort, so that eventually our protocol produces the correct output or malicious players are identified. This allows us to obtain the following. In case no cheating takes place, our protocol is as efficient as the original SPDZ protocol which has no cheater detection. In case cheating does take place, there may be some additional overhead, which is still reasonable in size though, and since the cheater knows he will be caught, this is actually unlikely to occur in typical application scenarios.

Samenvatting

Dit proefschrift maakt bijdragen aan vier deelgebieden in de cryptografie, met nadruk op de verbanden met de coderingstheorie (ofwel, de theorie van de foutcorrigerende codes). We maken gebruiken van coderingstheoretische technieken om cryptografische protocollen zowel te analyseren als te construeren met nieuwe en verbeterde eigenschappen.

Secret sharing is een belangrijk onderwerp in de moderne cryptografie, en het vormt de grondslag voor veel van de besproken begrippen on deze dissertatie. Een secret-sharingschema heeft als input een geheime waarde en geeft als output n *shares*. Deze shares hebben de eigenschap dat een kleine deelverzameling ervan geen enkele informatie geeft over de geheime input (*privacy*), maar dat met een voldoende grote deelverzameling het geheim kan worden achterhaald (*reconstructie*). Secret sharing heeft talrijke toepassingen in de cryptografie. Bijvoorbeeld, secret sharing met toegevoegde eigenschappen zoals lineariteit en multiplicativiteit vormt een fundamentele bouwsteen voor secure multiparty computation ofwel MPC. Met behulp van MPC kunnen n partijen de functie-waarde $f(x_1, \ldots, x_n)$ van een functie f op geheime inputs $x_1, \ldots, x_n$ correct uitrekenen zonder daarbij de inputs te hoeven uitwisselen; die blijven namelijk geheim.

Voorts heeft secret sharing een aantal directe toepassingen. De meest oorspronkelijke daarvan is betrouwbare en veilige gedistribueerde opslag van sensitieve informatie; hierbij geeft secret sharing een oplossing voor het *single-point-of-failure* probleem van traditionele opslag-methoden. Een *secret sharing* schema kan ook worden gezien als oplossing voor de volgende niet-interactieve variant van *perfectly secure message transmission.* Stel dat Alice veilig een bericht wil sturen naar Bob en daarvoor beschikking heeft over n communicatiekanalen, en zodat een aanvaller Eve enkele van deze kanalen kan afluisteren dan wel blokkeren. Op basis van *secret sharing* is het eenvoudig om een methode te geven die Alice in staat stelt om een bericht naar Bob te sturen zonder dat Eve informatie verkrijgt over de inhoud van het bericht. Tevens kan Eve het bericht naar Bob niet blokkeren, ook niet als zij alle communi-

catiekanalen blokkeert waarover ze controle heeft.

Het herstellen van data uit onvolledige informatie vormt een overeenkomst tussen secret sharing en coderingstheorie. Het is daarom wellicht niet verwonderlijk dat er tussen deze twee gebieden sprake is van een lang en vruchtbaar samenspel. Bijvoorbeeld, Massey heeft een constructie van *secret sharing* uit lineaire foutcorrigerende codes voorgesteld en heeft laten zien hoe de eigenschappen van het *secret-sharing* schema begrepen kunnen worden in termen van de eigenschappen van de onderliggende code. Hieruit volgt bijvoorbeeld het bestaan van *secret-sharing* schema's waarin de reconstructie bepaald wordt door de minimum afstand van de code en waarin de privacy bepaald wordt door de minimum afstand van de duale code. Er zijn voorbeelden van goede codes waarvan de duale code tevens goed is, zoals Reed-Solomon codes of random gekozen lineaire codes. Echter, het is thans een open probleem of er goede codes bestaan die een goede duale code hebben en die tevens zeer efficiënte (dat wil zeggen, linear-time) encodeer- en decodeer-algoritmen toelaten.

Dit probleem wordt omzeild in dit proefschrift door een nieuw verband te leggen tussen *secret-sharing* en codes, die als resultaat heeft dat de privacy van het cryptografische protocol niet meer afhangt van de duale van de onderliggende code, maar in plaats daarvan bepaald wordt door de *rate* van de onderliggende code en de parameters van een familie van *lineaire universele hash functies.* Dit geeft mogelijkheden om het potentieel van de meest recente codes volledig te exploiteren en daarmee verbeterde protocollen te genereren. We lichten dit toe met twee toepassingen. De ene toepassing maakt gebruik van codes waarvan de codering en decodering kan worden berekend in lineaire tijd. Hierdoor is het mogelijk een familie van *secret-sharing* schema's te construeren waarbij zowel het delen van de *shares* als het reconstrueren van geheime data geschiedt in slechts lineaire tijd.

De andere toepassing construeert *robuuste secret-sharing* schema's met behulp van codes met zogeheten lijstdecodering. Zulke protocollen zijn zelfs – hoewel met een kleine foutkans – in staat het gedeelde geheim te reconstrueren wanneer enkele van de *shares* incorrect zijn. De familie van protocollen uit deze toepassing optimaliseert de trade-off tussen de hoeveelheid extra data die toegevoegd moet worden aan de *shares* om robuustheid te behouden en de foutkans van de reconstructie van het gedeelde geheim.

Het volgende onderwerp van studie is *perfectly secure message transmission (PSMT).* In tegenstelling tot de niet-interactieve variant die eerder aangehaald is, wordt nu aangenomen dat Bob ook in staat is om berichten terug naar Alice te sturen over de n bestaande communicatiekanalen. De aanvaller Eve heeft nog steeds $t < n$ van deze kanalen onder controle, en kan daarmee berichten afluisteren en veranderen naar willekeur. Opmerkelijk is dat wederzijdse communicatie de veiligheid verhoogt, aangezien het in dit scenario mogelijk is om

perfecte privacy en reconstrueerbaarheid te bemachtigen zolang $t < n/2$. In het niet-interactieve geval kan dit alleen als $t < n/3$.

In dit proefschrift wordt een nieuw protocol gepresenteerd voor $t < n/2$ dat twee rondes heeft. Het gebruik van technieken die gebaseerd zijn op zogenaamde *syndrome computation* maakt een conceptueel simpelere aanpak mogelijk die tevens leidt tot verbeterde efficiëntie, namelijk een multiplicatieve factor n besparing in totale communicatie in het speciale geval van een constante bit-lengte secret.

PSMT kan gegeneraliseerd worden naar scenario's waarin Alice en Bob toegang hebben tot een complexere communicatie infrastructuur. Bijvoorbeeld, in *secure network coding* zijn ze verbonden door een netwerk dat gemodelleerd wordt door een gerichte, acyclische graaf met een enkele *source* knoop zonder inkomende takken en een enkele *sink* knoop zonder uitgaande takken. Alice kan data verzenden via de *source* en Bob ontvangt data via de *sink,* waarbij de tussenliggende knopen lineaire transformaties toepassen op de inkomende takken en resultaten uitsturen over de uitgaande takken. Nog steeds wordt aangenomen dat Eve een zeker aantal van de kabels van het netwerk onder haar controle heeft.

In eerdere studies wordt normaliter aangenomen dat communicatie in *secure network coding* maar één richting opgaat. In dit proefschrift stellen wij daarentegen ook interactieve protocollen voor die – analoog aan PSMT – door de communicatie in beide richtingen veilig blijven bij een groter aantal gecontroleerde netwerkkabels. De enige conditie is namelijk dat $t < C/2$ (vergelijken met $t < C/3$ in het niet-interactieve geval), waar C een invariant is van de netwerk topologie, de zogeheten *connectivity.* We beschrijven namelijk twee protocollen, beide aanpassingen van het schema voor PSMT. De eerste heeft twee fasen, terwijl de tweede er drie heeft en aangepast kan worden aan een situatie met meerdere ontvangers. Deze twee protocollen zijn bestand tegen het theoretisch maximale aantal aangedane netwerkkabels en blijven veilig, ongeacht de bewerkingen van de netwerkknooppunten.

Tenslotte verleggen we de focus naar *secure multi-party computation*, wat wordt afgekort naar MPC. Hierbij gaat het over n partijen – ook wel *spelers* genoemd – die elk een geheime inputwaarde x_i hebben. Het doel is om de uitkomst van een zekere functie $f(x_1, \ldots, x_n)$ correct te berekenen met deze geheime waarden als input, zodat de spelers hun inputs geheim houden.

Het recente werk van Damård et al. beschrijft een protocol dat zelfs met $n-1$ kwaadaardige spelers veilig blijft. Dit zogeheten SPDZ-protocol is geroemd vanwege zijn efficiëntie en de mogelijkheid in de praktijk te kunnen worden gebruikt. Dit SPDZ-protocol heeft echter als nadeel dat al één enkele kwaadaardige partij de hele gezamenlijke berekening kan doen mislukken. Dit

houdt onder andere in dat de eerlijke partijen genoodzaakt zijn de berekening af te breken en de uitkomst niet hebben, terwijl de kwaadaardige partij wellicht wel deze uitkomst kan leren van de gedeelde informatie. Sterker nog, deze oneerlijke partij kan dit doen zonder dat de andere partijen erachter kunnen komen wie de veroorzaker was.

In deze dissertatie wordt het SPDZ-protocol verbeterd zodat er de mogelijkheid bestaat om de kwaadaardige partij, die de berekening doet afbreken, te kunnen identificeren. Bijgevolg zullen partijen niet de drang hebben vals te spelen als zij dit verbeterde protocol gebruiken. Een kwaadwillige kan namelijk worden geïdentificeerd en vervolgens uit de groep gezet worden.

De hoofduitdaging was om identificatie van kwaadwilligen aan het SPDZ-protocol toe te voegen, zonder dat de algehele complexiteit significant verhoogd wordt. Om dit te bewerkstelligen, voeren we een nauwkeurig ontworpen *dispute control* systeem in dat de partijen in staat stelt om de executie van het protocol doorgang te laten vinden in gevallen waarin het oorspronkelijke SPDZ protocol een "abort" zou dienen te declareren, met de eigenschap dat de correcte output wordt gegenereerd of dat in ieder geval malicieuze spelers geïdentificeerd worden. Voorts, de protocol die wordt beschreven in deze dissertatie is – mits het protocol correct wordt uitgevoerd – even efficiënt als het SPDZ-protocol. Wanneer er wél sprake is van valsspelerij, kan het protocol een relatief redelijk kleine vertraging oplopen. Omdat kwaadwilligen kunnen worden opgespoord met dit protocol, is het echter niet waarschijnlijk dat deze vertraging werkelijk voorkomt in de praktijk.

Résumé

Le sujet de cette thèse est la cryptographie et son interconnexions avec la théorie des codes. En particulier, on utilise des techniques issues de la théorie des codes pour construire et analyser des protocoles cryptographiques avec des propriétés nouvelles ou plus avancées.

On se concentre d'abord sur le *partage de secret* ou *secret sharing,* un sujet important avec de nombreuses applications pour la cryptographie actuelle. Dans la variante à laquelle on s'intéresse, un schéma de partage de secret reçoit en entrée un élément secret, et renvoie en sortie n parts de telle façon que chaque ensemble de parts de taille suffisamment petite ne donne aucune information sur le secret *(confidentialité),* tandis que chaque ensemble de taille suffisamment grande permet de reconstituer le secret *(reconstruction).* Un schéma de partage de secret peut donc être vu comme une solution à un problème de communication où un émetteur Alice est connectée avec un destinataire Bob par n canaux distincts, dont certains sont contrôlés par un adversaire Ève. Alice peut utiliser un schéma de partage de secret pour communiquer un message secret a Bob de telle façon qu'Ève n'apprenne aucune information sur le secret en lisant les données transmises sur les canaux qu'elle contrôle, tandis que Bob peut recevoir le message même si Ève bloque ces dits canaux.

Notre contributions au partage de secret concernent ses liens avec la théorie des codes ; comme les deux domaines partagent un même but (récupérer des données à partir d'informations partielles), ce n'est pas surprenant qu'ils aient connu une interaction longue et fertile. Plus précisément, Massey commença une analyse fructueuse à propos de la construction et de l'étude d'un schéma de partage de secret à partir d'un code correcteur. L'inconvénient de cette analyse est que la confidentialité d'un schéma de partage de secret est estimé grâce au dual du code sous-jacent ; cela peut être problématique vu qu'il pourrait ne pas être possible d'obtenir des codes avec des propriétés souhaitables qui aient aussi un bon code dual.

On contourne ce problème en établissant une connexion nouvelle entre les

deux domaines, telle que la confidentialité d'un schéma de partage de secrets n'est plus contrôlée par le dual du code sous-jacent. Cela nous permet d'exploiter complètement le potentiel de certaines constructions récentes de codes pour obtenir des meilleurs schémas ; on illustre ceci avec deux applications. Premièrement, en utilisant des codes avec codage et décodage en temps linéaire on obtient une famille de schémas de partage de secret où le partage (calcul des parts issues du secret) tout comme la reconstruction peuvent s'effectuer en temps linéaire ; pour des seuils de confidentialité et de reconstruction croissants, ceci restait jusqu'à présent un problème ouvert. Deuxièmement, on utilise des codes avec décodage en liste pour construire des schémas de partage de secret robustes, c'est-à-dire des schémas qui peuvent reconstituer le secret même si certaines parts sont incorrectes, sauf avec une petite probabilité d'erreur. La famille que nous présentons optimise le compromis structurel entre les données additionnelles qui doivent être ajoutées aux parts pour obtenir la robustesse et la probabilité d'une erreur dans la reconstruction, atteignant la meilleur valeur possible.

Une variante interactive du partage de secret est donnée par la *transmission parfaitement sécurisée de message,* connue sous son acronyme anglais *PSMT* pour *perfectly secure message transmission.* On imagine maintenant que Bob aussi soit autorisé à envoyer des messages à Alice sur plusieurs canaux, et qu'Ève soit capable de lire les données envoyées sur les canaux qu'elle contrôle aussi que de les remplacer par des élément de son choix. La communication à deux sens améliore la sécurité : il est possible d'obtenir confidentialité et exactitude parfaites tant qu'Ève contrôle moins de la moitié des canaux, tandis que dans le cas non interactif, ceci peut être atteint seulement si elle contrôle moins d'un tiers des canaux.

On présente dans cette thèse un nouveau protocole qui marche en deux étapes et pour le maximum possible de canaux corrompus ; son importance pour le sujet est due à deux aspects. Premièrement, son efficacité est améliorée par rapport aux constructions précédemment connues, spécialement quand le secret à transmettre ne consiste qu'en un bit unique. Deuxièmement, il se différencie des travaux précédents en évitant les lourdes méthodes classiques et en utilisant au contraire des techniques plus simples issues de la théorie des codes.

La PSMT se peut généraliser à des scénarios de communication plus complexes. Par exemple, dans le *codage sécurisé de réseau* ou *secure network coding* Alice est connectée avec Bob par une infrastructure de communication plus compliquée, qui consiste en un réseau de câbles et de nœuds qui transforment et transmettent les données reçues ; on suppose toujours qu'Ève contrôle (c'est-à-dire, écoute et manipule) certains câbles dans le réseau.

Seulement la communication unilatérale a été étudiée auparavant dans le

codage sécurisé de réseau. On contribue au sujet en présentant des protocoles interactifs ; comme dans les scénarios précédents, cela nous permet d'obtenir des procédés sûrs pour un plus grand nombre de câbles corrompus. Plus précisément, on présente deux protocoles adaptés de notre construction de PSMT ; le premier marche en deux étapes, tandis que le seconde marche en trois, mais peut être adapté à un scénario où plusieurs destinataires (ou "Bobs") sont présents. Nos deux protocoles tolèrent le plus grand nombre possible de câbles corrompus, et restent sûrs quelle que soit la façon dont les nœuds du réseau transforment les données reçues.

Finalement, on se concentre sur le *calcul sécurisé multi-parties,* ou *MPC* de son nom en anglais. Dans le MPC, n utilisateurs détiennent des entrées privées $x_1, \ldots, x_n$ respectivement et veulent calculer la valeur $f(x_1, \ldots, x_n)$ d'une fonction f sur leurs entrées, tout en garantissant l'exactitude du calcul aussi que la confidentialité des données d'entrée.

Une construction récente est le protocole SPDZ réalisé par Damgård et al., remarquable par son exécution très rapide et ses intéressantes possibilités d'implémentation. Un inconvénient du protocole SPDZ est sa vulnérabilité à un attaque par déni de service: même un seul utilisateur malveillant peut causer un échec du calcul, ce qui veut dire que les utilisateur honnêtes doivent abandonner le calcul sans en apprendre le résultat, tandis que l'utilisateur malveillant pourrait l'apprendre. Qui plus est, un tel utilisateur malveillant peut monter ce type d'attaques sans qu'il puisse être identifie comme malhonnête.

On améliore le protocole SPDZ pour obtenir l'identification des utilisateurs malhonnêtes : un tel utilisateur qui fait intentionnellement échouer le protocole sera identifié comme malveillant. Par conséquent, dans des scénarios classiques du monde réel, les utilisateurs auront peu de motivation pour tricher, et si quelqu'un triche malgré tout, l'utilisateur malhonnête peut être identifié et expulsé du procédé et le calcul exécuté une autre fois, jusqu'à ce qu'il réussisse.

Le défi est d'ajouter cette fonctionnalité de détection des tricheurs au protocole originel sans augmenter significativement sa complexité. Si personne ne triche, notre nouveau protocole est aussi efficace que le protocole SPDZ originel qui ne possède pas de détection de tricheurs. En revanche, si quelqu'un triche, il est possible que le coût augmente – mais de manière raisonnable, et comme le tricheur sait qu'il sera attrapé, il est par ailleurs improbable que cela se passe dans des typiques scénarios réels.

Acknowledgments

First of all, I would like to thank my advisors Ronald Cramer, Serge Fehr and Gilles Zémor for their guidance throughout this PhD. Ronald constantly shared his great knowledge of cryptography and his many new ideas with me, while pushing me to look for the mathematics underlying cryptographic topics. Serge's constant presence and help has been invaluable at every stage of my doctorate; his steady flow of research ideas and feedback has been fundamental for the development of my scientific skills. And finally, thanks to Gilles for the long and fruitful discussions that gave me new perspectives on the interplay between Coding Theory and Cryptography; moreover, his relentless help in every sort of practical matters has been fundamental throughout my doctorate.

I am also grateful to the members of the Doctorate Committee for taking the time to read my thesis; a special thank to Yuval Ishai and Berry Schoenmakers for agreeing to write a report according to Bordeaux regulations.

I would also like to thank the friends and colleagues at CWI, both within and outside the Crypto group, for the discussions on math (or crypto, or coding) and for the great moments we had together; and of course, many thanks to all my friends, in the Netherlands, in Bordeaux, back in Italy and in the many countries united by the Erasmus program, for their support and for all the fun times we had.

Last but not least: grazie a tutta la mia famiglia, per essere sempre stata presente e avermi sostenuto (a volte anche con un oceano in mezzo) durante gli anni di questo dottorato.

Curriculum Vitae

Gabriele Spini was born in Sondrio, Italy, on June 19, 1989. He grew up in the nearby town of Morbegno, and obtained his high school diploma from Liceo Scientifico P. Nervi in 2008.

He then started his bachelor studies in Mathematics at the University of Milan, Italy (Università degli Studi di Milano), graduating *cum laude* in 2011.

Subsequently, he was accepted in the Algant-Master program, spending the first year (2011-2012) at the University of Milan, and the second year (2012-2013) at the University of Bordeaux, France – then Université de Bordeaux 1. Gabriele obtained his master's degree from both universities in 2013; his thesis, with title "A study of quantum error-correcting codes derived from platonic tilings", was written under the supervision of Gilles Zémor.

In 2013, Gabriele was awarded an Algant-Doc PhD position at Leiden University and the University of Bordeaux, under the supervision of Ronald Cramer, Serge Fehr and Gilles Zémor; most of his research was carried out at the Centrum Wiskunde & Informatica (CWI) of Amsterdam, the Netherlands.

In 2017, he obtained a post-doc position within the Cryptology research group of CWI, Amsterdam.

Publications

I. G. Spini and G. Zémor, *Perfectly Secure Message Transmission in Two Rounds,* Proceedings of the 14th IACR Theory of Cryptography Conference (TCC), Beijing, PRC, Springer LNCS Volume 9985, pages 286–304, October 2016.

II. G. Spini and S. Fehr, *Cheater Detection in SPDZ Multiparty Computation,* Proceedings of the 9th International Conference on Information-

Theoretic Security (ICITS), Tacoma, WA, United States, Springer LNCS Volume 10015, pages 151–176, August 2016

III. G. Spini and G. Zémor, *Universally Secure Network Coding with Feedback,* Proceedings of the IEEE International Symposium on Information Theory (ISIT), Barcelona, Spain, pages 2339–2343, July 2016.

IV. R. Cramer, I. B. Damgård, N. Döttling, S. Fehr and G. Spini, *Linear Secret Sharing Schemes from Error Correcting Codes and Universal Hash Functions,* Proceedings of the 34th Annual IACR EUROCRYPT, Sofia, Bulgaria, Springer LNCS, Volume 9057, Part II, pages 313–336, April 2015.