Enhanced Threshold Schemes and their Applications

Thalia May Laing

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Information Security Group
School of Mathematics and Information Security
Royal Holloway, University of London

2018

Declaration

These doctoral studies were conducted under the supervision of Professor Keith M. Martin.

The work presented in this thesis is the result of original research I conducted, in collaboration with others, whilst enrolled in the School of Mathematics and Information Security as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Thalia May Laing
18th September 2017

Abstract

Techniques that distribute data across multiple players in such a way that a threshold number of players can reconstruct the data, whilst any fewer than the threshold cannot learn information about the data, are well established. These techniques are called threshold schemes and are important as they allow for increased availability, add redundancy and offer security without the reliance on cryptographic keys. With constrained devices restricted by low computational capabilities becoming more prevalent, it is important to discuss the use of threshold schemes in this setting. We study efficient threshold schemes and their applications in constrained devices and illustrate the compromise between efficiency and security.

We begin by using combinatorial techniques to propose an ideal, efficient, perfectly secure threshold scheme, which we analyse and compare to existing efficient threshold schemes. We show our scheme is efficient, with respect to the number of bitwise operations, compared to other schemes.

Next, we revisit a computationally secure threshold scheme, called AONT-RS, originally proposed in 2010. We generalise the scheme to allow for greater cryptographic flexibility and prove the scheme to be secure in the random oracle model. When compared to Krawczyk's scheme, the generalised scheme is more efficient, but requires greater assumptions in order to be proven secure. We then discuss extending AONT-RS to be robust, which prevents the unauthorised alteration of data.

Following this, we consider repairable threshold schemes, which enable recovery of a corrupted share without the help of a dealer. We consider and refine existing methods, analysing the storage costs and bandwidth required for each repair. We introduce a new metric to measure the repairability of the scheme, then consider secure regenerating codes and applying these techniques to repairable threshold schemes. We finish by comparing a range of techniques and highlight how they find a compromise between bandwidth and storage.

Finally, we propose localised multi-secret sharing schemes, which are multi-secret sharing schemes for an ordered set of players in which a sufficient number of sufficiently close players are able to recover secrets. We define threshold versions of localised multi-secret sharing schemes, provide lower bounds on the share size and give explicit constructions of schemes to show these bounds are tight. We then analyse a range of approaches to relaxing the model that provides a trade-off between the share size and the security provided by the scheme. Finally, we explore applying localised threshold multi-secret sharing schemes to RFID enabled supply chains.

Contents

List of Figures

List of Tables

Acknowledgements

Without my untiring supervisor, Keith, I would not have reached this point. He has guided me through the last four years, encouraging me to step outside my comfort zone and keeping me motivated throughout, all whilst maintaining a good sense of humour. For every email telling me not to fret, and for the continual encouragement, I am immeasurably grateful.

I have learnt much from, and am beholden to, my co-authors, Li, Maura and Doug, each of whom have provided me with invaluable advice and guidance.

I am fortunate to have been surrounded by good friends in the department, including my favourite Rachel, Rob and Victor, all of whom have taught and supported me. In particular, I reserve a special mention for Sam, who has fielded many questions and provided much cake, for Thyla, who has been a steady source of wisdom and company, and Pip, who has provided me with advice, wine and laughter. I also extend my thanks to my non-departmental friend Ben, who proofread as much as he could before comparing his brain to a small device with limited computing ability, power and memory.

Finally, on a personal note, I would be lost were it not for the ongoing love and support of my friends and, in particular, my family, to whom words of thanks always feel inadequate. So, to Chris, for making me countless cups of tea, listening with unwavering patience and making even the hardest of tasks seem achievable; to Mum, Dad, Mandi and Mick, for their enduring belief and confidence in me; and to my phenomenal sisters, Claire, Pari, Cara, Marissa and Sian, for being my best friends and my comfort; thank you.

Chapter 1

Introduction

1.1 Motivation

Consider a scenario in which a secret key provides access to a number of important, confidential files. This set-up has three main drawbacks. Firstly, the security of the files relies on the secret key being stored securely; if an adversary is able to gain access to the key, they are able to access the files. Once access to the files has been gained, the adversary can read, alter, and/or delete any of the confidential information. The second drawback is that, if the key is lost, access to the files is lost with it. Finally, complete trust is placed in the entity that holds the key; if the important files do not belong exclusively to them, how can the entrusted entity be prevented from exploiting their responsibility?

To illustrate this scenario, consider a government with a secret key used to authorise the launching of nuclear missiles. The government wishes to avoid storing the key somewhere where an adversary may be able to gain access to it, for obvious reasons. They discuss giving the key to an individual, say the president, but have two main concerns with this idea. Firstly, they fear the president may be unavailable if the key is required, or that an adversary may remove the president, meaning the key is lost to the government and the nuclear missiles cannot be launched. Secondly, they worry the president may exploit his responsibility and use the key to launch the nuclear missiles, even if it is not absolutely necessary.

Instead of giving sole responsibility to the president, the government decide to distribute the key amongst three of its members by giving each of the three members a share of the key. The distribution is done in such a way that any two of the three members can collaborate and decide to use and recover the key, but any individual cannot do so alone.

This solution of splitting the key addresses a number of the drawbacks previously highlighted. Firstly, an adversary would be unable to learn the key by gaining access to just one share; instead, they would be required to learn the shares of at least two of the three chosen members of government. Secondly, even if one member becomes unavailable, the key is not lost as the other two members can collectively reconstruct it. Finally, as no individual knows the key, nobody is solely responsible for, or able to exploit, their knowledge of the key.

This process of sharing a key, or a secret, amongst a set of n *players* (which could be devices, servers or people), such that authorised sets of players can collectively recover the secret, whilst unauthorised sets of players cannot, is called *secret sharing*. In particular, a sharing of the secret that allows only sets of t players to recover the secret is called a (t,n)-*threshold scheme*, where the threshold, t, is the number of players required to collaborate in order to recover the secret. The previous example of distributing a key used to authorise the launching of nuclear missiles is an application of a $(2,3)$-threshold scheme, as three members of the government are given shares, and any two of the three are able to recover the key. Threshold schemes are a well-studied cryptographic primitive that were first introduced independently by Shamir and Blakley in 1979 [10, 91].

Now, more than ever, threshold schemes are of increasing importance. Given the growth and ubiquity of the Internet of Things, there are an increasing number of constrained devices that store an ever-growing amount of personal information and data. These devices may be small and mobile with limited computing ability, power and memory and, most importantly, may be vulnerable to compromise. To store confidential data solely on one of these devices would be a high-risk strategy and could lead to a host of problems. For example, an adversary may physically steal a mobile device or exploit a vulnerability, or a device may run out of power, making the data stored on it unavailable. In either case, securely distributing the data amongst multiple devices will increase confidentiality, provide reliability and enable a more robust system.

Providing security in this distributed manner does, however, have its limitations. In particular, there are lower bounds on the size of the data each device must store as their share, lower bounds on the amount of randomness required as input in order for the scheme to be secure, and potentially heavy computation required from each device in the system. Given the aforementioned limitations of the devices, it is necessary to explore threshold schemes that are efficient. That is, it is vital to seek schemes which guarantee security whilst meeting these lower bounds and requiring minimal computation. In some applications, increasing the efficiency of the system may only be possible by relaxing the security definition, but it is vital that the resulting security be well understood and remain sufficient for the application.

Threshold schemes also introduce some of their own, unique problems. For example, consider an adversary who gains access to one device which is a player in a threshold scheme. Although they are unable to learn any of the distributed data from the share stored on the individual device, the adversary may be able to alter the share on the device. If this device then contributes to recovering the data, the corrupted share it supplies may lead to some incorrect data being recovered by the authorised set of devices. One would hope the threshold scheme can recognise incorrect data has been recovered and even recover the correct data whilst also recognising which device submitted a corrupted share. Threshold schemes achieving this are called *robust*. After identifying which share has been corrupted, it may be desirable for the associated device to recover the correct share they were originally storing. Schemes able to do this without the help of a trusted third party are called *repairable*.

Threshold schemes have a number of applications besides the examples of secure storage, outlined above. Further applications include key escrow [38], and their application in building other cryptographic primitives, such as secure multiparty computation [46, 81, 28] and electronic voting [89, 49].

The aim of this thesis is to explore efficient threshold schemes and their extensions. In a number of cases, we will consider compromising security in order to improve efficiency.

1.2 Chapter overviews

We begin with Chapter 2, which introduces basic principles and notions to be used throughout the thesis. An introduction to the necessary definitions of information security as well as a primer on threshold schemes and information dispersal algorithms is given.

This is followed by Chapter 3, which focuses on perfect threshold schemes, which are threshold schemes that achieve information theoretic security. Here, we present an enhancement of a perfect threshold scheme originally proposed in a patent by HP [21]. We conduct a security and efficiency analysis of the scheme, concluding that the scheme is both ideal (meaning it requires the minimal amount of memory possible), and optimal with respect to the number of random bits required. Following this, we explore a number of other threshold schemes in the literature and compare their efficiencies. We conclude our scheme is more efficient than Shamir's threshold scheme [91], and has a more efficient recovery of the secret than the scheme by Kurihara *et al.* [62], whilst having a not much slower algorithm for sharing the secret.

Following this, Chapter 4 considers relaxing the security of threshold schemes from information theoretic to computational. Computationally secure threshold schemes are able to achieve smaller share sizes than perfect schemes and are ideal in settings where either the devices storing the data have small memory, or when the data to be stored is comparably large. We present a generalised version of the AONT-RS scheme by Plank and Resch [85], called AONT-RS0, and conduct the first formal security analysis. We then compare AONT-RS0 to Krawczyk's benchmark computationally secure threshold scheme [60]. We find AONT-RS0 is more efficient than Krawczyk's scheme, but achieves this by assuming the internal hash function is indistinguishable from a random oracle. We then discuss an adversary who is able to corrupt shares in order to prevent the original data from being recovered. By utilising two different techniques previously used to extend Krawczyk's scheme to be secure against such as adversary, we extend AONT-RS0 to be robust, and therefore able to recover the correct data in the face of such an adversary. We discuss how these two techniques vary in security and efficiency, with the more efficient scheme making greater assumptions and thereby achieving a weaker, yet still sufficient, level of security.

After considering schemes that are able to recognise corrupted shares and recover the original data in spite of this, in Chapter 5 we discuss schemes in which players are able to recover their share without the presence of a trusted third party, but with the help of other players. Schemes with this property are called repairable threshold schemes. We present, analyse and enhance a number of existing repairable threshold schemes before exploring the field of regenerating codes, which is able to offer a number of competitive solutions. We conclude by identifying the direct trade-off between storage and bandwidth requirements, and dispute the suggested trade-off between storage and recoverability conjectured in [97].

Chapter 6 proposes and defines localised threshold multi-secret sharing schemes (LMSS), which enable a threshold set of ordered players, that are close to each other in the underlying ordering, to recover some shared secret. If the set of players are either fewer than the threshold, or insufficiently close, they are unable to recover the secret. We provide a motivating scenario for such schemes, then discuss bounds and present constructions that meet these bounds. We then relax the security, in a number of measurable ways, in order to achieve more efficient schemes, then combine these techniques to give a flexible LMSS and explore their application to RFID enabled supply chains, an application originally considered in [56].

Finally, we summarise the work and explore further research questions in Chapter 7.

1.3 Publications

The research in Chapter 3 is joint work with Liqun Chen and Keith Martin [25], and was presented at CANS 2016.

Chapter 4 is based on the paper ‘Revisiting and extending the AONT-RS scheme: a robust computationally secure secret sharing scheme’ [26]. This work is also joint with Liqun Chen and Keith Martin and was presented at AfricaCrypt 2017.

Chapter 5 is joint work with Doug Stinson and has been accepted to appear in the Journal of Mathematical Cryptography [64].

Finally, Chapter 6 is joint work with Keith Martin, Doug Stinson and Maura Paterson and is based on the paper ‘Localised multi-secret sharing’, which appears in [63].

Chapter 2

Preliminaries

Contents

This chapter will introduce the core principles, notation and definitions that will be used throughout. We begin in Section 2.1 by introducing some of the key principles of information security and cryptography. We then introduce symmetric-key encryption schemes in Section 2.2, then define and explore secret sharing schemes and, in particular, threshold schemes in Section 2.3. We complete this chapter by discussing information dispersal algorithms in Section 2.4.

2.1 Principles of information security and cryptography

2.1.1 Basic principles

The desire, or need, to restrict access to information is familiar to many people; we are increasingly surrounded by scenarios where information must be stored, or transmitted, securely. Information security considers the protection of information, while cryptography provides the techniques used to secure information. We use the following terminology and notation to discuss cryptographic concepts.

A cryptographic *primitive* is a process that provides a number of security services. Primitives can be used as building blocks to construct more complex cryptographic structures. Two examples of primitives discussed include symmetric-key encryption schemes in Section 2.2, and cryptographic hash functions in Section 4.2.2.

A cryptographic *algorithm* is a specification of a primitive. It is a list of computational steps that can be implemented. A number of algorithms are defined throughout; the first two of which are the share and recover algorithms of Krawczyk's computationally secure threshold scheme, given in Figure 2.3. If an algorithm A is *deterministic*, meaning that each input to A will result in the same output, we write $x \leftarrow A(\cdot)$. If A is probabilistic, we write $x \overset{\$}{\leftarrow} A(\cdot)$, which means to choose x according to the distribution induced by algorithm A. If A is a finite set, $x \overset{\$}{\leftarrow} A$ means to choose x uniformly at random from A.

A cryptographic *scheme* or *system* refers to a collection of algorithms and the related infrastructure. The first cryptographic scheme we consider is a symmetric-key encryption scheme, presented in Definition 2.2.1. We then consider a number of threshold schemes, which are formally presented in Definition 2.3.2.

Finally, a cryptographic *protocol* is a sequence of message exchanges and operations between multiple players. One example of a protocol considered is the enrolment repairable threshold scheme repair protocol in Section 5.3.1.1.

2.1.2 Security goals

Before exploring any cryptographic techniques in detail, it must first be established what is meant by security. At a fundamental level, security is about providing a number of core security services. We focus on three security services: confidentiality, integrity and availability.

Confidentiality is the assurance that no one other than the intended recipient(s) can read the data. In other words, the data is private between the sender and the receiver.

Besides keeping data private, it can be important to know the data has not been tampered with. *Integrity* is the assurance that the data has not been altered, either maliciously or accidentally, in an unauthorised way.

The last security service we discuss is availability. *Availability* ensures that accessibility and usability are available upon demand by an authorised entity. The loss of availability as a result of an adversary's interventions is commonly referred to as a *denial of service attack*.

2.1.3 Adversaries

An *adversary* is a malicious entity who wishes to prevent a system from achieving its security goals. Before we evaluate the security of a system, it is important to first establish what the assumptions on, and the behaviour of, any adversaries are. If we underestimate the adversary's capabilities, the security of the system may be inadequate. Thus, it makes sense to err on the side of caution and assume a worst-case scenario with a strong adversary.

First and foremost, it is assumed the adversary knows the scheme it is attacking. That is, the details of all algorithms and protocols defined in the scheme are publicly available and known to the adversary. Instead of the scheme being private, there will be some information, such as a *cryptographic key*, that is unknown to the adversary. The security of the scheme relies on the privacy of this information. If it is made publicly available, the scheme cannot hope to achieve any of its security goals.

After assuming an understanding of the scheme, we must establish what other information the adversary is given. This is historically categorised into four theoretical attacks. The four attacks, increasing in power, are:

- *Ciphertext only attacks*: as well as the understanding of the scheme, the adversary has access to samples of ciphertext.
- *Known plaintext attack*: the adversary is given some arbitrary plaintext and the corresponding ciphertext.
- *Chosen plaintext attack*: the adversary is given some plaintext and ciphertext pairs, where the plaintexts are chosen by the adversary.
- *Chosen ciphertext attack*: the adversary is given some plaintext and ciphertext pairs, where either the plaintexts or ciphertexts are chosen by the adversary.

After assuming an understanding of the scheme and defining what other information the adversary has access to, we must establish the allowed adversarial behaviour. This defines what actions the adversary is permitted to undertake. An adversary may be restricted to eavesdropping on messages exchanged during a protocol, or they may have greater capabilities and be able to corrupt a player (for example, a device or a server), or multiple players, in the scheme. When a player is corrupted by an adversary, we assume they are henceforth under the control of the adversary and all information stored by the player is accessible to them. After corrupting a player, a *passive* adversary must correctly follow the protocol (or algorithm), but attempts to learn information that should remain private. This is a fairly weak adversary to consider, but models information leakage in a system. In contrast, an *active* adversary can deviate from the protocol, according to their instructions. This is a stronger adversary to consider and providing security in the presence of an active adversary is preferable. Throughout this thesis, we consider adversaries with differing abilities and behaviours.

The computational complexity of the adversary must also be considered. A *polynomial time* adversary is one allowed to run in (probabilistic) polynomial time. Schemes achieving security against this type of adversary are called *computationally secure*. In contrast, a *computationally unbounded* adversary has no computational or time limits. Schemes achieving security against this power of adversary are called *information theoretically secure*.

Finally, we consider the corruption strategy of an adversary. In settings with multiple players, the adversary may be able to corrupt more than one player. If so, it is important to define when and how the players are corrupted. There are two main strategies to consider. A *static corruption model* assumes the adversary is given a fixed set of players whom it controls. These players are corrupted and remain corrupted throughout, whereas the other players are honest and will remain so throughout. The alternative is an *adaptive corruption model*. Rather than being given a fixed set of player, this model allows an adversary to corrupt players during the procedure. An adversary can choose which players to corrupt depending on its view of the execution. As before, when a player is corrupted, they remain corrupted throughout.

2.1.4 Security games

In some circumstances, it may be useful to define different notions of security as *games*. Each game is executed by an entity called the *challenger*. The adversary is modelled as a probabilistic polynomial time algorithm $\mathcal{A}$, which will be called by the challenger with some input. The adversary $\mathcal{A}$ must then submit an output as response within a polynomial number of steps. Games may consist of multiple procedures and the challenger may call $\mathcal{A}$ numerous times. Between each call, the adversary will maintain its state. Games are designed to model realistic threats and the evolution of a system. For example, each call of $\mathcal{A}$ by the challenger models the adversary performing tasks at different points during the execution of the system.

During games, the challenger may provide the adversary with access to an *oracle* for a given function. Each oracle allows $\mathcal{A}$ to submit queries to the challenger and receive the corresponding output of the function for their chosen input. This enables the adversary to learn the outcome of a function without learning the private information used to compute the output. Oracle access models an adversary learning information by observing a system, witnessing messages being exchanged, or by corrupting players and learning information stored by them.

Some games end with the challenger outputting either ‘0’ or ‘1’, determined by the information submitted to the challenger by the adversary during the finalise procedure. A ‘1’ is synonymous with ‘true’ and means the output from the adversary is correct. In other cases, we write $b' = b$ as the end of the game. This denotes an equality comparison between b and b', where the game returns ‘true’ only if b and b' are equal.

2.1.5 Encryption schemes

An encryption scheme provides a method of translating some data, called a *plaintext* and denoted by M, into a *ciphertext*, denoted C, and back again. Even after seeing the ciphertext, the plaintext should remain confidential. Encryption schemes rely on *cryptographic keys*, with one being used for encryption and one for decryption. There exist two types of cryptographic schemes: symmetric and asymmetric.

In a *symmetric-key* encryption scheme, the same key is used for both encryption and decryption. With knowledge of the one key, someone can both encrypt a plaintext and decrypt a ciphertext. Because of this, if two entities wish to communicate privately, they must have a shared key. Importantly, this key must be private to the two parties; if an adversary has access to the key, no security can be expected.

If the scheme is *asymmetric*, the keys used for encryption and decryption are distinct. This ground-breaking technique was first developed in 1973 by mathematicians at GCHQ but was not publicised. Similar ideas, however, were developed by Diffie and Hellman in 1976 [32] and expanded on by Rivest, Shamir and Adleman in 1978 [87]. An asymmetric scheme is sometimes known as a *public-key* scheme as the encryption key can be made public. This is possible because these schemes rely on the idea that it is computationally infeasible to derive the decryption key from the encryption key, dependent on the computational infeasibility of an underlying computational problem. The proposal of public-key systems enabled two entities to send and receive data without having previously agreed on a symmetric key.

In this thesis, we use symmetric-key encryption schemes, which are formally defined in Section 2.2.

2.1.6 Information theory and the entropy function

Information theoretic notation will be useful in portraying the concept of perfect secrecy, which is the idea that a computationally unbounded adversary learns no additional information about a plaintext after learning the corresponding ciphertext. We briefly introduce information theory and the concept of entropy here.

Information theory was introduced by Shannon in 1948 [92]. One important notion of information theory is that of *Shannon entropy*, which quantifies the amount of uncertainty involved in a system. Specifically, for a discrete random variable $\boldsymbol{X}$ with possible values $\mathcal{X} = \{x_1, \ldots, x_n\}$ and probability mass function $\Pr(\boldsymbol{X})$, the entropy, denoted by $H(\boldsymbol{X})$, can be written as

$$H(\boldsymbol{X}) = -\sum_{x \in \mathcal{X}} \Pr(x) \log_b \Pr(x),$$

where b is the base of the logarithm. We often choose $b = 2$ and refer to the units of entropy as bits. If a system has the maximum amount of uncertainty, we say it has *maximal entropy.* For example, if x is chosen uniformly at random from $\{0,1\}^\lambda$, for some positive integer λ, x has maximal entropy and $H(x) = 2^\lambda$.

Conditional entropy is the amount of uncertainty involved in a system, given that some information is known. Specifically, for discrete random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$, with possible values $\mathcal{X} = \{x_1, \ldots, x_n\}$ and $\mathcal{Y} = \{y_1, \ldots, y_m\}$ and probability mass functions $\Pr(\boldsymbol{X})$ and $\Pr(\boldsymbol{Y})$ respectively, the conditional entropy, denoted by $H(\boldsymbol{X} \mid \boldsymbol{Y})$, can be written as

$$H(\boldsymbol{X} \mid \boldsymbol{Y}) = \sum_{\substack{x \in \mathcal{X} \\ y \in \mathcal{Y}}} \Pr(x, y) \log_b \frac{\Pr(x)}{\Pr(x, y)}.$$

The interested reader is directed to [92] for an in depth explanation of the entropy function.

The notion of perfect secrecy in a symmetric-key encryption scheme, and an example of a scheme that achieves this, is given in Section 2.2.1. The notion of perfect secrecy in a threshold scheme is given in Definition 2.3.4.

2.1.7 Constrained devices

This thesis considers a number of applications of threshold schemes in different settings. In some settings, the devices involved may be lightweight and have a number of constraints. These constraints include:

1. *Power:* Each device may have a small amount of energy and be unable to recharge. Thus, once the power has been drained, the device will terminate and no longer be able to communicate data. Both computations and communicating deplete energy from the device and so it is desired both are kept to the minimum level possible.
2. *Memory storage:* The devices may have limited memory to store information. It is therefore desirable that the amount of information needed to be stored is kept to a minimum.
3. *Vulnerable to compromise:* Each device may have minimal security protection. This means the device may be vulnerable to compromise, resulting in the confidential data it possesses being exposed.

Given these restrictions, it is important to carefully consider the computational power and memory required for each scheme.

One way to measure how complex an algorithm is is to use *big 'O' notation*, sometimes called asymptotic notation. Big 'O' notation is used to describe the behaviour of a function, or algorithm, when the input size grows. An introduction to big 'O' notation can be found in [2]. We will use big 'O' notation to analyse the bitwise complexity of a number of different algorithms. We do, however, note that big 'O' notation may not be very useful at describing the behaviour of functions with small inputs, as it only considers the leading, largest term, and does not consider any coefficients or smaller terms.

2.2 Symmetric-key encryption schemes

We now formally consider the definition and security of symmetric-key encryption schemes. Recall from Section 2.1.5 that, in a symmetric-key encryption scheme, the same key is used for both encryption and decryption.

Definition 2.2.1. *A* symmetric-key encryption scheme $\mathcal{E}$ *consists of three algorithms: KeyGen, Enc and Dec, as follows:*

- *The key-generation algorithm KeyGen is a probabilistic algorithm that outputs a key k chosen uniformly at random from a key space $\mathcal{K} \subseteq \{0,1\}^\lambda$, where λ is the security parameter. We denote this generation as $k \overset{\$}{\leftarrow} \mathcal{K}$.*
- *The encryption algorithm Enc takes as input a key $k \in \mathcal{K}$ and a plaintext message M, chosen from a plaintext space $\mathcal{M}$, and outputs a ciphertext C from a ciphertext space $\mathcal{C}$. Denote $Enc_k(M)$ as the encryption of the message M under the key k.*
- *The decryption algorithm Dec takes as input a key $k \in \mathcal{K}$ and a ciphertext $C \in \mathcal{C}$ and outputs a plaintext $M \in \mathcal{M}$. Denote $Dec_k(C)$ as the decryption of the ciphertext C under the key k.*

A symmetric-key encryption scheme $\mathcal{E}$ must satisfy the *perfect correctness* requirement: for every key $k \in \mathcal{K}$ and $M \in \mathcal{M}$,

$$Dec_k(Enc_k(M)) = M.$$

That is, encrypting a plaintext message M under k and then decrypting the resulting ciphertext C under the same key k, assuming no decryption errors occur, must result in the return of the original plaintext M.

Essentially, a symmetric-key encryption algorithm inputs a sequence of plaintext bits, performs a series of operations on these bits, then outputs a sequence called the ciphertext. The decryption algorithms inputs the ciphertext and reverses the operations to output the original sequence of plaintext bits.

We will normally assume that $\mathcal{K} = \{0,1\}^\lambda$, and $M = C = \{0,1\}^*$, where $\{0,1\}^*$ denotes a binary string of finite length.

Symmetric-key encryption schemes can either be *stream ciphers*, in which the plaintext is processed one bit at a time, or *block ciphers*, that operate on blocks of bits. An example of a stream cipher is the one-time pad, which is defined in Definition 2.2.2. An example of a block cipher is AES [30]. We do not focus on block ciphers here, but an introduction to their modes of operation is given in Section 4.2.1.

2.2.1 Perfect secrecy

In an encryption scheme, *perfect secrecy* is the notion that a computationally unbounded adversary learns no extra information about the plaintext after seeing the ciphertext. Using information theoretic notation (introduced in Section 2.1.6), if $\boldsymbol{M}$ denotes the discrete random variable corresponding to the choice of the plaintext and $\boldsymbol{C}$ denotes the corresponding ciphertext, the notion of perfect secrecy in an encryption system can be written as

$$H(\boldsymbol{M}) = H(\boldsymbol{M} \mid \boldsymbol{C}),$$

meaning that an adversary with an amount of uncertainty about the plaintext, denoted by $H(\boldsymbol{M})$, has the same amount of uncertainty about the plaintext even if they are given the ciphertext.

We now give an example of a symmetric-key stream cipher achieving perfect secrecy, called a one-time pad. In the following definition, the symbol '$\oplus$' denotes *exclusive or*, better known as XOR, which is essentially the bitwise addition of binary numbers modulo two.

Definition 2.2.2. *The following defines a* one-time pad *(OTP). Fix an integer $\ell > 0$. Let the message space $\mathcal{M}$, key space $\mathcal{K}$ and ciphertext space $\mathcal{C}$ all be $\{0,1\}^\ell$.*

- *Let KeyGen choose a key $k \xleftarrow{\$} \mathcal{K} = \{0,1\}^\ell$ according to a uniform distribution (meaning that each of the 2^ℓ strings in the key space is chosen with probability $2^{-\ell}$).*
- *Given a key $k \in \mathcal{K}$ and a plaintext message $M \in \mathcal{M}$, define $Enc_k(M) = k \oplus M = C$.*
- *Given a key $k \in \mathcal{K}$ and a ciphertext message $C \in \mathcal{C}$, define $Dec_k(C) = k \oplus C = M$.*

2.2.2 Computational security

Encryption schemes are primarily concerned with providing confidentiality. Here, we define two security properties, via security games, that characterise the confidentiality of a symmetric-key encryption scheme against a polynomial time adversary. The two properties are called indistinguishability and key-unrecoverability and are summarised in Figure 2.1. Both games aim to model a polynomial time adversary launching a *chosen-plaintext* attack. Both definitions of indistinguishability and key-unrecoverability are taken from [8].

2.2.2.1 Indistinguishability

As in [8], the game describing indistinguishability of an encryption scheme $\mathcal{E}$ is given as Game 2.1 and consists of three procedures: *Initialise*, *Deal* and *Finalise*. Intuitively, the challenger randomly chooses a bit b and generates a key k. The adversary can then submit any two distinct plaintext messages $M_0, M_1 \in \mathcal{M}$ of equal length. The challenger encrypts M_b and returns the ciphertext C to the adversary. The adversary may repeat this procedure multiple times, depending on their instructions. The adversary then outputs a guess b' for b and wins if $b' = b$.

Informally, a scheme achieving the indistinguishably property is such that no efficient algorithm can distinguish which of the two submitted plaintext messages were encrypted.

Let $\mathcal{A}$ be an adversary playing the indistinguishability game against a symmetric-key encryption scheme $\mathcal{E}$. Call $\mathcal{A}$ an *indistinguishability adversary* and the corresponding challenger an *indistinguishability challenger*. Let $\Pr[Ind^{\mathcal{A}}]$ denote the probability $\mathcal{A}$ outputs the correct guess b' for b during the finalise procedure. Define the advantage of $\mathcal{A}$ as

$$\mathbf{Adv}_{\mathcal{E}}^{Ind}(\mathcal{A}) = 2 \cdot \Pr[Ind^{\mathcal{A}}] - 1. \tag{2.2.1}$$

Say the encryption scheme $\mathcal{E}$ has the property of indistinguishability if the advantage of $\mathcal{A}$ is *negligible*, meaning that, for the given security parameter λ, the advantage of $\mathcal{A}$ vanishes faster than the inverse of any polynomial.

We said the adversary is allowed to repeat the deal procedure multiple times, dependent

Game 2.1: *Ind*

```
Procedure Initialise
1     $b \xleftarrow{\$} \{0,1\}$;
2 $k \xleftarrow{\$} KeyGen\{0,1\}^{\lambda}$;
Procedure Deal($M_0, M_1$)
1     if $M_0 = M_1, |M_0| \neq |M_1|$, or
        $M_0, M_1 \notin \mathcal{M}$ then
          return $\perp$;
2     else $C \xleftarrow{\$} Enc_k(M_b)$;
3     return $C$;
Procedure Finalise($b'$)
1     if $b' = b$ then
          return true;
2     else return false;
```

Game 2.2: *Key*

```
Procedure Initialise
1     $b \xleftarrow{\$} \{0,1\}$;
2 $k \xleftarrow{\$} KeyGen\{0,1\}^{\lambda}$;
Procedure Deal($M$)
1     if $M \notin \mathcal{M}$ then
          return $\perp$;
2     else $C \xleftarrow{\$} Enc_k(M)$;
3     return $C$;
Procedure Finalise($k'$)
1     if $k' = k$ then
          return true;
2     else return false;
```

Figure 2.1: Indistinguishability and key-unrecoverability games used to define security notions in a symmetric-key encryption scheme, previously defined in [8].

on their instructions. Rather than allowing $\mathcal{A}$ to repeat the procedure, we can consider a weaker adversary who is limited to only calling the deal procedure once. We call an adversary bounded in this way an *ind-1 adversary* and the corresponding challenger an *ind-1 challenger*. The advantage of an ind-1 adversary is computed as in (2.2.1). Any scheme achieving indistinguishability is also ind-1 secure, as an ind-1 adversary is weaker than an indistinguishability adversary.

2.2.2.2 Key-unrecoverability

As in [8], the key-unrecoverability game for a symmetric-key encryption scheme $\mathcal{E}$ (given here as Game 2.2) consists of three procedures: *Initialise*, *Deal* and *Finalise*. Intuitively, the challenger randomly chooses a bit b and generates a key $k \in \mathcal{K}$. The adversary can then submit any plaintext message $M \in \mathcal{M}$ during the deal procedure and will receive C, the encryption of M under k, in return. The adversary may repeat this procedure multiple times, depending on their instructions. The adversary then outputs a guess k' for k and wins if $k' = k$.

Informally, for a scheme achieving key-unrecoverability, no efficient algorithm can determine the key from sets of corresponding plaintext and ciphertext pairs.

Let $\mathcal{A}$ be an adversary playing the indistinguishability game against a symmetric-key encryption scheme $\mathcal{E}$. Call $\mathcal{A}$ a *key-unrecoverability adversary* and the corresponding challenger a *key-unrecoverability adversary.* Let $\Pr[Key^{\mathcal{A}}]$ denote the probability $\mathcal{A}$ outputs the correct guess k' for k during the finalise procedure. Define the advantage of $\mathcal{A}$ as

$$\mathbf{Adv}_{\mathcal{E}}^{Key}(\mathcal{A}) = 2 \cdot \Pr[Key^{\mathcal{A}}] - 1. \tag{2.2.2}$$

Say the encryption scheme $\mathcal{E}$ has the property of key-unrecoverability if the advantage of $\mathcal{A}$ is negligible.

As before, a key-unrecoverability adversary is able to call the deal procedure multiple times. Again, we can consider a weaker adversary and restrict $\mathcal{A}$ to calling the deal procedure only once. Call this restricted adversary a *key-1 adversary* and the corresponding challenger a *key-1 challenger.* The advantage of a key-1 adversary is computed as in (2.2.2). Any scheme achieving key-unrecoverability is also key-1 secure.

2.2.3 Relating security notions

We conclude this introduction to symmetric-key encryption schemes by commenting on the relationship between the aforementioned security notions.

A symmetric-key encryption scheme that achieves perfect secrecy also achieves indistinguishability, but does not necessarily achieve key-unrecoverability.

To illustrate this, consider the OTP from Definition 2.2.2. The OTP has perfect secrecy against an unbounded adversary; that is, seeing the ciphertext reveals no new information about the plaintext. A polynomial time adversary is weaker than an unbounded adversary, and so, if we consider a weaker, polynomial time adversary against the OTP, we can comment on the OTP and the properties of indistinguishability and key-unrecoverability.

Obviously, the OTP achieves indistinguishably as, prior to seeing the ciphertext, the adversary knows only that either M_0 or M_1 was encrypted but, as the OTP has perfect secrecy, after receiving the ciphertext they learn no extra information and therefore cannot determine which plaintext was chosen. Any scheme achieving indistinguishability is also ind-1 secure and so the OTP is ind-1 secure.

However, the OTP is not key-1 secure. This is because, during the deal procedure of the key-unrecoverability game, the adversary learns a plaintext and the corresponding ciphertext. Now, because $M \oplus k = C$ and $C \oplus k = M$, the adversary is able to recover the key by computing $M \oplus C = k$. Intuitively, despite the OTP achieving perfect secrecy, it is not secure under a known plaintext attack, or any stronger theoretical attack. Because the OTP is not key-1 secure, it does not achieve the stronger security of key-unrecoverability.

Conversely, neither indistinguishability nor key-unrecoverability guarantee perfect secrecy.

Now, we relate ind-1 and key-1 security. It is mentioned in [85] that schemes in which the ratio of message length to key length is large (rather than equal to one, as is true in the OTP), then it may be the case that ind-1 security implies key-1 security. However, the authors were unable to find neither a proof nor a counterexample that this is true, and there does not appear to be any solution in the literature since.

To summarise, perfect security implies indistinguishability, which in turn implies ind-1 security. However, perfect security does not imply key-unrecoverability, but key-unrecoverability does imply key-1 security. There are no concrete results relating ind-1 and key-1 security.

Moving forwards, we assume symmetric-key encryption schemes have both ind-1 and key-1 security. We note that conventional encryption schemes, including AES, achieve both security properties and thus it is a reasonable assumption to assume the existence of such a scheme [5].

2.3 Threshold schemes

In this section, we begin by defining secret sharing schemes. We then define and focus on threshold schemes, which are a type of secret sharing scheme, and discuss the relevant security notions.

Intuitively, a *secret sharing scheme* is a means of distributing a *secret* amongst a set of *players* so that *authorised* subsets of players can recover the secret, whereas the shares belonging to *unauthorised* subsets do not reveal any information about the secret.

Definition 2.3.1. *Given a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n players, a* secret sharing scheme Π *is a pair of algorithms, Share and Recover, and an* access structure*, which is a collection of sets of players: $\Gamma \subseteq 2^{\mathcal{P}}$. The access structure Γ must be* monotonic*, meaning that for $A, B \subseteq \mathcal{P}$ with $A \subseteq B$, if $A \in \Gamma$ we have $B \in \Gamma$ also.*

- *The Share algorithm is a probabilistic algorithm run by a* dealer *that takes as input a secret s from some* secret space $\mathcal{S}$ *and the access structure Γ and outputs n elements $v_1, \ldots, v_n$. Player P_i, for $1 \leq i \leq n$, is given v_i as their* share.
- *The Recover algorithm is a deterministic algorithm that takes as input a set A of player's shares and outputs a value $s' \in \{\mathcal{S} \cup \perp\}$, where $\perp$ represents an error.*

The secret sharing scheme should be both

- recoverable*, meaning that if A is an authorised subset of players, the secret output by the recover algorithm is the original secret. In other words, if $A \in \Gamma$ then $s' = s$, and*
- private*, meaning that if if A is an unauthorised subset of players, they cannot collectively recover the secret. I.e., if $A \notin \Gamma$ then $s' \neq s$.*

We will focus almost completely on (t, n)-threshold schemes, which are n player secret sharing schemes where the access structure Γ consists of all subsets containing at least t players. Intuitively, a (t, n)-threshold scheme guarantees that any t of the n players can recover the secret, whereas any fewer than t players will be unable to recover the secret. Such schemes were proposed independently by Blakley and Shamir in 1979 [10, 91].

Definition 2.3.2. *Let $t \in \mathbb{N}$, such that $1 \leq t \leq n$. A (t, n)-*threshold scheme *is a secret sharing scheme with n players such that the access structure Γ consists of all subsets of $\mathcal{P}$ that contain at least t players, so $\Gamma = \{A \subseteq 2^{\mathcal{P}} : |A| \geq t\}$. A (t, n)-threshold scheme is recoverable, meaning that any set of at least t players can collectively recover the shared secret, and private, meaning that any fewer than t players cannot recover the secret.*

In Definition 2.3.2, t is such that $1 \leq t \leq n$. We note that, if $t = 1$, any player could individually recover the secret. Thus, in general, we assume that $1 < t < n$.

We will also assume that the secret is ‘large’. That is, that $|s| > |n|$. This will be generally be true in the cases we consider, as the shortest s shared is likely to be the length of a key, so at least 128 bits, whereas n is likely to be much smaller than this. Exploring sharing a small secret (for example, a one bit secret), and the efficiencies of such a scheme is considered in [17, 16].

The following construction is due to Shamir and yields a (t, n)-threshold scheme for positive integers t and n with $1 \leq t \leq n$. It was one of the first proposed constructions for threshold schemes and widely used and standardised [52].

Construction 2.3.3. *Let $\mathcal{P}$ be a set of n players, and let $p > n$ be a prime number. Let the secret space $\mathcal{S}$ be equal to the finite field of p elements, $\mathbb{Z}_p$. For a given secret $s \in \mathbb{Z}_p$, the share and recover algorithms comprising Shamir's threshold scheme are as follows.*

- *Share: Select $t-1$ values $r_1, r_2, \ldots, r_{t-1}$ uniformly at random from $\mathbb{Z}_p$, and let $f \in \mathbb{Z}_p[x]$ be the polynomial defined by*

  $$f(x) = r_{t-1}x^{t-1} + r_{t-2}x^{t-2} + \cdots + r_1 x + s.$$

  We give each player P_i, for $1 \leq i \leq n$, the share $v_i = f(i)$.

- *Recover: A collection of t (or more) players perform polynomial interpolation on their shares in order to recover the polynomial f and hence determine the secret $s = f(0)$.*

Shamir's scheme is recoverable, as any set of t players can recover the secret via interpolation. Shamir's scheme is also private, as for any set of $t-1$ or fewer players, and for any element $s' \in \mathbb{Z}_p$, there exists a polynomial of degree at most $t-1$ consistent with their shares and having constant term s'. Thus the shares of an unauthorised set of players yield no information about the true value of s.

To illustrate Shamir's threshold scheme, we present Example 2.3.1.

Example 2.3.1. Let $\mathcal{P}$ be a set of $n = 5$ players and let $t = 3$, so any three players can collaborate to recover the secret. Let $p = 7$, so all computations are computed modulo 7, and choose $s = 4 \in \mathbb{Z}_7$.

To share the secret, choose $t-1=2$ values at random from $\mathbb{Z}_7$. Let these values be $r_1=6$ and $r_2=2$. Let $f \in \mathbb{Z}_7[x]$ be

$$\begin{aligned} f(x) &= r_2x^2 + r_1x + s \\ &= 2x^2 + 6x + 4, \end{aligned}$$

and give player P_i, for $1 \leq i \leq 5$, the share v_i, where $v_i = f(i)$. So P_1 receives $v_1 = 5$, player P_2 receives $v_2 = 3$, P_3 receives $v_3 = 5$, P_4 receives $v_4 = 4$ and P_5 receives $v_5 = 0$.

Assume the three players P_2, P_3 and P_5 collaborate to recover the secret. They combine their shares to compute f via Lagrange interpolation, as follows:

$$\begin{aligned} f(x) &= v_2\frac{(x-3)}{(2-3)}\frac{(x-5)}{(2-5)} + v_3\frac{(x-2)}{(3-2)}\frac{(x-5)}{(3-5)} + v_5\frac{(x-2)}{(5-2)}\frac{(x-3)}{(5-3)} \\ &= 3\frac{(x-3)}{6}\frac{(x-5)}{4} + 5\frac{(x-2)}{1}\frac{(x-5)}{5} + 0 \\ &= (x-3)(x-5) + (x-2)(x-5) \\ &= 2x^2 + 6x + 4, \end{aligned}$$

and so calculate the secret to be $s = f(0) = 4$.

Note that it is possible to replace $\mathbb{Z}_p$ by the finite field $GF(q)$ for any prime power $q > n$, and that a slight adjustment can make the scheme work for $q \geq n$. We analyse the efficiency of Shamir's threshold scheme in Section 3.3.2.

Now we have introduced threshold schemes and demonstrated a construction, we will formally define the relevant security notions. We begin by defining perfect threshold schemes, then progress to define computationally secure threshold schemes. Intuitively, a perfect threshold scheme requires sets of fewer than t players to learn (information theoretically) no information about s, whereas in a computationally secure scheme a negligible amount of information about s can be learnt.

2.3.1 Perfect threshold schemes

Here, we define perfect secrecy in a threshold scheme. Note that threshold schemes with perfect secrecy are call *perfect threshold schemes*. We begin by defining such schemes with information theoretic notation. We then consider two security games that allow us to provide an alternative, but equivalent, definition of perfect threshold schemes.

2.3.1.1 Information theoretic definition

In a threshold scheme, the notion of perfect secrecy is that a computationally unbounded adversary learns no information about the secret after seeing at most $t-1$ shares. Let $\boldsymbol{\mathcal{S}}$ denote the discrete random variable corresponding to the choice of secret, and let $\mathbf{A}$ denote the discrete random variable corresponding to the set of shares given to the players in the set $A \subseteq \mathcal{P}$.

Definition 2.3.4. *A (t,n)-threshold scheme is* perfect *if it is both*

- recoverable*, meaning that* $\mathrm{H}(\boldsymbol{\mathcal{S}} \mid \mathbf{A}) = 0$ *for any authorised set* $|A| \geq t$*, and*
- private*, so* $\mathrm{H}(\boldsymbol{\mathcal{S}} \mid \mathbf{B}) = \mathrm{H}(\boldsymbol{\mathcal{S}})$ *for any unauthorised set* $|A| < t$*.*

Shamir's threshold scheme, given in Construction 2.3.3, achieves perfect secrecy and is thus an example of a perfect threshold scheme.

We briefly note here that, in any perfect (t,n)-threshold scheme, distributing a secret of λ bits requires the generation of a minimum of $\lambda(t-1)$ bits of randomness [15]. As Shamir's scheme requires the generation of $t-1$ elements in the same field as the secret, Shamir's scheme achieves the minimal amount of randomness required to securely distribute the secret.

An alternative, but equivalent, definition of perfect threshold schemes relies on two security games between an adversary and a challenger. We discuss the security games, called *Priv* and *Rec*, which define the privacy and recoverability of a threshold scheme. Both games are attributed to [8] and are summarised in pseudo-code in Figure 2.2 as Games 2.3 and 2.4.

2.3.1.2 Privacy game

Intuitively, the privacy game, as in Game 2.3, models an adversary launching a chosen plaintext attack against the privacy of the threshold scheme, where the adversary learns

$t-1$ shares instead of the ciphertext.

The privacy game executes as follows. Given the parameters t and n, the challenger chooses a bit b at random. The adversary then chooses two secrets $s_0, s_1 \in \mathcal{S}$ and sends these to the challenger. The challenger checks $s_0, s_1 \in \mathcal{S}$; if not, the challenger halts and returns $\perp$ to the adversary. Otherwise, the challenger inputs s_b to the share algorithm of the threshold scheme, which outputs an n-vector $\boldsymbol{V}$ consisting of the n shares (where $\boldsymbol{V}[i] = v_i$). Nothing is returned to the adversary at this stage. The adversary can then make up to $t-1$ queries of the form $Corrupt(i)$ for $1 \leq i \leq n$ and receives the corresponding share $\boldsymbol{V}[i]$ in return. After the corrupt stage, the adversary must output a guess b' for b. The adversary wins if $b' = b$.

Let $\mathcal{A}$ be an adversary playing the *Priv* game against a secret sharing scheme Π. Call $\mathcal{A}$ a *privacy adversary* and the corresponding challenger a *privacy challenger.* Let $\Pr[Priv^{\mathcal{A}}]$ denote the probability $\mathcal{A}$ outputs the correct guess $b' = b$ during the finalise procedure. Define the *advantage* of $\mathcal{A}$ as

$$\mathbf{Adv}_{\Pi}^{Priv}(\mathcal{A}) = 2 \cdot \Pr[Priv^{\mathcal{A}}] - 1. \tag{2.3.1}$$

We define two notions of privacy, according to the advantage of a privacy adversary playing the *Priv* game.

Definition 2.3.5. *If the advantage of privacy adversary $\mathcal{A}$ against a secret sharing scheme Π, defined in* (2.3.1)*, is equal to zero, we say the scheme Π has* perfect privacy*.*

Definition 2.3.6. *If the advantage of privacy adversary $\mathcal{A}$ against a secret sharing scheme Π, defined in* (2.3.1)*, is negligible, we say the scheme Π has* computational privacy*.*

Notice that perfect privacy is a stronger security notion than computational privacy.

The privacy adversary can be considered to be either computationally unbounded or polynomial time. As the privacy adversary is able learn the shares during the corrupt procedure in an adaptive manner, $\mathcal{A}$ is an adaptive adversary.

The privacy game can be interpreted as a threshold scheme analogue of the indistinguishability game, defined in Game 2.1, played against a symmetric-key encryption scheme $\mathcal{E}$.

This is because a privacy adversary is unable to distinguish which secret was distributed, even after seeing $t-1$ shares.

2.3.1.3 Recoverability game

The recoverability game *Rec* for a (t,n)-threshold scheme is summarised in Game 2.4. Intuitively, the game models an adversary's ability to prevent the recovery of s by either deleting shares or submitting at most $n-t$ false shares to the recover algorithm of the threshold scheme. The game is initialised by letting the set T, which will denote the players the adversary corrupts, be the empty set. The adversary then chooses a secret s and submits this to the challenger, who inputs it to the share algorithm of the threshold scheme. As before, nothing is returned to the adversary at this stage. The adversary can then make up to $n-t$ queries of the form $Corrupt(i)$ for $1 \leq i \leq n$ and receives the corresponding share $\boldsymbol{V}[i]$ in return. Each player corrupted by the adversary is noted in the set T. During the finalise procedure, the adversary outputs a partially complete n-vector $\boldsymbol{V}_T$. This vector consists of $n-t$ shares learnt during the corrupt procedure, but each share has either been altered, so the learnt share has been replaced with an incorrect share, or deleted, meaning the learnt share is replaced with an empty string. This vector is then completed by the challenger who fills the remaining elements with valid shares from uncorrupted players; these shares are noted in the vector $\boldsymbol{V}_{\overline{T}}$. The complete vector $\boldsymbol{V}_T \cup \boldsymbol{V}_{\overline{T}}$ is then submitted to the recover algorithm. The adversary wins if the recovered secret s' is not equal to s, and hence the algorithm returns **true**. Otherwise, if $s' = s$, the adversary loses and hence **false** is returned.

Let $\mathcal{A}$ be an adversary playing the recoverability game *Rec* against Π. Call $\mathcal{A}$ a *recoverability adversary.* Let $\Pr[Rec^{\mathcal{A}}]$ denote the probability s is not recovered by the deterministic algorithm $Recover(\boldsymbol{V}_T \cup \boldsymbol{V}_{\overline{T}})$. Define the advantage of $\mathcal{A}$ as:

$$\mathbf{Adv}_{\Pi}^{Rec}(\mathcal{A}) = \Pr[Rec^{\mathcal{A}}]. \tag{2.3.2}$$

Definition 2.3.7. *If the advantage of privacy adversary $\mathcal{A}$ against a secret sharing scheme Π, defined in* (2.3.1)*, is equal to zero, we say the scheme Π has* perfect recoverability*.*

Definition 2.3.8. *If the advantage of recoverability adversary $\mathcal{A}$ against a secret sharing scheme Π, defined in* (2.3.2)*, is negligible, we say the scheme Π has* computational

```
Game 2.3: Priv
Procedure Initialise$(t, n)$
1   $b \xleftarrow{\$} \{0,1\}$;
2   $j = 1$;
3   return;
Procedure Deal$(s_0, s_1)$
1   if $s_0, s_1 \notin \mathcal{S}$ then
        return $\perp$;
2   else $\boldsymbol{V} \xleftarrow{\$} Share(s_b)$;
Procedure Corrupt$(i)$
1   if $j \leq t-1$ then
        return $\boldsymbol{V}[i]$;
        $j = j + 1$;
2   else return $\perp$;
Procedure Finalise$(b')$
1   if $b' = b$ then
        return true;
2   else return false;
```

```
Game 2.4: Rec
Procedure Initialise$(t, n)$
1   $T \leftarrow \emptyset$;
2   $j = 1$;
3   return;
Procedure Deal$(s)$
1   if $s \notin \mathcal{S}$ then
        return $\perp$;
2   else $\boldsymbol{V} \xleftarrow{\$} Share(s)$;
Procedure Corrupt$(i)$
1   if $j \leq n-t$ then
        return $\boldsymbol{V}[i]$;
        $j = j + 1$;
        $T \leftarrow T \cup \{i\}$;
2   else return $\perp$;
Procedure Finalise$(\boldsymbol{V_T})$
1   $s' \leftarrow Recover(\boldsymbol{V_T} \cup \boldsymbol{V_{\overline{T}}})$;
2   if $s' \neq s$ then
        return true;
3   else return false;
```

Figure 2.2: Privacy and recoverability games used to define perfect, computationally secure and robust, computationally secure threshold schemes.

recoverability.

Notice that perfect recoverability is a stronger security notion than computational recoverability. As in the privacy game, the recoverability adversary can be considered to be either computationally unbounded or polynomial time. Also as before, the recoverability adversary is adaptive as they are able to learn the shares during the corrupt procedure in an adaptive manner.

We will consider two recoverability adversaries with varying powers. The first adversary we consider is the weaker of the two and called a *restricted recoverability adversary*. Of the $n - t$ shares this adversary learns during the corrupt procedure, all must be erased, meaning the vector $\boldsymbol{V_T}$ consists of only empty shares. This adversary models an adversary who deletes or removes shares from the system.

The strongest recoverability adversary we consider, called an *unrestricted recoverability adversary* is allowed to replace at most t of the $n - t$ corrupted shares with incorrect shares, whilst the remaining shares must be replaced with empty strings [8]. That is, $\boldsymbol{V_T}$

must consist of at most t falsified shares, and the rest must be empty. This adversary models an adversary who actively corrupts shares, rather than just deleting them.

2.3.1.4 Game-based definition

Now we have introduced the relevant security games, we present the game-based definition of a perfect threshold scheme. This definition is equivalent to the information theoretic definition given in Definition 2.3.4.

Definition 2.3.9. *A* perfect (t, n)-threshold scheme *is a (t, n)-threshold scheme in which a computationally unbounded privacy adversary playing Game 2.3 has an advantage of zero and a computationally unbounded restricted recoverability adversary playing Game 2.4 also has an advantage of zero. In other words, a perfect (t, n)-threshold scheme is a (t, n)-threshold scheme with perfect privacy (against a computationally unbounded adversary) and perfect recoverability (against a restricted recoverability adversary).*

Intuitively, in a perfect (t, n)-threshold scheme, a privacy adversary who learns $t-1$ shares has no advantage over merely guessing which secret was distributed by the share algorithm.

2.3.1.5 Information rate

A measure of efficiency for a perfect threshold scheme is the information rate [95]. Let $\mathcal{V}_i$ denote the possible shares player P_i may receive, so P_i's share $v_i \in \mathcal{V}_i$. As the secret $s \in \mathcal{S}$, we can think of s as being represented by a bit-string of length $\log_2 |\mathcal{S}|$, by using a binary encoding, for example. Intuitively, player P_i receives $\log_2 |\mathcal{V}_i|$ bits of information as their share, but the information content of the secret is $\log_2 |\mathcal{S}|$ bits.

Definition 2.3.10. *In a perfect threshold scheme, the* information rate for player P_i, *denoted ρ_i, is the ratio*

$$\rho_i = \frac{\log_2 |\mathcal{S}|}{\log_2 |\mathcal{V}_i|}.$$

The information rate *of the scheme is defined as*

$$\rho = \min\{\rho_i : 1 \leq i \leq n\}.$$

A scheme is called ideal *if* $\rho = 1$.

Obviously, a high information rate is desirable and it is well known that the information rate for a perfect threshold scheme is upper bounded by one, so $\rho \leq 1$ [95]. This is why a perfect threshold scheme that achieves the optimal information rate is called ideal. The notion of an ideal threshold scheme was first introduced by Brickell [20]. Informally speaking, in a perfect threshold scheme, the size of each share is at least the size of the secret. An ideal scheme is one in which each player's share is the same size as the secret.

This bound on the information rate of a perfect threshold scheme can be restrictive. This may be especially true in settings where either the secret is large or the storage available to each player is small. In either case, it may be preferable to use a threshold scheme that achieves incremental, rather than perfect, secrecy. Ramp schemes offer such security and are introduced next.

2.3.2 Ramp schemes

One way to construct schemes with smaller shares, whilst still considering a computationally unbounded adversary, is to relax the requirement for the scheme to be perfect and allow the adversary to learn a measured amount of information about the secret. Ramp schemes are one example of how this can be done.

Definition 2.3.11. *Given a set $\mathcal{P} = \{P_1, \ldots, P_n\}$, a $(t_0, t_1; n)$-*ramp scheme *is a set of algorithms, Share and Recover, and two integers t_0, t_1 such that $0 \leq t_0 < t_1 \leq n$. The share algorithm distributes a secret s such that any set of at least t_1 players can pool their shares to recover the secret and a set of t_0 or fewer players reveals no information about the secret.*

Sets of players of size greater than t_0 but smaller than t_1 may learn partial information about the secret, hence ramp schemes are not necessarily perfect. To illustrate this, we observe that a $(t-1, t; n)$-ramp scheme is a perfect (t, n)-threshold scheme, and therefore a perfect threshold scheme is also a ramp scheme, but that a $(t-2, t; n)$-ramp scheme is not a perfect threshold scheme.

The average entropy of a player's share in a $(t_0, t_1; n)$-ramp scheme is known to be at least

$\log_2 |\mathcal{S}|/(t_1 - t_0)$ bits [54].

If we wish the security of the ramp scheme to be maximised with respect to the size of each share, then the information theoretic knowledge about the secret is likely to increase linearly with respect to the number of participants colluding in order to determine the secret. This motivates the following definition, which was first seen in [11].

Definition 2.3.12. *A $(t_0, t_1; n)$-ramp scheme is said to be* linear *if, for any set of players $A \subseteq \mathcal{P}$ such that $|A| = r$, where $t_0 \leq r \leq t_1$,*

$$H(\boldsymbol{\mathcal{S}} \mid \mathbf{A}) = \frac{t_1 - r}{t_1 - t_0} H(\boldsymbol{\mathcal{S}}).$$

So, in a linear $(t_0, t_1; n)$-ramp scheme, any set of at least t_1 players can recover the secret, whereas t_0 or fewer players learn no information about the secret. For every player contributing after the initial t_0 players have contributed their shares, a fixed amount of information is learnt about s. This continues in a linear fashion until t_1 players have contributed and s is learnt completely. In fact, after t_0 shares are pooled, every further share reveals $\log_2 |\mathcal{S}|/(t_1 - t_0)$ bits of information about s.

The following construction is based on Shamir's threshold scheme, presented in Construction 2.3.3, and is due to McEliece and Sarwate [73].

Construction 2.3.13. *Assume there is a (t, n)-threshold scheme as in Construction 2.3.3. In order to construct a linear $(t_0, t_1; n-m+1)$-ramp scheme, where $m = t_1 - t_0$ and $t_! = t$, let $\mathcal{P}$ be a set of $n-m+1$ players, and let $p > n$ be a prime. The secret for this scheme is an element $\mathbf{s} = (s_0, s_1, \ldots, s_{m-1}) \in \mathbb{Z}_p^m$. It is shared by selecting a polynomial f uniformly from the set of all polynomials in $\mathbb{Z}_p[m]$ that satisfy $f(0) = s_0, f(1) = s_1, \ldots, f(m-1) = s_{m-1}$. We identify each player with a unique nonzero element of $\mathbb{Z}_p \setminus \{0, 1, \ldots, m-1\}$, and to player P_i we assign the share $v_i = f(i)$.*

As before, any set of t_1 players can perform interpolation to recover f, which enables them to recover the entire secret. In addition, it can be shown that any set of t_0 or fewer players learn no information about the secret.

We present the following example, which illustrates Construction 2.3.13 by forming a linear $(t_0, t_1; n - (t_1 - t_0) + 1)$-ramp scheme from a perfect (t_1, n)-threshold scheme.

Example 2.3.2. Consider Example 2.3.1, where $t = 3$ and $n = 5$ with all computations computed in $\mathbb{Z}_7$. For the ramp scheme, let $t_1 = t = 3$ and choose t_0 to be strictly less than t_1, say $t_0 = 1$. We will construct a linear $(t_0, t_1; n - m + 1)$-ramp scheme using Construction 2.3.13. In this example, this is a linear $(1, 3; 4)$-ramp scheme.

Let $\mathcal{P} = \{P_2, P_3, P_4, P_5\}$ denote the four players and let the secret be $s = (s_0, s_1) = (4, 5) \in \mathbb{Z}_7^2$. Then, if we let $f = 2x^2 + 6x + 4 \in \mathbb{Z}_7[x]$ (as in Example 2.3.1) be the randomly generated polynomial that satisfies $f(0) = s_0 = 4$ and $f(1) = s_1 = 5$, we can give each player P_i, for $2 \leq i \leq 5$, the share $v_i = f(i)$. As in Example 2.3.1, any three players can construct s via polynomial interpolation. It can additionally be shown that any individual player learns no information about the secret, whilst two players learn one element of the secret.

Note that Example 2.3.2 is very similar to Example 2.3.1, but player P_1's share becomes part of the secret (so the secret increases from one to two elements) and the number of players in the scheme decreased from five to four (as P_1 is excluded from $\mathcal{P}$).

As with perfect threshold schemes, it is possible to calculate the information rate of a linear ramp scheme. This is done in [54], where they explain that in a linear (t_0, t_1, n)-ramp scheme, for any set A of r players,

$$H(\mathbf{A}) \geq \frac{r \cdot H(\boldsymbol{\mathcal{S}})}{(t_1 - t_0)}.$$

That is, for any player P_i in a linear ramp scheme, the size of their share v_i is at least $H(\boldsymbol{\mathcal{S}})/(t_1 - t_0)$ bits, meaning the information rate of the scheme is upper bounded by $\rho \leq t_1 - t_0$. If this lower bound is obtained for every player's share, the ramp scheme is called *optimal.* Farras et al. [35] provides a further discussion on the bounds and construction of ramp schemes.

2.3.3 Computationally secure threshold schemes

An alternative way to construct a threshold scheme with smaller shares than in a perfect threshold scheme is to consider a weaker adversary. Rather than a computationally unbounded adversary, we can consider a polynomial time adversary.

Algorithm 2.5: $Share^{HK0}(M)$.

```
1 $k \xleftarrow{\$} KeyGen(\{0,1\}^\lambda)$;
2 $C \xleftarrow{\$} Enc_k(M)$;
3 $\boldsymbol{K} \xleftarrow{\$} Share^{PTS}(k)$;
4 $\boldsymbol{C} \xleftarrow{\$} Share^{IDA}(C)$;
5 for $i \leftarrow 1$ to $n$ do
      $\boldsymbol{V}[i] \leftarrow (\boldsymbol{K}[i] || \boldsymbol{C}[i])$;
6 return $\boldsymbol{V}$.
```

Algorithm 2.6: $Recover^{HK0}(\boldsymbol{V})$.

```
1 for $i \leftarrow 1$ to $n$ do
      $\boldsymbol{K}[i]\boldsymbol{C}[i] \leftarrow \boldsymbol{V}[i]$;
2 $k \leftarrow Recover^{PTS}(\boldsymbol{K})$;
3 $C \leftarrow Recover^{IDA}(\boldsymbol{C})$;
4 $M \leftarrow Dec_k(C)$;
5 return $M$.
```

Figure 2.3: The share and recover algorithms defining HK0.

Computationally secure threshold schemes were first mentioned by Karnin *et al.* [58], with the first construction of a scheme being made by Krawczyk [60]. We define computationally secure schemes as defined in [8]. As with the game-based perfect threshold scheme definition, given in Definition 2.3.9, this definition relies on the privacy and recoverability games given in Figure 2.2. Computationally secure threshold schemes consider a polynomial time, adaptive adversary.

Definition 2.3.14. *A* computationally secure (t,n)-threshold scheme *is a (t,n)-threshold scheme in which a polynomial time privacy adversary has a negligible advantage and a polynomial time restricted recoverability adversary has an advantage of zero. In other words, a computationally secure (t,n)-threshold scheme is a (t,n)-threshold scheme with computational privacy (against a polynomial time adversary) and perfect recoverability (against a restricted recoverability adversary).*

In a computationally secure (t,n)-threshold scheme, the advantage of the privacy adversary is negligible. Because of this, perfect security is a stronger notion of security than computational security. However, computationally secure schemes are sufficient for most applications [60].

In 1994 Krawczyk proposed the first computationally secure threshold scheme [58]. We call this the HK0 scheme with share and recover algorithms denoted by $Share^{HK0}$ and $Recover^{HK0}$, which are defined in Figure 2.3. Krawczyk's motivation in proposing HK0 was to achieve shares smaller than were possible in perfect threshold schemes.

The HK0 share algorithm takes as input the plaintext M to be distributed, generates a λ bit key k and encrypts M under k to get the ciphertext C. The key k is shared via a perfect (t, n)-threshold scheme (denoted by $Share^{PTS}$), which returns an n vector $\boldsymbol{K}$. The ciphertext C is shared via some (t, n)-information dispersal algorithm (denoted by $Share^{IDA}$ and discussed in more detail in Section 2.4), which returns an n-vector $\boldsymbol{C}$. Each player P_i is given a share $\boldsymbol{V}[i]$ that consists of the ciphertext and the key, so $\boldsymbol{V}[i] = \boldsymbol{K}[i]\boldsymbol{C}[i]$.

The recover algorithm takes as input an n-vector $\boldsymbol{V}$, where the i^{th} element of this vector $\boldsymbol{V}[i]$ is either a string $\boldsymbol{V}[i] \in \{0, 1\}^*$, or the value $\Diamond$. In the first case, $\boldsymbol{V}[i]$ is the purported share of player P_i, while in the second case, $\boldsymbol{V}[i] = \Diamond$, the share has been marked as missing. Each share $\boldsymbol{V}[i]$ is then parsed into the ciphertext and key shares, $\boldsymbol{C}[i]$ and $\boldsymbol{K}[i]$, and the key k and ciphertext C are then recovered via the recovery algorithm of the perfect threshold scheme and the recovery procedure of the information dispersal algorithm, respectively. Finally, C is decrypted under k to return the plaintext, M.

2.3.4 Robust threshold schemes

In both perfect and computationally secure threshold schemes, we have considered the restricted recoverability adversary, who is limited to only deleting, and not altering, shares. With an adversary restricted in this way, it is guaranteed that, as long as t shares are submitted to the recover algorithm, s will be correctly recovered.

We now consider the unrestricted recoverability adversary, who is able to query $n - t$ shares during the corrupt procedure and can submit a vector $\boldsymbol{V_T}$ consisting of at most t false shares and the rest empty. An adversary with these powers was first considered by Tompa and Woll in [98], who demonstrated that such an adversary has a number of undesirable capabilities. For example, they may be able to prevent the correct secret from being recovered, the players may not know an incorrect secret has been recovered, and the adversary may be able to recover the correct secret for themselves.

Definition 2.3.15. *A* robust, computationally secure (t, n)-threshold scheme *is a (t, n)-threshold scheme in which a polynomial time privacy adversary playing Game 2.3, and a polynomial time unrestricted recoverability adversary playing Game 2.4, both have a negligible advantage at winning their respective games. In other words, a robust, computationally secure (t, n)-threshold scheme is a (t, n)-threshold scheme with computational privacy (against a polynomial time adversary) and computational recoverability (against* *an unrestricted recoverability adversary).*

Intuitively, a robust scheme ensures the recovery of the correct secret in the setting where the unrestricted recoverability adversary is allowed to both corrupt and/or delete a bounded number of shares.

2.4 Information dispersal algorithms

In this section, we introduce information dispersal algorithms, which are used regularly throughout this thesis and are closely related to a number of other constructions, such as threshold schemes, ramp schemes and error correcting codes. We will see that information dispersal algorithms are not necessarily secure, but that different constructions can guarantee some security. For this reason, we give a number of examples and delve into more detail than we have done previously.

2.4.1 General information dispersal algorithms

Information dispersal algorithms were first introduced by Rabin in 1989 [80].

Definition 2.4.1. *Let $t, n \in \mathbb{N}$ and let $1 \leq t \leq n$. A (t, n)-*information dispersal algorithm *(denoted as (t, n)-IDA) consists of a message space $\mathcal{M}$ and two algorithms $Share^{IDA}$ and $Recover^{IDA}$.*

- *$Share^{IDA}$ takes as input a message $M \in \mathcal{M}$ and outputs an n-vector $\boldsymbol{V}$.*
- *$Recover^{IDA}$ takes as input elements of the vector $\boldsymbol{V}$. If at least t elements are submitted correctly to $Recover^{IDA}$, the algorithm will output the original message M.*

Intuitively, a (t, n)-IDA transforms some data M into n shares, such that any t of these shares can recover the original data M.

The concept of a (t, n)-IDA is similar to that of a threshold scheme from Definition 2.3.2. They are similar in one way because they both distribute data and require the data to be recoverable from t shares. However, there is a key difference: the IDA does not consider the privacy of the scheme. In an IDA, there is no limit to the amount of information a set of fewer than t players can learn about the data, whereas in a threshold scheme fewer than t players should learn nothing (or a negligible amount) about the data.

However, by not considering the privacy of the data, IDAs are able to achieve smaller share sizes than threshold schemes. Recall how in a perfect threshold scheme, the size of each share must be at least the size of the secret. In contrast, IDAs are able to achieve much smaller shares. In fact, IDAs are able to generate shares that are about one t^{th} of the size of the data. IDAs such as these are very similar to linear ramp schemes, as we will see in Section 2.4.2.2. Assuming the input data has maximal entropy, one t^{th} of the size of the data is the theoretical lower bound on the size of each share whilst guaranteeing recoverability of the data for any set of t players [50]. We call an IDA achieving this bound *optimal*.

We measure how 'close' to optimal an IDA is by calculating the *information rate*. The information rate of an IDA is similar to the information rate of a perfect threshold scheme from Definition 2.3.10, and is computed in a similar manner: by calculating the minimum ratio between the size of shares given to players and the size of the data being shared. Using the theoretical share size given in [50], the upper bound on the information rate of an IDA is t. This is the information rate achieved by optimal IDAs. IDAs with an information rate lower than t are *non-optimal*.

One example of an IDA is a perfect threshold scheme, as it is able to guarantee recovery of the data from any t players. However, threshold schemes are non-optimal IDAs as their share sizes are much larger than the smallest possible share size for an IDA. The converse, however, is not true: not all IDAs are perfect threshold schemes, as they do not consider the security of the data.

Another example of an IDA is *replication*, in which data M is distributed amongst n players by giving each player an exact copy of M. Generally, for any $1 \leq t \leq n$, any t players can trivially recover M and thus replication is a valid (t, n)-IDA. Obviously, as any $t-1$ players can also recover M, this is not a perfect (t, n)-threshold scheme. Replication is a second example of an IDA in which each player's share is the same size as the data being distributed. Thus, replication achieves share sizes equal to those in an ideal, perfect threshold scheme; in both cases, these are much larger than achieved by optimal IDAs.

Despite being non-optimal, replication requires no (or minimal) computation in order to distribute (or recover) the data. Replication is an IDA that prioritises computational complexity above memory. However, we do note that non-optimal IDAs cannot guarantee successful decoding. To illustrate this, consider two players recovering the data from a replication IDA whose shares disagree in one symbol. There is no way to tell which player's share is correct, if either. Other non-optimal IDAs that achieve lower complexities by compromising memory (although not as completely as replication does), are found in [67] and [3].

Error correcting codes is a field of research that can provide a number of constructions for optimal (t, n)-IDAs that also provide some level of security. Influenced by this, we introduce error correcting codes next and explore their relation to IDAs.

2.4.2 Error correcting codes and IDAs

2.4.2.1 An introduction to error correcting codes

An error correcting code is a method of *encoding* data with some redundant information to ensure the original data can be recovered, or *decoded*, even if a number of errors occur during either data transmission or storage [70].

Definition 2.4.2. *An* error correcting code *(ECC) E of* length n *over a finite alphabet F is a subset of F^n. The elements of E are called* codewords*. The* size *of E is the number of codewords in the code, $|E| = m$. Define the* Hamming distance *between two codewords u, v in E, denoted $d(u, v)$, to be the number of positions in which u and v are different. The* minimum distance *of the code E is the minimum Hamming distance between any two distinct codewords in E, and is denoted by d.*

ECCs are able to detect and correct a number of errors. A code E is said to be e_1-*error detecting* if, whenever a codeword $u \in E$ is sent and between one and e_1 errors occur, the received word u' is not a codeword (meaning $u' \notin E$), and so the receiver will know an error has occurred in the channel. A code E is e_2-*error correcting* if, when a word $w \notin E$ is received, the Hamming distance between a codeword $u \in E$ and w is at most e_2, then

w is decoded to u using nearest neighbour decoding; that is, of all the codewords in the code E, the hamming distance between u and w is minimal.

Let E be a code of length n. Say E is *linear* if, for all codewords $u, w \in E$, we have $u + w \in E$, where addition is computed modulo q, where $|F| = q$. Intuitively, a code is linear if all linear combinations of the codewords are also codewords.

Definition 2.4.3. *If $u_1, \ldots, u_t$ is a basis for a linear code E, then we say E has* dimension *t. Assume E is a code over alphabet F with dimension t such that each codeword in E is of length n and d is the minimum distance of E. Call such a code an* $[n, t, d]$-code. *In any $[n, t, d]$-code, there are q^t possible codewords in E, where $|F| = q$.*

We briefly introduce the notion of a *dual code.*

Definition 2.4.4. *For a linear code E of length n over an alphabet F of size q (i.e. $E \subset F^n$), the dual code of E is the linear code $E^* = \{u \in F^n \mid \langle u, w \rangle = 0 \; \forall w \in E\}$, where $\langle u, w \rangle = \sum_{i=1}^{n} u_i w_i$ is a scalar product. The* dual distance *of E is the minimum distance of the code E^* and is denoted by d^*.*

One important type of ECC is a maximum distance separable code [70]. The definition of such codes relies on the *Singleton bound* [93], which says that, for any code E with length n, minimum distance d and dimension t, $d < n - t + 1$.

Definition 2.4.5. *A* maximum distance separable *(MDS) code is an $[n, t, d]$-code that meets the Singleton bound, so $d = n - t + 1$. We call such a code an $[n, t, d]$-MDS code.*

MDS codes have the maximum possible Hamming distance between codewords and each codeword can be separated into message symbols and check symbols. One important example of an MDS code is a Reed Solomon (RS) code [84].

Finally, we introduce the notion of an *erasure code.* An erasure code is a code that is able to recover data if it knows which symbols are missing or corrupt; that is, if it knows where the errors are, it can correct them, but it may not necessarily be able to detect them. An erasure code can tolerate up to $n - t$ erasures, where a known corrupted symbol is treated as an erasure. In such a code, recovery of a codeword is possible from any t of the n symbols. Because of this property, we refer to such a code as a (t, n)-ECC. For example, an RS code can be used as an erasure code and can thus be interpreted as a (t, n)-ECC.

2.4.2.2 Relating ECCs to IDAs

We can relate ECCs to IDAs. Conceptually, a (t, n)-IDA can be treated as a (t, n)-ECC, or, more specifically, an $[n, t, n - t + 1]$-MDS code. When t shares are input to the recover algorithm of the IDA, the scenario is equivalent to erasing $n - t$ symbols in a length n codeword, where the erased symbols are analogous to non-received shares. Thus, MDS codes, such as RS codes, can be used as IDAs. This is well known and there are many algorithms using these techniques, including Rabin's initial proposal for an IDA in [80]. These IDAs are optimal with respect to the size of each share and, if conventional approaches at implementation are used, require $\mathcal{O}(nt)$ operations for dispersal and $\mathcal{O}(t^2)$ operations for recovery.

We can also relate ECCs to threshold schemes; their relationship has been long studied [73, 71]. It is known that a length n code with minimum distance d and dual distance d^* gives a $(t_0, t_1; n-1)$-ramp scheme with $t_0 = d^* - 2$ and $t_1 = n - d + 1$ (see, for example, [77] for details). We, however, will use a result from [24] that equates a $[n, t, n-t+1]$-MDS code with a linear $(0, t; n)$-ramp scheme, as in Definition 2.3.11, assuming maximal entropy on the input data.

From now on, if we require a (t, n)-IDA with security properties equivalent to that of a linear $(0, t; n)$-ramp scheme, we use a $[n, t, n - t + 1]$-MDS code with share and recover algorithms denoted by $Share^{ECC}$ and $Recover^{ECC}$. If there are no security requirements on the IDA, we use the more general $Share^{IDA}$ and $Recover^{IDA}$ to denote the share and recover algorithms. In this case, either an ECC or a lower complexity algorithm, such as replication, will suffice with respect to security concerns. The choice of exactly which IDA construction to use, however, will depend on the application.

2.4.3 Secure IDA constructions using ECCs

Here, we introduce a construction of a secure IDA that was originally introduced by Rabin. This IDA will be used throughout the thesis to build threshold schemes. After introducing the construction, we consider one method for improving the efficiency of this scheme and mention some efficient IDAs proposed in the literature.

2.4.3.1 Rabin's IDA

The construction of Rabin's IDA, and many other MDS encodings, relies on a matrix-vector product. The following describes an IDA, attributed to Rabin [80]. This IDA is equivalent to a linear $(0, t; n)$-ramp scheme and is optimal with respect to the memory required for each share.

Consider an $n \times t$ public matrix G. Let the elements of G be in some finite field $\mathbb{F}$ and let G be such that any t of the n rows are linearly independent. Consider the message to be dispersed as a t vector, $\boldsymbol{M}$, with each element of the vector also in the finite field $\mathbb{F}$. Now, to distribute the data $\boldsymbol{M}$, compute the matrix-vector product $G \cdot \boldsymbol{M}$. This will result in an n vector $\boldsymbol{V}$, where each player P_i is allocated the element $\boldsymbol{V}[i]$ as their share.

To recover the data $\boldsymbol{M}$, a set of t players must collaborate and pool their shares to construct a t vector $\boldsymbol{V}'$. A $t \times t$ matrix G' is then constructed from the t rows of G that correspond to the t players collaborating for the reconstruction. The matrix-vector product $(G')^{-1} \cdot \boldsymbol{V}'$ is then computed, which will recover the message vector $\boldsymbol{M}$. Note that the inverse of G' will always exist because G was chosen such that any t rows are linearly independent.

We illustrate this process in Example 2.4.1.

Example 2.4.1. Let $n = 5$ and $t = 3$ and let us work in the finite field of seven elements, $\mathbb{Z}_7$. Consider the message to be distributed as a 3-vector, where $\boldsymbol{M} = (5, 3, 2)^T$. Let the publicly known 5×3 matrix G be

$$G = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 4 & 1 \\ 3 & 2 & 6 \\ 4 & 2 & 1 \\ 5 & 4 & 6 \end{pmatrix},$$

which is such that any t rows are linearly independent over the finite field $\mathbb{Z}_7$. To encode $\boldsymbol{M}$, compute the matrix-vector product $G \cdot \boldsymbol{M}$, where all operations are computed modulo 7:

$$G \cdot \boldsymbol{M} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 4 & 1 \\ 3 & 2 & 6 \\ 4 & 2 & 1 \\ 5 & 4 & 6 \end{pmatrix} \begin{pmatrix} 5 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \\ 5 \\ 0 \\ 0 \end{pmatrix} = \boldsymbol{V}.$$

Allocate $\boldsymbol{V}[i]$ to player P_i as their share, for $1 \leq i \leq 5$.

Assume a set of $t = 3$ players wishes to reconstruct the data. Say these three players are P_2, P_3 and P_5. They construct a vector from their shares, $\boldsymbol{V}' = \{3, 5, 0\}^T$ and construct a matrix G' consisting of the rows of G corresponding to their shares. So, here, G' consists of the second, third and fifth rows of G. Then, compute $(G')^{-1}\boldsymbol{V}'$ as follows:

$$(G')^{-1} \cdot \boldsymbol{V}' = \begin{pmatrix} 2 & 4 & 1 \\ 3 & 2 & 6 \\ 5 & 4 & 6 \end{pmatrix}^{-1} \begin{pmatrix} 3 \\ 5 \\ 0 \end{pmatrix} = \begin{pmatrix} 6 & 3 & 3 \\ 1 & 0 & 1 \\ 6 & 1 & 4 \end{pmatrix} \begin{pmatrix} 3 \\ 5 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \\ 2 \end{pmatrix}.$$

This results in the message $\boldsymbol{M}$ being output.

Although we have presented Example 2.4.1 as an IDA, it can be interpreted as an MDS code. The matrix G is called the *generator matrix* of the code; all possible output vectors $\boldsymbol{V}$ are the codewords of the code which each have length $n = 5$. Any three rows of the matrix G can act as a basis for the codeword, which has dimension $t = 3$. The minimum Hamming distance between any two codewords is $n - t + 1 = 3$. Shares that are not submitted to the recover algorithm are treated as erasures and the code is able to tolerate up to $n - t$ erasures. Hence, this is a $[5, 3, 3]$-MDS erasure code.

In general, the complexity of the share and recover algorithms, illustrated by Example 2.4.1, is dependent on the overhead of computing matrix-vector products. In particular, the matrix-vector multiplication required for dispersal requires multiplying an $n \times t$ matrix with a t-vector, which would require $\mathcal{O}(nt)$ operations. Recovery requires the multiplication of a $t \times t$ matrix with a t-vector, which requires $\mathcal{O}(t^2)$ operations.

Prior to briefly exploring efficient IDAs, we present the following result, which will be used in Chapter 3.

Theorem 2.4.6. *Consider an IDA based on a matrix-vector product construction with algorithms* $Share^{ECC}$ *and* $Recover^{ECC}$. *Assume there are* $\ell - 1$ t*-vectors over a finite field* $\mathbb{F}$, *denoted* $x_1, \ldots, x_{\ell-1} \in \mathbb{F}^t$, *where the* j^{th} *element of* x_i *is denoted as* $x_i[j]$. *Consider a further vector,* $x_\ell \in \mathbb{F}^t$, *such that* $x_\ell = x_1 + \cdots + x_{\ell-1}$. *Assume each vector* x_i, *for* $1 \leq i \leq \ell$, *is dispersed via* $Share^{ECC}$, *resulting in the* n*-vector* $X_i \leftarrow Share^{ECC}(x_i)$. *Let the* k^{th} *element of* X_i, *for* $1 \leq k \leq n$, *be denoted as* $X_i[k]$. *Then, for all* i,

$$X_1[i] + X_2[i] + \cdots + X_{\ell-1}[i] = X_\ell[i].$$

Proof. If the vector x_ℓ is distributed via the IDA, then

$$x_\ell G = X_\ell.$$

Now, as $x_\ell = x_1 + \cdots + x_{\ell-1}$, we can expand this to give

$$\begin{aligned} x_\ell G &= (x_1 + \cdots + x_{\ell-1})G \\ &= x_1 G + \cdots + x_{\ell-1} G \\ &= X_1 + \cdots + X_{\ell-1} \\ &= X_\ell, \end{aligned}$$

and hence

$$X_1[i] + X_2[i] + \cdots + X_{\ell-1}[i] = X_\ell[i],$$

as required. □

2.4.3.2 Efficient, secure IDA constructions

Since Rabin proposed the IDA presented in Section 2.4.3.1, there have been a number of proposals for more efficient schemes. One simple refinement of the scheme, suggested in [8], is to let the first t rows of G be the identity matrix, I_t. This means that the first t elements of the codeword $\boldsymbol{V}$ need not be encoded. We briefly illustrate this idea in Example 2.4.2 by continuing from Example 2.4.1.

Example 2.4.2. Replace the first $t = 3$ rows of G with the 3×3 identity matrix I_t and compute the matrix-vector product $G \cdot \boldsymbol{M}$:

$$G \cdot \boldsymbol{M} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 4 & 2 & 1 \\ 5 & 4 & 6 \end{pmatrix} \begin{pmatrix} 5 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 5 \\ 3 \\ 2 \\ 0 \\ 0 \end{pmatrix} = \boldsymbol{V}.$$

The first three elements of $\boldsymbol{V}$ required no computation. Recovery is as before, but we note that if players P_1, P_2 and P_3 pooled their shares no computation would be required as $G' = I_t = (G')^{-1}$.

We call a code which contains the original word in the codeword, as in Example 2.4.2, *systematic*. Perhaps it is more intuitive to view this systematic IDA as a linear $(0, t; n)$-ramp scheme than it was to view Rabin's original IDA as one. Clearly, each player learns some information (specifically, one t^{th} of the information) about the original data. As more players pool their shares, more information about the data is contributed, until t shares completely recover the data.

This systematic IDA is slightly more efficient than the original IDA proposed by Rabin because the first t elements of the codeword require no encoding. Instead, distribution can be interpreted as treating the final $n-t$ rows of G as a matrix and multiplying this by the vector in order to compute the check sums, which are then appended to the word. This dispersal requires multiplying an $(n-t) \times t$ matrix with a t-vector, which is equivalent to $\mathcal{O}(t(n-t)) \approx \mathcal{O}(nt)$ operations. Recovery is as before, requiring $\mathcal{O}(t^2)$ operations, unless the elements submitted to the recover algorithm are the first t elements of the codeword that were not encoded. If this is the case, no operations would be required as the computation required to recover the message would be $(G')^{-1} \cdot \boldsymbol{M} = (I_t)^{-1} \cdot \boldsymbol{M} = I_t \cdot \boldsymbol{M} = \boldsymbol{M}$, which is trivial to compute.

There are a number of other IDA constructions that deviate from the matrix-vector construction and, in doing so, achieve optimal IDAs with a lower complexity than Rabin's construction. These include: [79], which achieves a complexity of $\mathcal{O}(n \log n)$ for encoding and $\mathcal{O}(t(n-t+\log t))$ for decoding; [94], which requires $\mathcal{O}(n \log n)$ for both encoding and decoding, and [66], which achieves $\mathcal{O}(n \log t)$ for encoding and $\mathcal{O}(t \log^2 t)$ for decoding. Further exploring efficient IDAs is outside the scope of this thesis, but interested readers are directed to [66] for a discussion on efficient IDAs.

Chapter 3

An efficient, perfect threshold scheme

Contents

The work in this chapter is joint with Liqun Chen and Keith Martin and is published in [25].

3.1 Introduction

In Definition 2.3.4, we gave an information theoretic based definition of a perfect (t, n)-threshold scheme and in Definition 2.3.10 we described what it meant for such a scheme to be ideal. In this chapter, we propose an ideal, perfect threshold scheme that is computationally efficient.

3.1.1 Efficiency of threshold schemes

We begin by qualifying what is meant by an efficient threshold scheme. In order to do this, we consider the limitations of constrained devices, presented in Section 2.1.7. Firstly, constrained devices have a limited amount of power, so an efficient scheme should minimise the number of computations required by the device. Secondly, the constrained devices may have minimal memory available for storage. Therefore, an efficient scheme should minimise the share sizes, meaning it should be ideal, or at least close to ideal. Finally, we note that generating true random bits is incredibly difficult to do, even for computationally able devices. So, an efficient threshold scheme should minimise the number of random bits required.

It may be the case that the share and recover algorithms of the threshold scheme are run by different devices. For example, a dealer may be responsible for generating the secret, running the share algorithm and securely sending the shares to the n players. When the time comes to recover the secret, the players themselves may collaborate and run the recover algorithm, or may outsource this computation to a trusted external player. Each case may require different efficiency demands. For example, if the dealer is a constrained device which must generate and distribute multiple distinct secrets, reducing the number of computations in the share algorithm may be a priority. If, however, the dealer is a computationally able device and the players collaborating to recover the secret are constrained, reducing the number of computations in the recover algorithm may be a priority.

For this reason, we will consider the computational efficiency of the share and recover algorithms separately, as different schemes may be more appropriate, dependent on what is being prioritised in the application.

Note that, in this chapter, we only consider perfect, ideal threshold schemes and analysing the efficiency of such schemes. There are many ways the efficiency of schemes could be improved on if we relax these requirements; this is out of the scope for this thesis, but we give some examples of how to relax these requirements here. Firstly, we could firstly relax the requirement of perfect security to non-perfect security, such as is done in ramp schemes or computationally secure schemes, and consider the resulting efficiency gains. Non-perfect schemes and improved information rate are considered in [35], whilst computationally secure schemes are considered in Chapter 4. Secondly, we could relax the requirement for the schemes to be ideal and explore if allowing for larger share sizes can improve the efficiency. Finally, we could relax the requirement for the schemes to be threshold and instead consider *near-threshold* schemes and explore their efficiencies, as is done in [29].

3.1.2 Efficiency of Shamir's threshold scheme

Shamir's (t, n)-threshold scheme, presented in Construction 2.3.3, is an ideal, perfect threshold scheme that requires the generation of minimal randomness and is based on polynomial interpolation. The scheme is elegant but, despite improvements in implementation, has a fairly heavy computational cost; this is particularly true for the recover algorithm, due to its reliance on polynomial interpolation. In applications where constrained devices are responsible for running either the share or recover algorithm, any improvements to the computational cost are of clear benefit.

In particular, we see in Section 3.3.2 that the complexity of Shamir's share algorithm when a λ bit secret is distributed is $\mathcal{O}(\lambda^2 nt)$, whilst the complexity of Shamir's recover algorithm is $\mathcal{O}(\lambda^2 t \log^2 t)$. Given the constrained devices that may be required to run these algorithms, there has been effort in the literature to construct more efficient schemes.

3.1.3 Alternative efficient threshold schemes

There have been efforts in the literature to create threshold schemes that achieve the properties of Shamir's threshold scheme (that is, a perfect, ideal threshold scheme requiring minimal randomness) but that can be implemented efficiently, possibly using only XOR operations. These schemes claim to be lighter than Shamir's and are therefore more applicable to constrained devices.

Kurihara *et al.* presented the first XOR-based, ideal (t, n)-threshold scheme in [62] as a generalisation of the work in [61], which considered $(3, n)$-threshold schemes. In their work, Kurihara *et al.* construct shares by XORing pieces of the secret with multiple random numbers and distributing these amongst the players. Recovery is possible by multiplying a vector consisting of the shares by a matrix generated via Gaussian elimination, which is computationally heavy. Kurihara *et al.* analysed the efficiency of their scheme and compared it to Shamir's scheme.

Since the work by Kurihara *et al.*, few schemes achieving the same properties have been proposed. Lv *et al.* proposed one such scheme in [69] and a multi-secret analogue in [68], but Wang and Desmedt [99] show that in [69] the size of the shares are smaller than the size of the secret and therefore cannot be correct. Their criticism is, however, only true in the multi-secret sharing case. Nonetheless, the scheme by Lv *et al.* does have a number of flaws, such as incompatible matrix-vector multiplications and an incorrect analysis of the number of randomly generated values, and will therefore not be further considered.

Wang and Desmedt proposed their own (t, n)-threshold scheme that is equivalent to an ECC [99]. They proved their scheme to be secure and claimed it could be implemented using only XOR and cyclic shift operations. However, they do not provide an exact share algorithm, an efficiency analysis or compare their scheme to the current literature.

3.1.4 The presented scheme

In this chapter, we present a perfect (t, n)-threshold scheme that is ideal and requires minimal randomness. The scheme is efficient to implement, dependent on the chosen primitives. The scheme presented is a modified version of a perfect threshold scheme that forms an embedded component of a computationally secure threshold scheme presented by Camble, Chen, Henry and Watkins from Hewlett Packard (HP) in [21]. From their computationally secure threshold scheme, we extract a perfect threshold scheme and improve it by reducing the number of randomly generated bits required, thereby achieving smaller share sizes for each player. Throughout, we refer to the perfect threshold scheme directly extracted from the computationally secure threshold scheme as the original HP scheme, and to our improved version of their scheme as the modified HP scheme.

After presenting the modified HP scheme, we analyse the components of the scheme and highlight the security assumptions on each element. As the original HP scheme was presented without analysis, we present a proof of security for the modified HP scheme, which can be easily adapted to prove the security of the original HP scheme, and provide an efficiency analysis. We then compare the modified HP scheme with other threshold schemes, including Shamir's threshold scheme [91], a scheme presented by Kurihara *et al.* [62], and one by Wang and Desmedt [99].

3.2 The modified HP scheme

In this section, we present the modified HP scheme, which is a perfect (t, n)-threshold scheme. The scheme is defined in the Galois field $\mathbb{F} = GF(2^\ell)$, as this is most likely the chosen field for implementation. Therefore, the secret and all shares are binary strings. The scheme could, however, be generalised to any Galois field $GF(q^\ell)$.

3.2.1 Definition of the scheme

The modified HP scheme constitutes a share and recover algorithm, both presented in Figure 3.1.

For the scheme, we assume there exists an IDA with equivalent security properties to a linear $(0, t; n)$-ramp scheme. As discussed in Section 2.4.2, such an IDA can be conceptually treated as an ECC, and thus the share and recover algorithms of the IDA will be denoted by $Share^{ECC}$ and $Recover^{ECC}$. Throughout this chapter, when we refer to an IDA, we mean one with this security property. One possible candidate for the IDA is the systematic version of Rabin's IDA, introduced in Section 2.4.3.2.

Algorithm 3.1: *Share*(s).

```
1 for $i \leftarrow 1$ to $t-1$ do
      $r_i \xleftarrow{\$} \{0,1\}^\lambda$;
2 $s' = s \oplus r_1 \oplus \cdots \oplus r_{t-1}$;
3 $\boldsymbol{S} \leftarrow Share^{ECC}(s')$;
4 for $i \leftarrow 1$ to $t-1$ do
      $\boldsymbol{R}_i \leftarrow Share^{ECC}(r_i)$;
5 for $i \leftarrow 1$ to $n$ do
      $\boldsymbol{V}[\boldsymbol{i}] \leftarrow (\boldsymbol{S}[i]||\boldsymbol{R}_1[1+i \mod n]||\dots||\boldsymbol{R}_j[j+i \mod n]||\dots||\boldsymbol{R}_{t-1}[(t-1)+i$
       $\mod n])$;
6 return $\boldsymbol{V}$.
```

Algorithm 3.2: *Recover*($\boldsymbol{V}$).

```
1 for $i \leftarrow 1$ to $n$ do
      $(\boldsymbol{S}[i]||\boldsymbol{R}_1[1+i \mod n]||\dots||\boldsymbol{R}_j[j+i \mod n]||\dots||\boldsymbol{R}_{t-1}[(t-1)+i$
       $\mod n]) \leftarrow \boldsymbol{V}[i]$;
2 $s' \leftarrow Recover^{ECC}(\boldsymbol{S}')$;
3 for $i \leftarrow 1$ to $t-1$ do
      $r_i \leftarrow Recover^{ECC}(\boldsymbol{R}'_i)$;
4 $s = s' \oplus r_1 \oplus r_2 \oplus \cdots \oplus r_{t-1}$;
5 return $s$.
```

Figure 3.1: The share and recover algorithms defining the modified HP scheme.

3.2.1.1 The share algorithm

The share algorithm, as in Algorithm 3.1, is a probabilistic algorithm that inputs a secret $s \in \{0,1\}^\lambda$. The algorithm begins by randomly generating $t-1$ *dummy keys* of equal length to the secret, which are labelled $r_1, \dots, r_{t-1}$. The $t-1$ dummy keys and the secret s are all XORed to give $s' \in \{0,1\}^\lambda$.

Now, consider s' and each dummy key $r_1, \dots, r_{t-1}$ as a string of t words, where each word consists of $\lceil \frac{\lambda}{t} \rceil$ bits. This is achieved by parsing each of $s', r_1, \dots, r_{t-1}$ into t elements, with each element in $GF(2^{\lceil \frac{\lambda}{t} \rceil})$. This enables us to treat each value $s', r_1, \dots, r_{t-1}$ as a t-vector or, specifically, as an element in $\mathbb{F}^t$, where $\mathbb{F} = GF(2^{\lceil \frac{\lambda}{t} \rceil})$.

If λ is divisible by t, s' and the dummy keys will parse exactly into t words. If not, each string must be padded with $(-\lambda) \mod t$ elements to ensure that, when parsed, each word is an element in $\mathbb{F}$. For ease of notation, we assume λ is divisible by t.

So, the dummy keys $r_1, \dots, r_{t-1}$ and s' are treated as t-vectors and then independently dispersed via the IDA share algorithm, $Share^{ECC}$. This results in t n-vectors, which we denote as $\boldsymbol{S}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_{t-1} \in \mathbb{F}^n$. Let $\boldsymbol{R}_i[j]$ denote the j^{th} element in the vector $\boldsymbol{R}_i$.

Elements of each of the output vectors are then allocated to a vector $\boldsymbol{V}$, where the i^{th} element of $\boldsymbol{V}[i]$ is the share to be given to P_i. Elements are allocated to the vector $\boldsymbol{V}$ such that each share $\boldsymbol{V}[i]$ contains exactly t elements, where each element is from both a distinct vector (of $\boldsymbol{S}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_{t-1}$) and from a distinct position in their vector (so no two elements in the share come from the same position in their vector). The t elements allocated to share $\boldsymbol{V}[i]$ are then concatenated and given to player P_i, for $1 \leq i \leq n$. Note that $\boldsymbol{V}[i] \in \mathbb{F}^t$.

One possible way to allocate elements to the share vector can be illustrated by constructing a $t \times n$ matrix D, where each n-vector $\boldsymbol{S}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_{t-1}$ defines a row of D, so

$$D = \begin{pmatrix} \boldsymbol{S}[1] & \boldsymbol{S}[2] & \ldots & \boldsymbol{S}[n-1] & \boldsymbol{S}[n] \\ \boldsymbol{R}_1[1] & \boldsymbol{R}_1[2] & \ldots & \boldsymbol{R}_1[n-1] & \boldsymbol{R}_1[n] \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \boldsymbol{R}_{t-1}[1] & \boldsymbol{R}_{t-1}[2] & \ldots & \boldsymbol{R}_{t-1}[n-1] & \boldsymbol{R}_{t-1}[n] \end{pmatrix}.$$

A new matrix, D', can then be constructed from D by shifting elements in row i, for $1 \leq i \leq t$, $i-1$ places to the left. So, the first row consisting of $\boldsymbol{S}$ does not change, but the second row of D', consisting of $\boldsymbol{R}_1$ shifts each element one place to the left from D', and so on. Call D' the *dispersal matrix*, where:

$$D' = \begin{pmatrix} \boldsymbol{S}[1] & \boldsymbol{S}[2] & \ldots & \boldsymbol{S}[n-1] & \boldsymbol{S}[n] \\ \boldsymbol{R}_1[2] & \boldsymbol{R}_1[3] & \ldots & \boldsymbol{R}_1[n] & \boldsymbol{R}_1[1] \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \boldsymbol{R}_{t-1}[t+1] & \boldsymbol{R}_{t-1}[t+2] & \ldots & \boldsymbol{R}_{t-1}[t-1] & \boldsymbol{R}_{t-1}[t] \end{pmatrix}.$$

The elements in column j of D' are then allocated to the j^{th} element of $\boldsymbol{V}$, denoted $\boldsymbol{V}[j]$, and then given to P_j as their share.

3.2.1.2 The recover algorithm

The recover algorithm, as in Algorithm 3.2, inputs an n-vector $\boldsymbol{V}$, where the i^{th} element of the vector, $\boldsymbol{V}[i]$, is either a string $\boldsymbol{V}[i] \in \{0,1\}^*$, or the value $\Diamond$. In the first case, $\boldsymbol{V}[i]$ is the purported share of player P_i. In the second case, $\boldsymbol{V}[i] = \Diamond$, the share from player P_i has not been submitted and so is marked as missing.

Each share $\boldsymbol{V}[i]$ is parsed into the t elements from column i of D'. From these values, the t-vectors $\boldsymbol{S}', \boldsymbol{R}'_1, \ldots, \boldsymbol{R}'_{t-1}$ are constructed, which are then used to recover the values s' and $r_1, r_2, \ldots, r_{t-1}$ via the $\mathit{Recover}^{ECC}$ algorithm. These values are then XORed to retrieve the secret, s.

3.2.1.3 Comparison to the original HP scheme

In the modified HP scheme, presented in Section 3.2.1, $t-1$ random dummy keys are generated. This is an improvement on the original HP scheme, where t random dummy keys are generated [21]. The generation of fewer dummy keys improves the scheme to be more lightweight, as generating true randomness is hard, and results in smaller dimensions for the matrices D and D' which thereby decreases the size of the shares given to each player and increases the information rate of the scheme.

Additionally, the original HP scheme did not specify the security requirements of the IDA, whereas we have noted the IDA must have equivalent security properties to a linear $(0, t; n)$-ramp scheme.

3.2.2 Security analysis

It will now be proved that the modified HP scheme presented in Section 3.2.1, and summarised in Figure 3.1, satisfies the recoverability and privacy requirements for a perfect (t, n)-threshold scheme, as in Definition 2.3.10. The original HP scheme has no such analysis [21] and so this security analysis is novel. It is noted that the proof given here can be easily adapted to prove the original HP scheme is also perfectly secure.

The structure of the proof is as follows. In Lemma 3.2.1, it is shown that any distribution of elements from the matrix D to the n players that allows any t players to learn at least t shares in every row of D will allow recovery of the secret. Theorem 3.2.2 then shows that the distribution of elements from D via D' satisfies this condition, thus the scheme achieves recoverability. Lemma 3.2.3 shows that any distribution of elements from the matrix D that allows no more than $t-1$ players to learn at most $t-1$ elements in each column of D achieves privacy. Theorem 3.2.4 shows that the distribution of elements from D via D' satisfies this property, thus achieving privacy. Together, Theorems 3.2.2 and 3.2.4 show that the modified HP scheme satisfies the requirements to be a perfect (t, n)-threshold scheme, as in Definition 2.3.10.

3.2.2.1 Recoverability

Lemma 3.2.1. *Let n players be allocated elements from the matrix D. If any set of at least t players learn at least t elements in every row of D, s can be recovered.*

Proof. Assume elements from the $t \times n$ matrix D are allocated such that any t players learn at least t shares in every row. We wish to show that a set of (at least) t players can pool their shares and learn $s', r_1, \ldots, r_{t-1}$, then recover s.

Let D_i denote the i^{th} row of D, for $1 \leq i \leq t$. Each row D_i, when transposed and considered as an n-vector, is the output of a (t, n)-IDA. In particular, $D_1^T \leftarrow Share^{ECC}(s')$ and $D_{i+1}^T \leftarrow Share^{ECC}(r_i)$ for $1 \leq i \leq t-1$.

For any row D_i, t players can pool their corresponding elements to form a t-vector, which can be used as input to $Recover^{ECC}$, which will return the dispersed value: s' if $i = 1$, or r_i if $2 \leq i \leq t$. The players can repeat this procedure for every row of D and reconstruct all the values $s', r_1, \ldots, r_{t-1}$. Once these values are obtained, they are able to XOR them and output s. □

Theorem 3.2.2. *The modified HP (t, n)-threshold scheme satisfies the property that any set of t players learn at least t elements in every row of D.*

Proof. Each player is given a column of the matrix D', where D' is formed by shifting row

i of D, for $1 \leq i \leq t$, i places to the left. As each player is allocated a distinct column of D', each player is necessarily given a distinct element from every row. Therefore, when any t players pool their shares, there will be t distinct elements in every row. □

Theorem 3.2.2 shows that the modified HP scheme meets the requirements in Lemma 3.2.1 and hence meets the recoverability requirements of a perfect (t, n)-threshold scheme.

3.2.2.2 Privacy

Now we must prove the privacy requirement is also satisfied. Intuitively, Lemma 3.2.3 shows that an unauthorised set of players must be prevented from learning all elements in a given column of D, otherwise the players could calculate the corresponding part of s by XORing all elements in that column.

Lemma 3.2.3. *If elements from D are allocated to shares such that any unauthorised set of at most $t-1$ players learn no information about at least one element from every column, then no information is learnt about s. In information theoretic terms, $H(\boldsymbol{\mathcal{S}}) = H(\boldsymbol{\mathcal{S}} \mid U)$, where U is the set of shares given to the set of unauthorised players.*

Proof. Let s be a string of λ bits. Assume an unauthorised set of at most $t-1$ players collectively has the set U of shares, made of elements from the matrix D. Assume the set of players can learn no information about at least one element from every column in D.

Using the result from Theorem 2.4.6 and working in the field $\mathbb{F} = GF(2^\ell)$, we can see that XORing any of the first t columns of the matrix D will give the corresponding fragment of the secret s. Similarly, if we XOR any of the final $n-t$ columns of D, the output will be the corresponding entry of the codeword vector output by the IDA share algorithm with s as its input.

Now, let $D[i]$ denote the i^{th} column of D. For any given column i, for $1 \leq i \leq n$, U can contain any number of elements (but not every element) of $D[i]$. So U will contain at most $t-1$ of the t elements in each column. Without loss of generality, choose a column j; we will prove no information is learnt about s from this individual column. This argument can then be applied to every other column of D.

Denote the set of elements in column j and not in the set U as U'_j. Note that $|U'_j| \geq 1$. Let u_j be the XOR of all the elements in U. Let x_j be the XOR of all elements in U'_j, so x_j is the XOR of all elements in column j that are unknown to U. Assume the secret s is also distributed via the IDA, resulting in the n-vector $\boldsymbol{S}^*$. Denote the i^{th} element of $\boldsymbol{S}^*$ as $\boldsymbol{S}^*[i]$. Note that $u_i \oplus x_i = \boldsymbol{S}^*[i]$. As the dummy keys are all randomly generated, each value r_i has entropy 2^λ. Thus each element of $\boldsymbol{R}_i$ has entropy $2^\lambda/t$, and so $H(x_j) = 2^\lambda/t$.

Now, we can equate this to a OTP, presented in Definition 2.2.2, as follows. The value u_i is equivalent to a known ciphertext and x_i is equivalent to an unknown key. The XOR of these would reveal the plaintext message, which here is $\boldsymbol{S}^*[i]$. As x_i has entropy $H(x_i) = 2^\lambda$, the value $\boldsymbol{S}^*[i]$ also has entropy 2^λ. Thus no information is learnt about $\boldsymbol{S}^*[i]$.

This is true for any $1 \leq i \leq n$, and thus no information about s is learnt by the unauthorised set of at most $t-1$ players with the combined set of shares U. □

Theorem 3.2.4. *The modified HP (t, n)-threshold scheme satisfies the condition that any set of $t-1$ players learns no information about at least one element in every column.*

Proof. In the scheme, each player is given a column of the matrix D', so each player will receive exactly one element from each row of D. Therefore, any set of $t-1$ players will learn exactly $t-1$ elements of each row. As the IDA is a linear $(0, t; n)$-ramp scheme and because each of the dummy keys and s are generated uniformly at random, players with only $t-1$ elements are unable to learn any further elements from each row.

Each player will also be given elements that come from $n-t$ distinct columns of D. Therefore, a set of up to $t-1$ players can pool their shares and learn at most $t-1$ shares in each column. □

Theorem 3.2.4 proves the modified HP scheme meets the requirements of Lemma 3.2.3 and thus satisfies the privacy requirement. Therefore the modified HP scheme presented in Section 3.2, and summarised in Figure 3.1, satisfies both the recoverability and privacy requirements to be a perfect (t, n)-threshold scheme.

3.3 Efficiency analysis

In this section, we analyse the efficiency of the modified HP scheme, defined in Figure 3.1, then analyse other efficient, perfect threshold schemes in the literature and compare them to the modified HP scheme.

All schemes considered are ideal and require the minimum number of random bits. Our comparison focuses on the efficiency of the share and recover algorithms and, in particular, considers the number of bitwise operations required for each of the algorithms separately. This allows us to highlight which scheme may be best in environments where the efficiency of either the share or recover algorithm is prioritised. Note that, for the analysis and comparison, we follow [62] and consider the number of bitwise operations required, including the number of XORs and multiplications, but do not consider the number of bit shifts.

The comparable schemes we consider, besides the modified HP scheme, are Shamir's threshold scheme [91], which is a benchmark scheme for secret sharing and the scheme presented by Kurihara *et al.* [62], which claims to be the first ideal, perfect (t, n)-threshold scheme implementable using only XOR operations. We also briefly discuss an ideal threshold scheme by Wang and Desmedt [99].

3.3.1 The modified HP scheme

Here, we analyse the efficiency of the modified HP scheme presented in Section 3.2.1 and summarised in Figure 3.1.

3.3.1.1 Information rate

We show that the modified HP scheme, which is a perfect (t, n)-threshold scheme, is ideal and thus is an improvement on the original HP scheme [21].

Theorem 3.3.1. *Let $s \in \{0,1\}^\lambda$. The (t, n)-threshold scheme is ideal if λ is divisible by t. If λ is not divisible by t, each share has $\lambda \mod t$ more elements than the secret.*

Proof. In general, $(-\lambda) \mod t$ bits of padding will be added to s to ensure each element in the matrix D will consist of $\lceil \frac{\lambda}{t} \rceil$ bits. Each player will receive t elements from D, and thus the size of each player's share will be $\lceil \frac{\lambda}{t} \rceil t$ bits. Therefore, the information rate is calculated as

$$\rho = \lambda \Big/ \left(\left\lceil \frac{\lambda}{t} \right\rceil t \right).$$

If λ is divisible by t, the information rate of the scheme is equal to one and the scheme is ideal. □

In contrast, the original HP scheme required t strings of λ bits to be randomly generated, rather than $t-1$ strings [21]. As a result, D is a $(t+1) \times n$ matrix, with each element consisting of $\lceil \frac{\lambda}{t} \rceil$ bits. Each player is required to store $t+1$ elements of D as their share, and so the information rate of the original HP scheme is

$$\rho = \lambda \Big/ \left(\left\lceil \frac{\lambda}{t} \right\rceil t + 1 \right).$$

Assuming that λ is divisible by t, the information rate of the original HP scheme can be written as:

$$\rho = \frac{t}{t+1}.$$

3.3.1.2 Complexity of share algorithm

With respect to randomness, the share algorithm of the modified HP scheme requires the generation of $\lambda(t-1)$ bits, treated as $t-1$ strings of length λ. These are the strings $r_1, \ldots, r_{t-1}$. This is the minimum possible number of random bits that could be generated in order for the scheme to be perfectly secure [15], and thus, in this respect, the threshold scheme is optimal.

These random strings are XORed, along with the secret s, to calculate s'. In total, this requires $t-1$ XORs of λ bit strings. This is equivalent to $\lambda(t-1)$ XORs in the field $GF(2^\lambda)$.

Each value $s', r_1, \ldots, r_{t-1}$ is then dispersed via the IDA algorithm $Share^{ECC}$. This is computationally the most expensive operation and the complexity depends on the choice of IDA. If the systematic version of Rabin's IDA is used, described in Section 2.4.3.2, each distribution requires $\mathcal{O}(t(n-t))$ operations due to the matrix-vector multiplication. Specifically, if we consider this in more detail, each distribution requires $t(n-t)$ multiplications and $(n-t)(t-1)$ additions in the field $GF(2^{\lceil \frac{\lambda}{t} \rceil})$. As there are t values to be distributed $(s', r_1, \ldots, r_{t-1})$, the dispersal of all values requires a total of $t^2(n-t)$ multiplications and $t(n-t)(t-1)$ additions. (Note that we do not consider the computation costs of constructing the dispersal matrix for the IDA, as it is a public matrix that can be pre-computed and reused each time.) Each player is then allocated elements output by the IDA, which requires no additions or multiplications.

Therefore, a total of $t(t-1)(n-t) + t(t-1)$ additions and $t^2(n-t)$ multiplications in the field $GF(2^{\lceil \frac{\lambda}{t} \rceil})$ are required.

Let us give the complexity of the modified HP share algorithm using big 'O' notation. Obviously, XORing two λ bit strings requires λ bitwise XORs, and if we assume that multiplication of two λ bit numbers requires $2\lambda^2$ bitwise XORs (λ^2 operations to multiply the strings and another λ^2 operations to compute modular reduction), we can say the share algorithm requires

$$\frac{\lambda}{t}\left(t(t-1)(n-t) + t(t-1)\right) + 2\left(\frac{\lambda}{t}\right)^2 t^2(n-t)$$

bitwise operations, which gives a complexity of $\mathcal{O}(\lambda^2 n)$, as, in general, $\lambda > t$.

Pre-computation: We may have the ability to compute some operations required prior to the secret s being submitted. Pre-computing operations can minimise the complexity of the algorithm after s is submitted. In particular, the $t-1$ dummy keys $r_1, \ldots, r_{t-1}$ can be generated and XORed in advance. Then, when the secret s is submitted, only one XOR of λ bits is left to compute $s' = s \oplus (r_1 \oplus \ldots \oplus r_{t-1})$. The $t-1$ dummy keys can also be distributed via the IDA, which allows us to compute the majority of the operations in advance. Specifically, we can pre-compute $t(t-1)(n-t)$ multiplications and $(t-1)^2(n-t)$ additions in the field $GF(2^{\lceil \frac{\lambda}{t} \rceil})$, leaving only $t+t(n-t)$ additions and $(n-t)(t-1)$ multiplications after the submission of s.

If we again want to use big 'O' notation, the pre-computation bitwise complexity is $\mathcal{O}(\lambda^2 n)$, whilst the bitwise complexity after s is submitted is reduced to $\mathcal{O}(\lambda^2 n/t)$. Therefore, allowing pre-computation reduces the bitwise complexity of the algorithm

	Additions	**Multiplications**
Before s is input	$(t-1)^2(n-t)+t(t-2)$	$t(t-1)(n-t)$
After s is input	$(t-1)(n-t)+t$	$t(n-t)$
Total	$t((t-1)(n-t)+(t-1))$	$t^2(n-t)$

Table 3.1: The number of operations in the field $GF(2^{\lceil \frac{\lambda}{t} \rceil})$ computable before and after the secret is submitted to the share algorithm of the modified HP scheme.

run after s is submitted.

Table 3.1 summarises the exact number of operations that can be pre-computed (prior to knowledge of the input s), the number of operations that must wait until after s is submitted, and the total number of operations. Note that the operations are computed in the field $GF(2^{\lceil \frac{\lambda}{t} \rceil})$.

3.3.1.3 Complexity of recover algorithm

The recover algorithm is initiated by at least t players submitting shares. The first step for the algorithm to take is to parse the shares into their respective parts and recover the values $s, r_1, \dots, r_{t-1}$. This is done via the IDA recover algorithm, which, if an IDA such as Rabin's is used, requires the inversion of a $t \times t$ matrix G' followed by a matrix-vector product. Matrix inversion could be achieved via Gauss-Jordan elimination, this requires approximately $2t^3/3$ operations [41]. More specifically, Gauss-Jordan elimination will require about $t(t-1)(2t+5)/6$ additions and $t(t^2+3t-1)/3$ multiplications in $GF(2^{\lceil \frac{\lambda}{t} \rceil})$.

Once the inverse of G' has been computed, it must be multiplied with each t-vector $\boldsymbol{S'}, \boldsymbol{R'_1}, \dots, \boldsymbol{R'_{t-1}}$. Each matrix-vector multiplication requires t^2 multiplications and $t(t-1)$ additions in the field $GF(2^{\lceil \frac{\lambda}{t} \rceil})$. This must be done for each of the t vectors, totalling t^3 multiplications and $t^2(t-1)$ additions. Finally, the strings $s', r_1, \dots, r_{t-1}$ must be XORed. This requires $(t-1)$ additions in the field $GF(2^\lambda)$, which is equivalent to about $t(t-1)$ additions in $GF(2^{\lceil \frac{\lambda}{t} \rceil})$.

So, the recover algorithm requires a total of $t^2(t-1)+t(t-1) = t^3 - t$ additions and t^3 multiplications in $GF(2^{\lceil \frac{\lambda}{t} \rceil})$, plus the operations required to invert the $t \times t$ matrix G'.

This is a grand total of $t^2(t-1) + (t(t-1)(2t+5)/6)$ additions and $t^3 + (t(t^2+3t-1)/3)$ multiplications in $GF(2^{\lceil \frac{\lambda}{t} \rceil})$.

As with the share algorithm, we know the XOR of two λ bit strings requires λ bitwise XORs, and we can assume multiplication of two λ bit strings requires $2\lambda^2$ bitwise operations, therefore we can calculate the number of bitwise XORs and the complexity of the recover algorithm to be:

$$\frac{\lambda}{t}\left(t^2(t-1) + \frac{t(t-1)(2t+5)}{6}\right) + \frac{\lambda^2}{t}\left(t^3 + \frac{t(t^2+3t-1)}{3}\right) = \mathcal{O}(\lambda^2 t).$$

Pre-computation: In the recover algorithm, there are not many computations that could be calculated in advance of the players submitting their shares. The only operation that could possibly be computed is the inversion of the matrix G' required for the IDA recover algorithm. Note that the matrix G' is constructed according to the identities of the players submitting shares to the recover algorithm, thus it is easy to pre-compute the inverse if we know which players will, or are likely to, submit shares to the recover algorithm. However, if we do not know which players will submit (or are likely to submit) their shares, there are a total of $\binom{n}{t}$ possible sets of players that could submit their shares and thus there are that many possibilities for the matrix G'. Constructing every possible G' and computing each inverse may be computationally heavy and unnecessary, dependent on the parameters n and t. For example, if $n = 4$ and $t = 3$, there are only four possibilities for the matrix G', so it may be reasonable to pre-compute the four inverses for the four possibilities of G'. But if $n = 16$ and $t = 10$, there are 8008 possibilities for G', which may be too heavy to pre-compute. In this case, inverting the matrix G' will have to be delayed until after the players have submitted their shares and their identities are known.

If G' is able to be constructed and inverted prior to s being submitted, the recover algorithm requires only t^3 multiplications and $t^2(t-1) + t(t-1) = t^3 - t$ additions in $GF(2^{\lceil \frac{\lambda}{t} \rceil})$, which is a complexity of $\mathcal{O}(\lambda^2 t)$.

3.3.2 Shamir's threshold scheme

Shamir's perfect threshold scheme was initially introduced in Construction 2.3.3.

3.3.2.1 Description of Shamir's threshold scheme

In order to allow for easy comparison, we consider Shamir's threshold scheme over the finite field $GF(2^\lambda)$, where $2^\lambda \geq n$. Recall that, for some secret $s \in GF(2^\lambda)$, sharing a secret requires evaluating a polynomial $f(x) \in GF(2^\lambda)[x]$, with coefficients $r_1, r_2, \ldots, r_{t-1}$ chosen uniformly at random from $GF(2^\lambda)$. The share given to player P_i is $f(x_i)$, for $1 \leq i \leq n$. Recovery requires polynomial interpolation on any t shares.

3.3.2.2 Complexity of Shamir's share algorithm

As already discussed, Shamir's scheme requires the minimal amount of randomness and is ideal.

In order to calculate the player's shares, a polynomial of degree $t-1$, namely

$$f(x) = r_{t-1}x^{t-1} + r_{t-2}x^{t-2} + \cdots + r_1 x + s, \tag{3.3.1}$$

must be evaluated at n distinct points. We briefly consider three methods of decreasing complexity for how to evaluate a polynomial at a point, x. In each method, we count the number of multiplications and additions in the field $GF(2^\lambda)$.

1. *Naïve approach:* We could independently compute each term $r_i x^i$, for $0 \leq i \leq t-1$, which would require $t(t-1)/2$ multiplications and $t-1$ additions for each x.
2. *Recursively computing x^i:* Compute x^i recursively, so once x^i has been computed, store this and calculate x^{i+1} by computing just one more multiplication, $x^{i+1} = x^i \cdot x$. This would require a total of $2(t-1)-1$ multiplications and $t-1$ additions.
3. *Nested Multiplications:* Treat the polynomial as a sequence of additions and multiplications and iteratively evaluate it. For example, for different degree polynomials, we can treat the polynomial as a nested sequence:

$$\begin{aligned} &\text{if } t-1 = 2, \qquad f(x) = s + x(r_1 + r_2 x) \\ &\text{if } t-1 = 3, \qquad f(x) = s + x(r_1 + x(r_2 + r_3 x)) \\ &\qquad\qquad \ldots . \end{aligned}$$

In general,

$$f(x) = s + x(r_1 + x(r_2 + \ldots x(r_{t-2} + r_{t-1}x)\ldots)).$$

Then compute the following values:

$$\begin{aligned} b_{t-1} &= r_{t-1} \\ b_{t-2} &= r_{t-2} + xb_{t-1} \\ &\ldots \\ b_1 &= r_1 + xb_2 \\ f(x) = b_0 &= s + xb_1. \end{aligned}$$

This method requires just $t-1$ multiplications and $t-1$ additions.

If we use the method of nested multiplications, the evaluation of each of the n values $f(i)$, for $1 \le i \le n$, requires a total of $n(t-1)$ additions and $n(t-1)$ multiplications in $GF(2^\lambda)$. If we wish to use big 'O' notation, this is a total number of bitwise operations and complexity of

$$\lambda n(t-1) + 2\lambda^2 n(t-1) = \mathcal{O}(\lambda^2 nt).$$

Pre-computation: Prior to s being submitted, we are able to generate the random values $r_1, \ldots, r_{t-1}$. We can compute the majority of the operations in the polynomial: in fact, for each i, we can compute $f(i) - s$. Then, when the secret is input, we can add s to each pre-computed value and all the shares are thus calculated. If we employ the third, most efficient method of evaluating $f(x)$, this allows us to pre-compute all $n(t-1)$ multiplications and $n(t-2)$ additions. After s is submitted, only n additions are left to compute.

The pre-computation has a bitwise complexity of $\mathcal{O}(\lambda^2 nt)$. After s is submitted, the complexity of computation required is $\mathcal{O}(\lambda n)$.

Table 3.2 summarises the number of additions and multiplications required for Shamir's share algorithm and highlights the number that can be pre-computed.

	Additions	**Multiplications**
Before s is input	$n(t-2)$	$n(t-1)$
After s is input	n	0
Total	$n(t-1)$	$n(t-1)$

Table 3.2: The number of operations in the field $GF(2^\lambda)$ computable before and after the secret is submitted to Shamir's share algorithm.

3.3.2.3 Complexity of Shamir's recover algorithm

In order to recover the secret, Shamir's recover algorithm requires the submission of t shares. Without loss of generality, we denote the t submitted shares as $v_1, \ldots, v_t$. These shares are used as input to a polynomial interpolation algorithm, which will output the original polynomial f.

As an example, Lagrange interpolation could be used to compute $f(x)$ by letting $(x_i, y_i) = (i, v_i)$ and calculating

$$f(x) = \sum_{j=1}^{t} f_j(x),$$

where

$$f_j(x) = y_j \prod_{\substack{k=1 \\ k \neq j}}^{t} \frac{x - x_k}{x_j - x_k}.$$

The computation of each $f_j(x)$ requires $t-2$ subtractions (which are counted as additions) and $t-1$ multiplications. This needs to be repeated for each $1 \leq j \leq t$, then each $f_j(x)$ must be summed to calculate $f(x)$, which is an additional $t-1$ additions. So, Lagrange interpolation requires a total of $t(t-1)+(t-1) = t^2-1$ additions and $t(t-1)$ multiplications in $GF(2^\lambda)$. Thus, we can calculate the complexity of Shamir's recover algorithm to be

$$\lambda(t^2-1) + 2\lambda^2 t(t-1) = \mathcal{O}(\lambda^2 t^2).$$

Polynomial evaluation algorithms that have a lower asymptotic complexity of $\mathcal{O}(\lambda^2 t \log^2 t)$ are discussed in [59] and [2]. If one of these algorithms were to be used, the recover algorithm would have complexity $\mathcal{O}(\lambda^2 t \log^2 t)$ instead of $\mathcal{O}(\lambda^2 t^2)$. Which algorithm is used in practice depends on the parameters t, n and λ and the application. Given the asymptotic nature of big ‘O’ notation, it may be that an algorithm with a higher bitwise complexity actually requires fewer operations for smaller parameters (because big ‘O’ notation does not take into account the coefficients or other terms). Therefore, when comparing asymptotic complexities, we will take into account these more efficient algorithms with lower complexities, but when we are explicitly considering the exact number of operations in the scheme, we will use the number of multiplications and XORs computed using Lagrange interpolation.

Pre-computation: No computations are possible prior to the t players submitting their shares.

3.3.3 The threshold scheme by Kurihara *et al.*

In [62], Kurihara *et al.* present the first XOR-based, ideal, perfect (t, n)-threshold scheme. We discuss and analyse their scheme here.

3.3.3.1 Description of the scheme by Kurihara *et al.*

Assume a secret $s \in \{0,1\}^{\lambda}$ is to be distributed. The share algorithm begins by fixing a prime $p \geq n$, such that $\lambda = d(p-1)$ for some d, then parses s into $p-1$ elements, with each element consisting of d bits. Next, they generate $(t-1)(p-1)$ random strings, each consisting of d bits, then XOR sets of the random strings with specified elements of s to create $n(p-1)$ other elements, $\omega_{(i,j)}$, for $1 \leq i \leq n$ and $1 \leq j \leq p-1$. Of these values, $p-1$ are allocated to each share.

In order to reconstruct the secret, t players submit their shares to the recover algorithm, which are each parsed into their $p-1$ d-bit elements $\omega_{(i,j)}$. From all these elements, they construct a $t(p-1)$-dimensional vector $\boldsymbol{w}$ with each element being a d bit string. They then define a function $MAT(\cdot)$ which takes as input the indices of the collaborating players and uses forward and backward substitution to output a binary $(p-1) \times t(p-1)$ matrix D such that $D \cdot \boldsymbol{w} = \boldsymbol{s}$, where $\boldsymbol{s}$ is a $(p-1)$-dimensional vector with d bit elements such that the concatenation of elements of $\boldsymbol{s}$ reveals the secret s.

3.3.3.2 Complexity of the share algorithm by Kurihara *et al.*

As each player receives $p-1$ of the $\omega_{(i,j)}$ values, where each $\omega_{(i,j)}$ consists of d bits, each share is $(p-1)d = \lambda$ bits and thus the scheme is ideal. Also, the scheme requires the random generation of $(t-1)(p-1)d = \lambda(t-1)$ bits, which is the minimum number possible in order for the scheme to be perfectly secure. In this respect, the scheme by Kurihara *et al.* is optimal.

Now, we consider the number of operations required to share s. Interestingly, the scheme by Kurihara *et al.* requires only XORs and no multiplications. The only computation required is during the step that XORs sets of the random strings with specified elements of s. Of the n shares, $n-1$ shares require $(p-2)(t-1)d + (t-2)d$ XORs to compute, whereas the n^{th} shares requires $(p-1)(t-1)d$ XORs. This is a total of

$$d(npt - np - nt + 1)$$

bitwise operations.

Using big 'O' notation, we say the share algorithm has a bitwise complexity of $\mathcal{O}(dnpt)$. As $d(p-1) = \lambda$, we count this as $\mathcal{O}(\lambda nt)$.

Pre-computation: As with the modified HP scheme, the random strings can be generated and XORed prior to the submission of s. This means that we can compute $(p-2)(t-2)d + (t-2)d$ XORs for the first $n-1$ shares, and $(p-1)(t-2)d$ XORs for the last share. After some rearranging, this is a total of

$$d(p-1)(t-2)n$$

bitwise operations, leaving $(p-2)(n-1)d + (p-1)d$ bitwise XORs to be computed after s is submitted. This is a complexity of $\mathcal{O}(\lambda nt)$ before s is submitted, and $\mathcal{O}(d(p-2)n) \approx \mathcal{O}(\lambda n)$ after.

Table 3.3 shows the number of XORs needed before and after s is submitted to the share algorithm, as well as the total number of bitwise XORs.

	Additions
Before s is input	$d(p-1)(t-2)n$
After s is input	$(p-2)(n-1)d+(p-1)d$
Total	$d(n(pt-p-t)+1)$

Table 3.3: The number of bitwise XORs computable before and after the secret is submitted to the share algorithm by Kurihara *et al.*, where $\lambda = d(p-1)$ for some prime $p \geq n$.

3.3.3.3 Complexity of the recover algorithm by Kurihara *et al.*

The only steps of the recover algorithm that require computations (other than bit shifts, concatenations and parsing), are the function $MAT(\cdot)$ and the matrix-vector multiplication $D \cdot \boldsymbol{w}$.

We first consider the number of operations in the function $MAT(\cdot)$. The function $MAT(\cdot)$ requires forward substitution on a $(t(p-1)) \times (tp-2)$ matrix, and backward substitution on a $(p-1) \times (t+1)(p-1)$ matrix. Using big 'O' notation, the authors claim this is a bitwise complexity of $\mathcal{O}(t^3p^3)$. When calculating the number of operations, we assume the coefficient is 2/3, as is true with Gaussian elimination, and so will count this as a total of $(2t^3p^3)/3$ bitwise operations.

Next, we analyse the number of operations required to compute $D \cdot \boldsymbol{w} = \boldsymbol{s}$. Each element of $\boldsymbol{s}$ requires a total of $t(p-1)$ multiplications between the binary elements of M and the d bit elements of $\boldsymbol{w}$, and $t(p-1)-1$ XORs of d bit elements. There are $p-1$ elements in $\boldsymbol{s}$, so this is a total of $t(p-1)^2$ multiplications (of binary elements with d bit strings) and $(t(p-1)-1)(p-1)$ additions (of d bit strings).

If we implement multiplication between elements of M and $\boldsymbol{w}$ such that, for any $a \in \{0,1\}$ and $b \in \{0,1\}^d$

$$a \cdot b = \begin{cases} b \text{ if } a = 1 \\ 0 \in \{0,1\}^d \text{ if } a = 0, \end{cases}$$

then multiplication can be implemented via a look-up table and thus we count each multiplication as one bitwise operation. So, calculating $M \cdot \boldsymbol{w} = \boldsymbol{s}$ requires $t(p-1)^2$ bitwise operations (attributed from the multiplications), plus $(t(p-1)-1)(p-1)$ XORs of d bit strings. This is a total of $d(tp-t-1)(p-1)+t(p-1)^2$ bitwise operations.

We can then compute the total complexity of this recover algorithm as

$$d(tp-t-1)(p-1)+t(p-1)^2+\frac{2}{3}t^3p^3=\mathcal{O}(t^3p^3)\approx\mathcal{O}(t^3n^3),$$

given that p and n are of the same magnitude.

Pre-computation: In [62], they claim the function $MAT(\cdot)$ can be pre-computed, *i.e.* prior to the shares being submitted to the recover algorithm. However, just as in the modified HP recover algorithm, the $MAT(\cdot)$ function acts on the identities of players submitting shares to the recover algorithm: for example, if $t=3$ and players P_1, P_2 and P_3 submitted shares, $MAT(1,2,3)$ would need to be computed, whereas if players P_1, P_2 and P_4 submitted shares, $MAT(1,2,4)$ should be computed. Thus, as with the recover algorithm of the modified HP scheme, whether or not $MAT(\cdot)$ is pre-computed depends on whether it is known which players will submit shares to the recover algorithm, and the parameters t and n (in case all possibilities for $MAT(\cdot)$ are computed in advance).

So, the computation of $MAT(\cdot)$ must wait until after the players have submitted shares, unless it is know which players will submit, or are likely to submit, their shares to the recover algorithm.

If we are able to pre-compute the $MAT(\cdot)$ function, the complexity of operations required after s is submitted is

$$d(tp-t-1)(p-1)+t(p-1)^2=\mathcal{O}(dtp^2)\approx\mathcal{O}(\lambda tn),$$

as $d(p-1)=\lambda$ and p is of the same magnitude as n.

3.3.4 Wang and Desmedt's threshold scheme

In [99], Wang and Desmedt present an efficient, ideal (t,n)-threshold scheme for $\lambda \geq n$, equivalent to an $[q,t,q-t+1]$-MDS code, for some prime $q \geq \lambda+1$. They define the scheme to be a collection of $t \times n$ matrices, where each matrix is a different distribution of the secret, and each column of any given matrix is a share. As no specific method of constructing these matrices is defined, we do not describe the scheme in great detail, but do include it as another example of a perfect, ideal, efficient threshold scheme.

One possible share algorithm involves constructing a t-vector from the secret and multiplying this vector with a $t \times n$ binary matrix. If multiplication is implemented as a look-up table, the bitwise complexity of constructing this matrix and computing the product is $\mathcal{O}(\lambda n t)$. The binary matrix can be generated in advance, but the matrix-vector product cannot be pre-computed. Therefore, the ability to compute computations in advance does not reduce the complexity of operations required after s is submitted.

The recover algorithm for the scheme is equivalent to a decoding procedure presented by Blaum and Roth [12] that decodes an array code with at most r erasures and no errors. The procedure requires $\mathcal{O}(r(q^2 + r))$ XOR operations, for some prime $q \geq \lambda$. The number of erasures Wang and Desmedt's scheme can correct is $n - t$, thus the decoding algorithm requires $\mathcal{O}((n-t)(q^2 + n - n)) = \mathcal{O}(q^2 n)$ bitwise XORs. As q is of the same magnitude as λ, we say the recover algorithm requires $\mathcal{O}(\lambda^2 n)$ bitwise XOR operations.

3.3.5 Discussion

All schemes considered are ideal and require the minimal amount of randomness. In this respect, all schemes are optimal. However, the schemes range in efficiency for the share and recover algorithms. We discuss and compare these now.

Table 3.4 shows the bitwise complexities for the share and recover algorithms. The analysis of the share algorithm is separated into three columns: the first shows the complexity of the operations that can be pre-computed; the second shows the complexity of operations that must wait to be computed until after s is submitted, and the third is the complexity, assuming no pre-computation is possible.

3.3.5.1 Comparing share algorithms

Without pre-computation. Let us consider the efficiency of the share algorithms without pre-computation. Clearly, from Table 3.4, the schemes by Kurihara *et al.* and Wang and Desmedt achieve the lowest complexities. The modified HP scheme has the next lowest complexity, with Shamir's scheme having the highest.

	Share			**Recover**
Scheme	**Before s is submitted**	**After s is submitted**	**Total**	**Total**
Modified HP	$\mathcal{O}(\lambda^2 n)$	$\mathcal{O}(\lambda^2 n/t)$	$\mathcal{O}(\lambda^2 n)$	$\mathcal{O}(\lambda^2 t)$
Shamir [91]	$\mathcal{O}(\lambda^2 nt)$	$\mathcal{O}(\lambda n)$	$\mathcal{O}(\lambda^2 nt)$	$\mathcal{O}(\lambda^2 t \log^2 t)$
K. *et al.* [62]	$\mathcal{O}(\lambda nt)$	$\mathcal{O}(\lambda n)$	$\mathcal{O}(\lambda nt)$	$\mathcal{O}(\lambda nt + n^3 t^3)$
W. & D. [99]	$\mathcal{O}(\lambda nt)$	$\mathcal{O}(\lambda nt)$	$\mathcal{O}(\lambda nt)$	$\mathcal{O}(\lambda^2 n)$

Table 3.4: Complexity of share and recover algorithms for different perfect threshold schemes.

Now, as already discussed, big 'O' complexity does not take into account any coefficients or smaller terms and is useful for giving the complexity when the parameters are very large. In a setting such as ours, when the parameters λ, n and t are reasonably small (an example implementation in [8] requires $\lambda = 128, n = 16, t = 10$), it may be the case that the schemes with higher complexities are able to run with fewer bitwise operations than those with lower complexities.

Considering the schemes in more detail than the asymptotic complexity is difficult as there are a number of variables; we cannot straightforwardly compare the number of multiplications and XORs in the schemes as, in general, these operations are computed in different fields. On top of this, much of the efficiency depends on the implementation of each scheme and the hardware it runs on. Note that we have already assumed multiplication of two λ bit strings requires $2\lambda^2$ bit operations (λ^2 bitwise operations for multiplication and a further λ^2 for modulo reduction), but this is a fairly naïve implementation and could be reduced with, for example, the use of Karatsuba's algorithm; this is just one example of how the implementation can affect the efficiency.

To demonstrate the number of bitwise operations required for each scheme and the effect of different implementations, we fix some parameters and assume multiplication is implemented naïvely, at first, then using Karatsuba's algorithm [57]. We assume no precomputation is allowed and choose the parameters to be $n = 16, t = 10$ and $\lambda = 128$ bits.

Assuming a naïve implementation of multiplication, where multiplication of two λ bit strings requires $2\lambda^2$ bitwise operations, we can calculate the number of operations required for the modified HP scheme and the schemes proposed by Shamir and Kurihara *et al.*

The modified HP share algorithm would require

$$\begin{aligned}
&\left\lceil \frac{\lambda}{t} \right\rceil [t((t-1)(n-t)+(t-1))] + 2\left(\left\lceil \frac{\lambda}{t} \right\rceil\right)^2 [t^2(n-t)] \\
&= 13[(10((9\times 6)+9))] + (2\times 169)(100\times 6) \\
&= (13\times 10\times 63) + (2\times 169\times 600) \\
&= 210990
\end{aligned}$$

bitwise operations. Shamir's share algorithm would require

$$\begin{aligned}
&\lambda[n(t-1)] + 2\lambda^2[n(t-1)] \\
&= (128\times 16\times 9) + (2\times 128^2\times 16\times 9) \\
&= 18432 + 4718592 \\
&= 4737024
\end{aligned}$$

operations. Finally, as the share algorithm by Kurihara *et al.* requires no multiplication, we can let $p = 17$ and $d = 8$ and compute the number of bitwise operations required in their share algorithm as:

$$\begin{aligned}
&d(n(pt-p-t)+1) \\
&= 8(16(170-17-10)+1) \\
&= 8((16\times 143)+1) \\
&= 8\times 2289 \\
&= 18312.
\end{aligned}$$

Now, if we use a more efficient implementation that utilises Karatsuba's algorithm [57], multiplication of two λ bit strings requires $2\lambda^{\log_2(3)}$ bitwise operations, rather than $2\lambda^2$. The number of operations required for the modified HP share algorithm is reduced to 78990, whilst the number of operations in Shamir's share algorithm is reduced to 648288. The number of operations in the scheme by Kurihara *et al.* remains the same, as no multiplications are required.

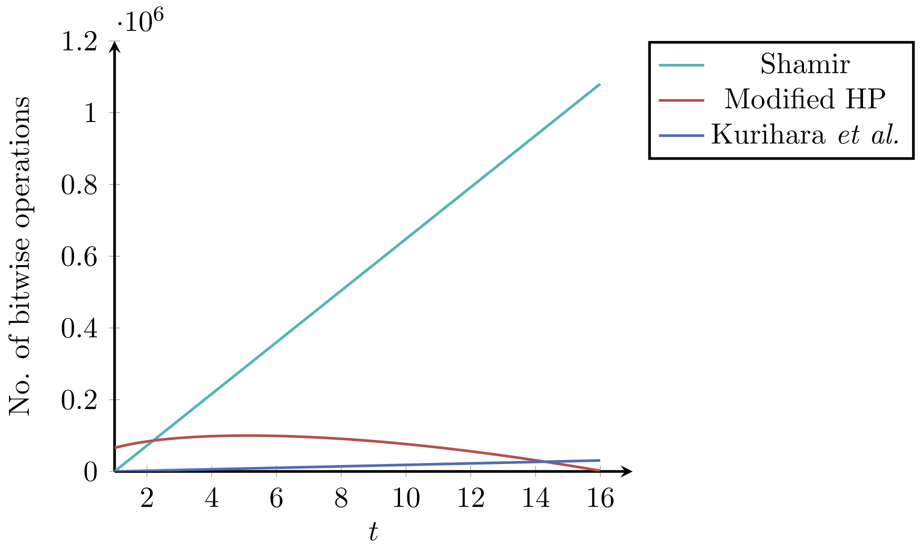

Figure 3.2: The number of bitwise operations required for the three share algorithms, given parameters $\lambda = 128$, $n = 16$ and for varying $1 \leq t \leq n$.

Figure 3.2 shows the number of bitwise operations required for the three share algorithms for the parameters $\lambda = 128$, $n = 16$ and for varying values of $1 \leq t \leq n$, assuming Karatsuba's implementation of multiplication. It is obvious to see that Shamir's share algorithm requires the greatest number of operations, whilst the share algorithm by Kurihara *et al.* requires the least. However, it is interesting to note that, if either t is very small or very large (comparative to n), the number of bitwise operations required in the modified HP share algorithm is close to that in the share algorithm by Kurihara *et al.* In particular, if $t = n$, the modified HP scheme requires the fewest number of bitwise operations. This is because, if $t = n$, the scheme does not require the use of an IDA and essentially generates $t-1$ strings as the first $t-1$ shares, then calculates the final share as the sum of all others.

We briefly note that the hardware used will affect the efficiencies. Some hardware can compute one multiplication in one clock cycle. One example of such a processer is ARM's Cortex M-Series, which is able to compute the multiplication of 16 bit strings in one clock cycle [4]. If such a processor is used, both the XOR of two 16 bit strings and the multiplication of two 16 bit strings could be be computed in one clock cycle. Assuming the same parameters as before ($n = 16, t = 10, \lambda = 128, p = 17, d = 8$), and assuming computations are done on strings of $\lambda/t = 13$ bit, the modified HP share algorithm would be computable in 1230 clock cycles, whilst the share algorithm by Kurihara *et al.* would be computable in 1408 clock cycles.

With pre-computation. Now, let us consider the effect allowing pre-computation has.

From Table 3.4, we can see that the share algorithms by Shamir and Kurihara *et al.* are reduced in complexity with respect to the number of operations required after s is input. The modified HP share algorithm is able to reduce the complexity by a factor of t, whilst Wang and Desmedt's algorithm maintains the same complexity.

If we consider the number of bitwise operations required more closely, we can see that, in fact, if λ and n are fixed, both share algorithms by Shamir and Kurihara *et al.* require a constant number of operations for any $1 \leq t \leq n$. In contrast, the number of operations required in the modified HP share algorithm after s is submitted is dependent on t and is always greater than either algorithm by Shamir or Kurihara *et al.*, except for, again, when $t = n$. In the case where $t = n$ the modified HP share algorithm requires fewer bitwise operations than either algorithm by Shamir or Kurihara *et al.*

3.3.5.2 Comparing recover algorithms

Without pre-computation. We can compare the complexities of the recover algorithms, given in Table 3.4, assuming pre-computation is not possible. It is immediately obvious that the complexity of the modified HP recover algorithm is lower than the complexities of both Shamir and Wang and Desmedt's schemes, given that t is necessarily equal to, or less than, n. However, it is not immediately obvious whether the modified HP recover algorithm is more efficient than the recover algorithm by Kurihara *et al.* This depends on the choice of λ, n and t.

As before, if we consider implementing multiplication using Karatsuba's algorithm, we can consider the number of bitwise operations required. For now, we include the computations required to invert the $t \times t$ matrix D' for the IDA in the modified HP recover algorithm and the $MAT(\cdot)$ function in the recover algorithm by Kurihara *et al.*

The modified HP recover algorithm requires

$$\begin{aligned}
&\frac{\lambda}{t}\left(t(t^2-1)+\frac{t(t-1)(2t+5)}{6}\right)+2\left(\frac{\lambda}{t}\right)^{\log_2(3)}\left(t^3+\frac{t(t^2+3t-1)}{3}\right)\\
&\quad=13\times\left((10\times 99)+\frac{(10\times 9\times 25)}{6}\right)+(2\times 59)\left(1000+\frac{1000+300-10}{3}\right)\\
&\quad=13\times(990+375)+118\times(1000+430)\\
&\quad=185315
\end{aligned}$$

bitwise operations, whilst Shamir's recover algorithm requires

$$\begin{aligned}
&\lambda(t^2-1)+2\lambda^{\log_2(3)}t(t-1)\\
&\quad=128(100-1)+(2\times 2187)(10\times 9)\\
&\quad=406332
\end{aligned}$$

bitwise operations. If we assume the forward and backward substitution in the recover algorithm by Kurihara *et al.* requires about $2/3(n^3p^3)$ bitwise operations, their recover algorithm requires a total of:

$$\begin{aligned}
&d(tp-t-1)(p-1)+t(p-1)^2+\frac{2}{3}t^3p^3\\
&\quad=8(170-10-1)(16)+10(16)^2+\frac{2}{3}(10^3\times 17^3)\\
&\quad=(8\times 159\times 16)+2560+\left(\frac{2}{3}\times 1000\times 4913\right)\\
&\quad=20351+2560+3275333\\
&\quad=3298244
\end{aligned}$$

bitwise operations.

The number of bitwise operations required for the parameters $n = 16$, $\lambda = 128$, and for varying values of $1 \leq t \leq n$ for the three recover algorithms, are shown in Figure 3.3, assuming Karatsuba's implementation of multiplication.

As we can see, the modified HP recover algorithm requires fewer bitwise operations than both the recover algorithms by Kurihara *et al.* and Shamir. From Figure 3.3, we can see that the recover algorithm by Kurihara *et al.* requires many more bitwise operations. This is mainly due to the forward and backward substitution required in the $MAT(\cdot)$ function

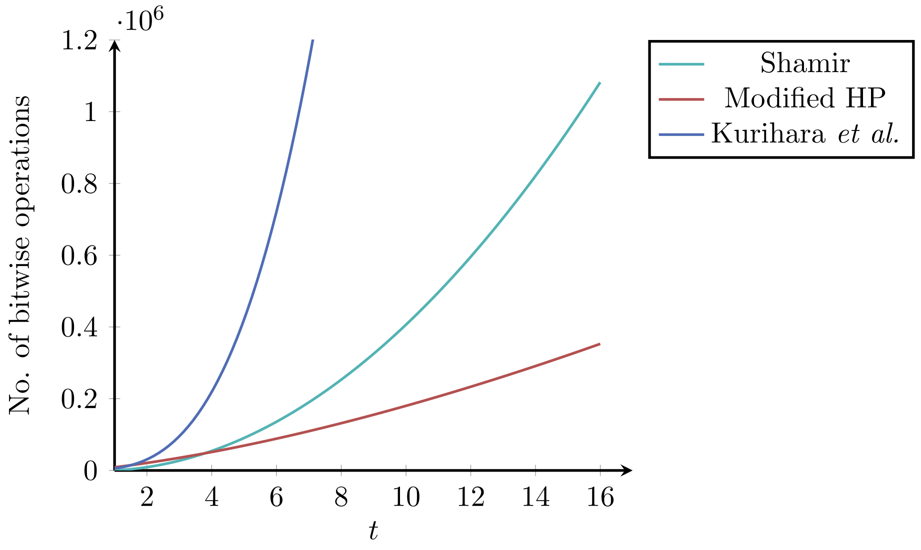

Figure 3.3: The number of bitwise operations required for the recover algorithms, given parameters $\lambda = 128$, $n = 16$ and for varying $1 \leq t \leq n$.

of their recover algorithm.

With pre-computation. As previously said, Kurihara *et al.* claim the forward and backward substitution required in the $MAT(\cdot)$ function of their recover algorithm can be pre-computed. As we have already said, $MAT(\cdot)$ relies on the identities of the players submitting shares, so whether it can be pre-computed depends on either knowledge of the players that will submit shares (which would be unusual), or on the parameters t and n (if $\binom{n}{t}$ is small enough it may be worthwhile computing). If pre-computation is possible, continuing from the same parameters as before, the modified HP recover algorithm requires 130672 bitwise operations, whilst Shamir's algorithm requires the same number of operations (as no pre-computation is possible with Shamir's recover algorithm), whilst the scheme by Kurihara *et al.* would require only 22912 bitwise operations. Therefore, if pre-computation in the recover algorithm is possible, the algorithm by Kurihara *et al.* requires, by far, the least number of bitwise operations.

3.3.5.3 Summary

To summarise, here are the conclusions we draw from the analysis.

- The modified HP share algorithm requires fewer bitwise operations than Shamir's share algorithm for all $2 \leq t \leq n$.

- For $2 \leq t < n$, the modified HP share algorithm requires more bitwise computations than the share algorithm proposed by Kurihara *et al.* How many more depends on a number of parameters, including the implementation of multiplication of bit strings and the hardware used.
- For $2 \leq t < n$, and assuming pre-computation is possible, the modified HP share algorithm requires more bitwise operations to be computed after s is submitted than both share algorithms by Shamir and Kurihara *et al.*
- When $t = n$, the modified HP share algorithm requires fewer bitwise XORs than both share algorithms by Shamir and Kurihara *et al.*, with or without pre-computation.
- The modified HP recover algorithm is faster than both recover algorithms proposed by Shamir and Kurihara *et al.*, assuming no pre-computation is possible.
- If pre-computation of the $MAT(\cdot)$ function in the recover algorithm by Kurihara *et al.* is possible, the recover algorithm by Kurihara *et al.* is the most efficient.

The modified HP scheme finds a compromise between the efficiencies of the share and recover algorithms. Even though the share algorithm is not the most efficient, it requires fewer bitwise computations than Shamir's share algorithm and manages to achieve an efficient recover algorithm simultaneously.

Chapter 4

An efficient, computationally secure threshold scheme

Contents

This chapter is based on joint work with Liqun Chen and Keith Martin [26].

4.1 Introduction

Computationally secure threshold schemes, which were introduced in Definition 2.3.14, relax the security requirements of perfect threshold schemes and, in doing so, are able to achieve smaller share sizes. The computationally secure threshold scheme goals were first mentioned by Karnin *et al.* [58], but it was in Krawczyk's work in 1994 that the first computationally secure threshold scheme, HK0, defined in Figure 2.3, was proposed [58].

Since Krawczyk's work, computationally secure threshold schemes have found many applications, but there were no new constructions proposed until 2010, when Resch and Plank proposed a computationally secure (t, n)-threshold scheme in the form of a dispersed storage system. The scheme is called AONT-RS [85] and blends an all-or-nothing transform (AONT) [86] with Reed-Solomon (RS) coding [84]. The scheme aims to achieve computational security whilst being more efficient than other schemes, including Krawczyk's HK0 scheme.

AONT-RS is a feature in the object storage system sold by Cleversafe, a company recently acquired by IBM [48], who renamed the product to IBM Cloud Object Storage. In 2016, the system was rated the overall leader in the Gartner critical capabilities for object storage report [23].

Going back to 1994, after considering computationally secure threshold schemes, Krawczyk extended HK0 and proposed goals for a robust, computationally secure threshold scheme, defined in Definition 2.3.15, along with a candidate solution, called HK1 [58]. Prior to Krawczyk's work, robustness had only be studied in the information theoretic setting in [73] and [98]. Krawczyk's work was revisited in 2007 by Bellare and Rogaway [8], in which they proposed formal definitions for a computationally secure threshold scheme and proved Krawczyk's robust HK1 scheme to be secure, but only after assuming the hash function is indistinguishable from a random oracle. They then proposed a refined version of Krawczyk's scheme, called HK2, which achieves the robust computationally secure goals under standard assumptions. The HK1 and HK2 schemes are introduced in Section 4.4.1.

In this chapter, we revisit the AONT-RS scheme presented in [85]. We present the original scheme, then define a generalised version, called AONT-RS0, and clearly state each previously undefined assumption. We then formally prove AONT-RS0 achieves computational security, assuming the hash function is indistinguishable from a random oracle. Previously, no thorough security analysis had been conducted on AONT-RS. We then compare AONT-RS0 to Krawczyk's HK0, considering the security, share size and efficiency of both the share and recover algorithms. We show AONT-RS0 is more efficient than HK0, but is considered less secure due to assumptions made in the security proof. Thus, AONT-RS0 achieves a more efficient scheme by compromising, but still maintaining a sufficient level of, security.

We then discuss two methods of extending AONT-RS0 to be robust. The two techniques discussed, which utilise hash functions and commitment schemes, are used in HK1 and HK2 respectively to extend HK0 to be robust. In both cases, we discuss the resulting security and compare the two techniques.

4.2 Preliminaries

Here, we introduce block cipher modes of operations for symmetric-key encryption schemes and cryptographic hash functions. Both will be used when defining AONT-RS.

4.2.1 Block cipher modes of operation

Recall from Section 2.2 a symmetric-key encryption scheme can either be a stream cipher, in which the plaintext is processed one bit at a time, or a block cipher, which operates on blocks of bits. The number of bits in a block is generally a number fixed by the encryption scheme. An example of a block cipher is the AES algorithm [30], which has a block size of 128 bits.

Block ciphers can be used in different modes of operation [74]. These modes are operational rules for a generic block cipher that each result in different properties being achieved when they are applied to a plaintext consisting of more than one block. Each mode of operation supplies the encryption scheme with additional properties and the mode used will depend on the application and desired properties. Here, we introduce three modes of operation: Electronic Code Book (ECB) mode, Cipher Feedback (CFB) and Counter (CTR) mode.

Other examples of modes of operation include *Cipher Block Chaining* (CBC) mode and *Cipher Feedback* (CFB) mode. All these modes of operation are discussed in NIST Special Publication 800-38A [74] and ISO/IEC 10116 [50].

4.2.1.1 Electronic Code Book (ECB) mode

Electronic Code Book (ECB) mode is the intuitive way that a block cipher would be implemented and works as follows. The plaintext M is split into blocks of a defined size. These are labelled $M_0, \ldots, M_{b-1}$. The first block M_0 is encrypted under the symmetric key k, resulting in the ciphertext block C_0. The second plaintext block M_1 is then independently encrypted, also under k, to get C_1. This continues block by block, until all the blocks have been encrypted. ECB encryption is illustrated in Figure 4.1. The encryption process can be expressed as:

$$C_i = Enc_k(M_i).$$

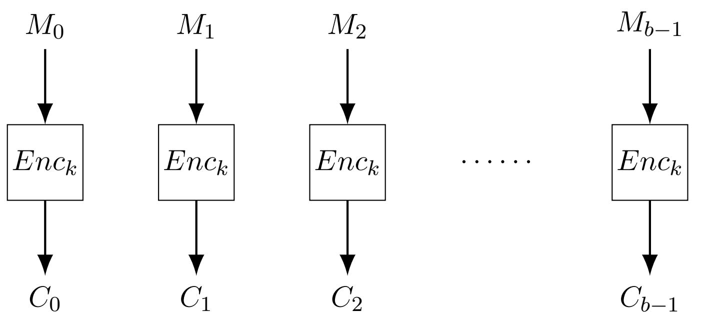

Figure 4.1: Encryption using ECB mode

Decryption in ECB is also intuitive: block by block, each ciphertext is decrypted under the symmetric key k and the plaintext blocks are recovered. The decryption process can be expressed as:

$$M_i = Dec_k(C_i).$$

ECB mode is rarely used in practice, because it has a number of drawbacks. Importantly, if we encrypt the same plaintext block twice in ECB mode, it will always result in the same ciphertext block. This makes the scheme vulnerable to statistical analysis, especially if the number of potential plaintexts is small, and to dictionary attacks, where an adversary compiles a dictionary of known plaintext and ciphertext pairs generated under a given key and compares received ciphertexts with the list of ciphertexts in the dictionary.

Alternate modes of encryption are able to combat this weakness.

4.2.1.2 Cipher Feedback (CFB) mode

Cipher Feedback (CFB) involves the use of an initialisation vector (IV) which must be known to both the sender and receiver. The IV is encrypted under the symmetric key k, which is XORed with the first block of the plaintext, M_0. This results in the first ciphertext block, C_0. The ciphertext block C_0 is then 'fed back' and used as input to the encryption algorithm, which is then XORed with the second block of plaintext, M_1, to give the second ciphertext C_1. This continues until the final ciphertext, M_{b-1} is XORed with $Enc_k(C_{b-2})$ to output the final ciphertext, C_{b-1}.

Encryption using CFB mode is illustrated in Figure 4.2, and can be expressed as:

$$C_i = M_i \oplus Enc_k(C_{i-1}).$$

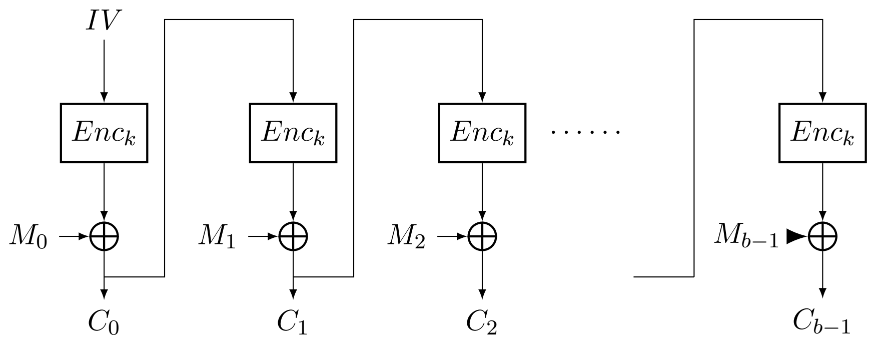

Figure 4.2: Encryption using CFB mode

Decryption in CFB mode is very similar to encryption. To begin, the IV is encrypted and the result of this is XORed with the first ciphertext, C_0, to give the first plaintext, M_0. Then, encrypt C_0 and XOR the result of this with C_1. This will recover M_1. Continue until M_{b-1} is recovered. This process is summarised as

$$M_i = C_i \oplus Enc_k(C_{i-1}).$$

CFB mode offers a number of advantages over ECB mode. It incorporates *message dependency*, meaning each ciphertext block depends on the current plaintext block and previous ciphertext blocks, which in turn rely on previous plaintext blocks. This helps to overcome the weaknesses of ECB. One advantage CFB gains in doing this is that identical plaintext blocks do not encrypt to the same ciphertext block. This is because the previous ciphertext block influences the next block, rather than each ciphertext block being independently influenced by only the plaintext message and the symmetric key.

In CFB mode, if there is an error in the transmission of a ciphertext block C_i, then only plaintext blocks M_i and M_{i+1} will be affected. This is more than would be affected in ECB mode, where if C_i is corrupted only M_i would be affected, but is still fairly limited.

4.2.1.3 Counter (CTR) mode

Counter (CTR) mode is the final block cipher mode we introduce. CTR mode is a counter-based version of CFB mode without the feedback. As in CFB mode, a shared IV is encrypted and the output is XORed with the first plaintext, M_0 to calculate the ciphertext C_0. Then, rather than C_0 being fed back into the encryption algorithm, $IV + 1$ is input. Encryption using CFB mode is illustrated in Figure 4.3 and can be summarised as

$$C_i = M_i \oplus Enc_k(IV + i),$$

whilst decryption is

$$M_i = C_i \oplus Enc_k(IV + i).$$

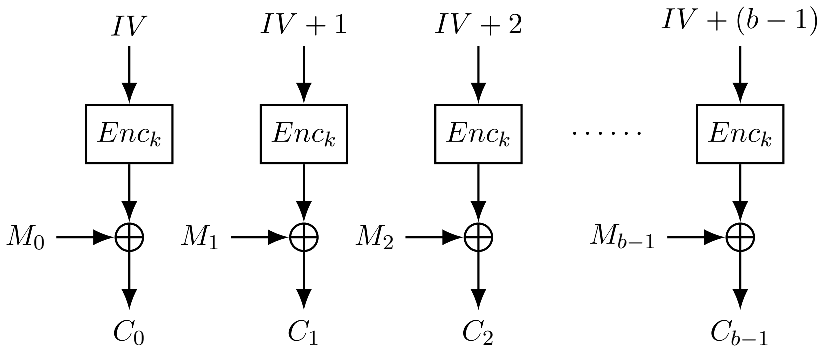

Figure 4.3: Encryption using CTR mode

As with CFB mode, CTR mode has the advantage that two identical plaintext blocks will be encrypted to distinct ciphertext blocks. An additional advantage of CTR mode is that if one error occurs in a ciphertext block, only the corresponding bits in the plaintext blocks will be affected; because of this, we say CTR mode has no *error propagation*. This is in contrast to both ECB and CFB mode. If an error occurs in ECB mode, the corresponding plaintext block is corrupted, whilst in CFB mode two blocks are corrupted.

CTR mode does not have message dependency, but it does have positional dependency since the ciphertext block depends on the position of the current plaintext block within the message. Because of this, an obvious disadvantage of CTR is that a synchronous counter is mandatory.

4.2.2 Cryptographic hash functions

Hash functions are a cryptographic primitive commonly used as a one way function. A hash function is a mathematical function, $Hash(\cdot)$, that compresses inputs into a fixed length output: they take as input some numerical value of arbitrary length and output a numerical value of fixed length, called the *digest*. Hash functions do not have a key and are publicly computable, meaning that anyone (including adversaries) can compute the hash of any value. Hash functions should also be 'easy' to compute. That is, the digest of an input should be computable in polynomial time.

We require hash functions to have following three security properties.

- *Pre-image resistance.* It should be hard (in terms of efficiency and speed) to reverse a hash function. That is, given a digest h, it should be a difficult operation to find any input value x such that $Hash(x) = h$.
- *Second pre-image resistance.* Given an input x and its digest h, it should be hard to find a distinct input $x' \neq x$ such that $Hash(x') = h$.
- *Collision resistance.* It should be hard to find two distinct inputs that produce the same digest. That is, it is hard to find two values, x and x', such that $Hash(x) = Hash(x')$.

Intuitively, pre-image resistance protects against an adversary who has only a digest and is trying to reverse the function and learn the input. Whereas second pre-image resistance protects against an adversary who has an input value and its digest, and wants to find a different input value that results in the same hash.

Hash functions cannot be collision free, as the domain of the function is much larger than the range and thus collisions are inevitable. In order for the hash function to be collision resistant, we require that these existing collisions are hard to find.

Examples of hash functions include the SHA-2 and SHA-3 families [37] and Whirlpool [51].

4.3 The AONT-RS scheme

In [85], Plank and Resch describe a new dispersal scheme, which is equivalent to a computationally secure threshold scheme. They call their construction AONT-RS as it blends an all-or-nothing transform (AONT) [86] with Reed-Solomon (RS) coding [84].

In this section, we introduce AONT-RS as defined in [85]. We then discuss generalising the scheme and state all necessary assumptions on the cryptographic primitives involved.

4.3.1 The original AONT-RS definition

Let $\mathcal{E}$ be a symmetric-key encryption scheme, as in Definition 2.2.1, with algorithms $KeyGen$, Enc and Dec and key space $\mathcal{K} = \{0,1\}^\lambda$. Let $Hash$ be a cryptographic hash function, as in Section 4.2.2, that outputs a digest of length λ. Finally, assume $Share^{RIDA}$ and $Recover^{RIDA}$ are the algorithms for the efficient version of Rabin's IDA, discussed in Section 2.4.3.2.

In [85], they do not explicitly define what security properties $\mathcal{E}$ or $Hash$ must have. Because of this, we present the AONT-RS construction and then discuss the required assumptions when considering the scheme's generalisation, which we call AONT-RS0.

The share and recover algorithms of AONT-RS, as presented in [85], are given in Figure 4.5.

The share algorithm inputs the plaintext message M. The algorithm generates a sym-

Algorithm 4.1: *Share*(M).

```
1 $k \xleftarrow{\$} KeyGen(\{0,1\}^\lambda)$;
2 $(M_0||\ldots||M_{b-1}) \leftarrow M$;
3 $M_b \xleftarrow{\$} \{0,1\}^\lambda$;
4 for $i \leftarrow 0$ to $b$ do
      $C_i \leftarrow M_i \oplus Enc_k(i)$;
5 $C \leftarrow (C_0||C_1||\ldots||C_{b-1}||C_b)$;
6 $h = Hash(C)$;
7 $c_d = h \oplus k$;
8 $\boldsymbol{V} \leftarrow Share^{RIDA}(C||c_d)$;
9 return $\boldsymbol{V}$.
```

Algorithm 4.2: *Recover*$(\boldsymbol{V})$.

```
1 $j = 0$;
2 for $i \leftarrow 0$ to $n$ do
3     if $\boldsymbol{V}[i] \neq \perp$ then
          $\boldsymbol{V}'[j] \leftarrow [j]$;
          $j = j + 1$;
4 if $j < t - 1$ then
      return $\perp$;
5 $(C||c_d) \leftarrow$
  $Recover^{RIDA}(\boldsymbol{V}'[0], \ldots, \boldsymbol{V}'[t-1])$;
6 $h = Hash(C)$;
7 $k = h \oplus c_d$;
8 for $i \leftarrow 0$ to $b$ do
      $M'_i = C_i \oplus Enc_k(i)$
9 if $M'_b \neq M_b$ then
      return $\perp$;
10 $M \leftarrow (M'_0||\ldots||M'_{b-1})$;
11 return $M$.
```

Figure 4.4: The share and recover algorithms defining AONT-RS, as proposed in [85].

metric key k using the *KeyGen* algorithm of $\mathcal{E}$. The plaintext M is then parsed into b blocks, $M_0, \ldots, M_{b-1}$, where the size of each block is defined by $\mathcal{E}$ and the number of blocks is dependent on the size of M. A new, publicly known block M_b, called the *canary*, is then randomly generated. The scheme then encrypts all plaintext blocks in what we recognise to be CTR mode, as defined in Section 4.2.1.3. Following this, the ciphertext blocks are concatenated to give the full ciphertext C. This ciphertext is then input to the hash function, which outputs the digest, h, which is then XORed with the key k to give what we call the *difference value*, c_d. Finally, the ciphertext C and the difference value c_d are concatenated and distributed via the efficient version of Rabin's IDA, $Share^{RIDA}$. This returns an n-dimensional vector $\boldsymbol{V}$. Each player P_i, for $1 \leq i \leq n$, is given the i^{th} element of $\boldsymbol{V}$, $\boldsymbol{V}[i]$, as their share.

The recover algorithm takes as input an n-vector $\boldsymbol{V}$, where the i^{th} element of this vector, $\boldsymbol{V}[i]$ is either the purported share of player P_i, $\boldsymbol{V}[i] \in \{0,1\}^*$, or the value $\Diamond$, indicating a missing share. A new t-vector $\boldsymbol{V}'$ is then created from the non empty elements of the vector $\boldsymbol{V}$. If there are fewer than t non-empty elements, the algorithm halts. Otherwise, $\boldsymbol{V}'$ is input to $Recover^{RIDA}$, which returns $(C||c_d)$. These values are then parsed and the ciphertext C is input to the hash function in order to calculate the digest h. Now that h and c_d are known, they can be XORed in order to recover the key k. The ciphertext C is then separated into the $b+1$ ciphertext blocks and, using the key k, the plaintext blocks $M_0, \ldots, M_b$ are recovered via decryption using CTR mode. The final recovered plaintext, M_b, is compared to the canary; if they are distinct, the algorithm terminates. Else, if they are equal, the plaintext blocks are concatenated and M is output.

4.3.2 Generalising AONT-RS to AONT-RS0

In [85] the authors do not define any restrictions on the size of the ciphertext or any assumptions on the hash function $Hash$ or the symmetric-key encryption scheme $\mathcal{E}$. Also, both the encryption mode of operation for $\mathcal{E}$ and the construction of the IDA are specified.

Here, we show that small ciphertexts can lead to information leakage and discuss what size the ciphertext must be to prevent this. We specify required assumptions on the symmetric-key encryption scheme and the hash function. We then generalise AONT-RS for other encryption modes and IDAs.

For now, we ignore the concept of the canary M_b. This is because the canary does not contribute to either the privacy or the recoverability of the scheme. Instead, the canary extends the scheme to have integrity. Because of this, we delay further discussion of the canary until Section 4.4.

4.3.2.1 The ciphertext size

Resch and Plank claim their system is secure because $t-1$ players are unable to recover all of $(C||c_d)$ due to the security of the IDA.

More specifically, without enough shares, fewer than t players cannot learn both C (which enables them to learn $h = Hash(C)$) and c_d fully, and so cannot learn k or M fully. Learning either C or c_d in isolation does not help the adversary; full knowledge of both C and c_d are required in order to learn M.

The authors assume an unauthorised set of players can learn at most:

- some or all of c_d and some (but not all) of C, or

- none of c_d and all of C,

and therefore, they say the scheme is secure.

We show here that this is not true. When C is a short ciphertext (in relation to the security parameter λ and the threshold value t), the adversary may be able to learn enough to learn some information about k. We show that, with a small ciphertext, an adversary can learn either

- C completely and partial information about c_d, which leaks information about k, or
- all of c_d and some of C, allowing them to guess the remainder of C.

Learning C completely and c_d partially. Consider the following example. Let $C, k \in \{0,1\}^{128}$. Let there be $n = 5$ players $P_1, \dots, P_5$ and let $t = 4$. The string $C||c_d$ would be parsed into four words to make the t-dimensional vector $\boldsymbol{M}$, where each element of $\boldsymbol{M}$ is 64 bits. Let c_0 and c_1 be the two elements that comprise C and let $c_{d,0}$ and $c_{d,1}$ be the two halves of c_d, each 64 bits. The vector $\boldsymbol{M}$ is then multiplied on the left by the generator matrix, which gives

$$G \cdot \boldsymbol{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ G_{4,0} & G_{4,1} & G_{4,2} & G_{4,3} \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ c_{d,0} \\ c_{d,1} \end{pmatrix} = \begin{pmatrix} C_0 \\ C_1 \\ c_{d,0} \\ c_{d,1} \\ x \end{pmatrix},$$

where $G_{i,0}$, for $i = \{0, \dots, 3\}$ are chosen such that any four rows of G are linearly independent and $x = G_{4,0} \cdot C_0 + G_{4,1} \cdot C_1 + G_{4,2} \cdot c_{d,0} + G_{4,3} \cdot c_{d,1}$.

Players P_1, P_2 and P_3 are an unauthorised set, yet they could learn all of C and $c_{d,0}$. They could then compute $Hash(C) = h$ and XOR the first half of h with $c_{d,0}$ to recover the first half of k. This reduces the security from 128 to 64 bits.

This attack can be prevented if c_d is contained entirely in one share. So if $c_d \in \{0,1\}^\lambda$, then C should be such that $C \in \{0,1\}^\omega$, where $\omega \geq (t-1)\lambda$.

Learning C partially and c_d completely. An alternative version of this attack utilises the fact that the hash function H is deterministic.

Consider the following example. Assume an adversary knows all of c_d and all but one bit of C. They can construct two possibilities for C, one where they guess the unknown bit is a '0', which we label C_0, and the other where the unknown bit is a '1', labelled C_1. The adversary then computes the corresponding hashes, $h_0 = Hash(C_0)$ and $h_1 = Hash(C_1)$, then computes two key candidates $k_0 = c_d \oplus h_0$ and $k_1 = c_d \oplus h_1$ and decrypts both ciphertexts C_0 and C_1 with their corresponding candidate keys to reveal two plaintext messages M_0 and M_1. From these, the adversary can guess which plaintext message is likely to be the true message and has thus learnt k completely.

In general, if the adversary knows c_d and all but j bits of C, if $j < \lambda$ this attack is quicker than a brute force approach, in which the adversary tries every possible key. In order to prevent this attack, we must ensure an adversary is unable to learn at least λ bits of C if c_d is known. This is assured if each element of $\boldsymbol{M}$, is at least λ bits, so $\boldsymbol{M}[i] \in \{0,1\}^\lambda$, meaning that C must be such that $C \in \{0,1\}^\omega, \omega \geq (t-1)\lambda$.

In summary, both attacks can be prevented if $C \in \{0,1\}^\omega$, where $\omega \geq (t-1)\lambda$. Thus any plaintext should be encrypted to give a ciphertext $C \in \{0,1\}^\omega$: if we assume the encryption scheme is *length preserving* and does not have a large *ciphertext expansion*, meaning C is not much larger than M, then we can assume that M is not much smaller. If M encrypts to be a ciphertext C that is smaller than this, C should be extended via padding.

It is worth highlighting that a computationally secure threshold scheme, such as AONT-RS, will be used in practice to distribute large ciphertexts, given the lower bound on share sizes in perfect threshold schemes. Thus it is unlikely the ciphertext will be smaller than $\lambda(t-1)$ bits. However, a threshold scheme should be able to distribute files of any size, so this may be one disadvantage of AONT-RS.

4.3.2.2 Assumptions on the encryption scheme

We observe the symmetric-key encryption scheme $\mathcal{E}$ need not operate in CTR mode, but must have ind-1 and key-1 security.

To demonstrate why ind-1 security is necessary, consider an adversary with a non-negligible ind-1 advantage who is playing both the privacy game, *Priv*, defined in Game 2.3, against a privacy challenger (this game defines the security of AONT-RS), and the indistinguishability game, *Ind*, defined in Game 2.1, against an ind-1 challenger (this defines the security of $\mathcal{E}$). For simplicity, assume the IDA is systematic, meaning that corrupted shares will directly reveal segments of the ciphertext, and that the privacy challenger shares their generated key with the ind-1 challenger. The adversary submits two messages, M_0 and M_1, to both the privacy challenger and the ind-1 challenger. The ind-1 challenger would return a ciphertext C_b corresponding to one of the two messages; the adversary has a non-negligible advantage at guessing which was chosen (as we have assumed $\mathcal{E}$ does not have ind-1 security). Assume the adversary then uses this advantage to guess which plaintext was encrypted by the privacy challenger. They corrupt up to $t-1$ shares during the corrupt procedure of the privacy game against AONT-RS, then, from the information learnt, the adversary can determine whether or not the privacy challenger is revealing parts of the known ciphertext, or a different ciphertext. If the shares match the ciphertext received from the ind-1 challenger, the adversary guesses the same value b was chosen. Else, the adversary guesses a different b. Thus ind-1 security for $\mathcal{E}$ is necessary.

In order to show why key-1 security is also necessary for the symmetric-key encryption scheme, consider a similar scenario in which an adversary with a non-negligible advantage against a key-1 challenger plays both the privacy game against AONT-RS and the key-1 game against $\mathcal{E}$. The adversary submits two messages M_0 and M_1 to the privacy challenger, who randomly chooses which message to distribute. The privacy challenger then shares the key k with the key-1 challenger, to whom the adversary submits one of the two messages, say M_0 (without loss of generality). The key-1 challenger then returns the ciphertext, C_0, and the adversary, who it is assumed has a non-negligible advantage, guesses k. The adversary can then run the corrupt procedure with the privacy adversary, who will return elements of the ciphertext. If the shares match the ciphertext returned by the key-1 challenger, the adversary guesses $b = 0$. Otherwise, the adversary guesses $b = 1$.

Thus, the symmetric-key encryption scheme must have both ind-1 and key-1 security. As a side note, we refer to the discussion in Section 2.2.3 where we mentioned that in encryption schemes with a large ratio between message length and key length, it may be the case that ind-1 security implies key-1 security, but that neither a counterexample nor a proof of this has yet been found. We highlight that, given the condition on the size of the ciphertext, that is $C = \{0, 1\}^{\omega}$, where $\omega \geq (t-1)\lambda$ and $k \in \{0, 1\}^{\lambda}$, then, if the encryption scheme does not have a large message expansion, the ratio between the size of the plaintext and the size of the key length is likely to be large. So, in this encryption scheme, it may be true that ind-1 security implies key-1 security, but this is not proven.

We make one further comment on the security of the symmetric-key encryption scheme, $\mathcal{E}$. Ind-1 security, and not general indistinguishability, is required for the $\mathcal{E}$ because each time a new M is submitted, a new key k is generated. So, each key is only used to encrypt one plaintext and thus ind-1 security is sufficient. Similarly, key-1 security and not general key-unrecoverability is required. It is noted that, if AONT-RS were to maintain a key and use it to distribute multiple plaintexts, rather than generating a new key after each submission, the security requirements on the encryption scheme will need to be revisited. In particular, indistinguishability and key-unrecoverability will be necessary for $\mathcal{E}$. We do not, however, consider a key being used multiple times and instead assume a new key is generated with each submission, as suggested in [85].

Finally, we discuss the use of CTR mode. If we assume the encryption scheme is implemented in ECB mode, two submissions of the same plaintext to AONT-RS will be encrypted differently as each time a message is submitted, AONT-RS generates a new key. However, if two plaintext blocks within the plaintext are identical, they will be encrypted to the same ciphertext block; the use of CTR mode (or other modes) will prevent this. Therefore, which mode of operation used relies on the application and the necessary properties: if ECB mode is used, identical plaintext blocks will be encrypted to identical ciphertext blocks, but a synchronous counter is not required. If CTR mode is used, identical plaintext blocks will be encryption to distinct ciphertext blocks, but a synchronous counter is required.

4.3.2.3 Assumptions on the hash function

In order for AONT-RS to be secure, the hash function must be indistinguishable from a random oracle.

Let $\{0,1\}^*$ denote the space of finite binary strings and assume $Hash : \{0,1\}^* \to \{0,1\}^\lambda$. In order for $Hash$ to be indistinguishable from a random oracle, we must assume that each bit of the digest, $h = Hash(x)$, is chosen uniformly and independently at random. This means that the entropy of $h = Hash(x)$ is maximal, so $H(h) = 2^\lambda$. A security proof under this assumption is said to be in the *random oracle (RO) model.*

From a theoretical point of view, assuming the hash function is indistinguishable from a random oracle is a very strong assumption as no actual hash function can have this property. However, authors, such as Bellare and Rogway in [6], have argued that such a model is valuable and can capture the properties practical primitives really seem to possess and provide provable security without losing efficiency.

The idea of the random oracle model builds on work by Goldreich, Goldwasser and Micali [39, 40] and Fiat-Shamir [36]. A discussion of instantiating random oracles with primitives, such as a hash function, is given in [6].

4.3.2.4 Requirements of the IDA

In the AONT-RS definition, it is assumed the internal IDA is the efficient version of Rabin's IDA, described in Section 2.4.3.2. We wish to generalise this so other IDAs can be used.

In particular, any IDA that is equivalent to, or has stronger security properties than, a linear $(0, t; n)$-ramp scheme can be used; the efficient version of Rabin's IDA meets this security requirement. This security requirement is necessary as an unauthorised coalition of at most $t-1$ players must learn no information about at least λ elements of the input to the IDA, $C||c_d$. As previously shown, if the unauthorised coalition learns more than this, they will learn information about the key k or the plaintext M. If a linear $(0, t; n)$-ramp scheme is used with an input of at least $t\lambda$ bits, it is guaranteed a coalition of $t-1$ players

Algorithm 4.3: $Share^{A0}(M)$.

```
1 $k \xleftarrow{\$} KeyGen(\{0,1\}^{\lambda})$;
2 $C \xleftarrow{\$} Enc_k(M)$;
3 $h = Hash(C)$;
4 $c_d = h \oplus k$;
5 $\boldsymbol{V} \leftarrow Share^{ECC}(C||c_d)$;
6 return $\boldsymbol{V}$.
```

Algorithm 4.4: $Recover^{A0}(\boldsymbol{V})$.

```
1 $j = 0$;
2 for $i \leftarrow 0$ to $n$ do
3     if $\boldsymbol{V}[i] \neq \perp$ then
          $\boldsymbol{V}'[j] \leftarrow [j]$;
          $j = j + 1$;
4 if $j < t - 1$ then
      return $\perp$;
5 $(C||c_d) \leftarrow$
      $Recover^{ECC}(\boldsymbol{V}'[0], \ldots, \boldsymbol{V}'[t-1])$;
6 $h = Hash(C)$;
7 $k = h \oplus c_d$;
8 $M \leftarrow Dec_k(C)$;
9 return $M$.
```

Figure 4.5: The share and recover algorithms defining AONT-RS0.

will always be missing at least λ bits.

Other information dispersal algorithms with stronger properties can also be used, such as a perfect threshold scheme, but one equivalent to a linear $(0, t; n)$-ramp scheme will achieve optimal storage and sufficient security.

Rather than using $Share^{RIDA}$ and $Recover^{IDA}$ to denote the efficient version of Rabin's IDA, we will use $Share^{ECC}$ and $Recover^{ECC}$ to denote a more general IDA with the required security properties, as discussed in Section 2.4.2.

4.3.2.5 Generalised definition of AONT-RS

We now combine the aforementioned assumptions and security requirements to re-define AONT-RS.

Fix integers $t, n \in \mathbb{N}$ such that $1 \leq t \leq n$. Assume there exists an ind-1 and key-1 secure symmetric-key encryption scheme $\mathcal{E}$ with key space $\mathcal{K} = \{0,1\}^{\lambda}$, message space $\mathcal{M} = \{0,1\}^*$ and ciphertext space $\mathcal{C} = \{0,1\}^{\omega}$, where $\omega \geq (t-1)\lambda$. Assume there exists a hash function $Hash(\cdot)$ that outputs a digest of λ bits and is indistinguishable from a random oracle. Assume there is an IDA equivalent to a linear $(0, t; n)$-ramp scheme with algorithms $Share^{ECC}$ and $Recover^{ECC}$.

```
Procedure $Initialise$   $G_0$
1   $k \xleftarrow{\$} KeyGen\{0,1\}^{\lambda}$;
2   $k' = k$;
3   $b \xleftarrow{\$} \{0,1\}$;
Procedure $Initialise$   $G_5$
1   $k \xleftarrow{\$} KeyGen\{0,1\}^{\lambda}$;
2   $k' \xleftarrow{\$} KeyGen\{0,1\}^{\lambda}$;
3   $b \xleftarrow{\$} \{0,1\}$;
Procedure $Deal(x_0, x_1)$                                   $G_0, G_5$
1   $C \leftarrow Enc_k(x_b)$;
2   $h \leftarrow Hash(C)$;
3   $h \oplus k' = c_d$;
4   $\boldsymbol{X} \leftarrow Share^{ECC}(C||c_d)$;
Procedure $Corrupt(i)$                                       $G_0, G_5$
1   $\boldsymbol{V}[i] \leftarrow (\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\dots||\boldsymbol{S}_n[i])$;
2   return $\boldsymbol{V}[i]$
Procedure $Finalise(b')$                                     $G_0, G_5$
1   if $b' = b$ then
        return true;
2   else return false;
```

Figure 4.6: Procedures used to construct games G_0 and G_5, used to prove the privacy of AONT-RS0 in Theorem 4.3.1.

From now on, we call the generalised AONT-RS scheme under the above assumptions with algorithms as in Figure 4.5, the AONT-RS0 scheme. Denote AONT-RS0 by Π_0 with algorithms $Share^{A0}$ and $Recover^{A0}$.

4.3.3 Security analysis of AONT-RS0

We now prove AONT-RS0 achieves computational privacy in the RO model.

Theorem 4.3.1 (Privacy of AONT-RS0). *Let $\mathcal{A}$ be a privacy adversary against the AONT-RS0 scheme Π_0 and let the internal hash function $Hash$ be indistinguishable from a random oracle. Let the ciphertext be $C \in \{0,1\}^{\omega}$, where $\omega \geq (t-1)\lambda$. Then there is an ind-1 adversary $\mathcal{B}$ attacking the indistinguishability of $\mathcal{E}$ such that*

$$\mathbf{Adv}_{\Pi_0}^{Priv}(\mathcal{A}) \leq \mathbf{Adv}_{\mathcal{E}}^{Ind}(\mathcal{B}),$$

where $\mathcal{B}$ makes only one query during the deal procedure of the indistinguishability game

Ind and the running time of $\mathcal{B}$ is that of $\mathcal{A}$ plus one execution of the AONT-RS0 share algorithm, $Share^{A0}$.

Proof. The proof relies on games G_0 and G_5, as in Figure 4.6, where both G_0 and G_5 consist of four procedures: *Initialise*, *Deal*, *Corrupt* and *Finalise*. We define the procedures individually and put next to the procedure the games in which it appears. Games G_0 and G_5 only differ in the initialise procedure.

The advantage of the AONT-RS0 privacy adversary $\mathcal{A}$ can be defined as

$$\mathbf{Adv}^{Priv}_{\Pi_0}(\mathcal{A}) = 2 \cdot \Pr[G_0^{\mathcal{A}}] - 1. \tag{4.3.1}$$

To justify this, we explain that G_0 is just like the game defining the privacy of a threshold scheme, as in Game 2.3, but with the AONT-RS0 share algorithm.

Game G_5 differs from G_0 only because the key k used to encrypt the message M is different to the value k' used to compute $c_d = h \oplus k'$. We claim that

$$\Pr[G_0^{\mathcal{A}}] = \Pr[G_5^{\mathcal{A}}], \tag{4.3.2}$$

as the hash function $Hash$ behaves as a random oracle. Due to the properties of the IDA and the size of the ciphertext $C = \{0,1\}^{\omega}$, where $\omega \geq \lambda(t-1)$, the adversary is always missing either at least λ bits of C or all of c_d. Thus, the adversary can learn either h or c_d. If $\mathcal{A}$ learns h, $h = c_d' \oplus k'$ for some $c_d' \neq c_d$. If $\mathcal{A}$ learns c_d, then $c_d = h' \oplus k'$ for some $h' \neq h$. Thus, (4.3.2) holds true.

Construct an adversary $\mathcal{B}$ attacking the indistinguishability of $\mathcal{E}$ such that

$$2 \cdot \Pr[G_5^{\mathcal{A}}] - 1 \leq \mathbf{Adv}^{Ind}_{\mathcal{E}}(\mathcal{B}). \tag{4.3.3}$$

Adversary $\mathcal{B}$ will run $\mathcal{A}$, who will be playing G_5, as a subroutine, and will act as their challenger. Adversary $\mathcal{B}$ will use the advantage of $\mathcal{A}$ to gain an advantage in the indistinguishability game, as follows.

Adversary $\mathcal{B}$ picks $k' \xleftarrow{\$} \{0,1\}^{\lambda}$ and runs $\mathcal{A}$. Adversary $\mathcal{A}$ submits (x_0, x_1) to $\mathcal{B}$ during the deal procedure, and $\mathcal{B}$ will query x_0, x_1 to its indistinguishability challenger and receive

$C \xleftarrow{\$} Enc_k(x_b)$ in return, where k is the key generated by the challenger. Now, $\mathcal{B}$ executes the rest of the deal procedure of G_5 using k'; so $\mathcal{B}$ computes $Hash(C) = h$, $h \oplus k' = c_d$ and $\boldsymbol{X} \leftarrow Share^{ECC}(C||c_d)$. When $\mathcal{A}$ submits $Corrupt(i)$ queries, $\mathcal{B}$ responds with $\boldsymbol{V}[i]$, where $\boldsymbol{V}[i] \leftarrow (\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\dots||\boldsymbol{S}_n[i])$. When $\mathcal{A}$ outputs a bit b' and submits this to $\mathcal{B}$, $\mathcal{B}$ passes this onto their indistinguishability challenger. The advantage of $\mathcal{B}$ is $2 \cdot \Pr[b' = b] - 1$.

By combining (4.3.1), (4.3.2) and (4.3.3), we see that

$$\mathbf{Adv}^{Priv}_{\Pi_0}(\mathcal{A}) \leq \mathbf{Adv}^{Ind}_{\mathcal{E}}(\mathcal{B}),$$

as required. □

4.3.4 Efficiency analysis of AONT-RS0 and comparison with HK0

In [85], Resch and Plank briefly compare AONT-RS to Krawczyk's HK0 [60]. They then conduct a performance comparison of AONT-RS against Rabin's IDA [80] (defined in Section 2.4.3.1) and Shamir's threshold scheme [91] (introduced in Construction 2.3.3). However, neither Rabin's IDA nor Shamir's threshold scheme have similar security goals to AONT-RS. Rabin's IDA has security equivalent to a linear $(0, t; n)$-ramp scheme and would not be used to distribute data if there were privacy concerns, whereas Shamir's threshold scheme achieves perfect security, which would not be used to share large data due to the bounds on the share sizes. The authors do not conduct a thorough comparison of AONT-RS with Krawczyk's HK0, which achieves computational security.

Here, we conduct a thorough comparison between AONT-RS0 and Krawczyk's HK0 in which we consider the security achieved, the share sizes for each scheme and the efficiency of the share and recover algorithms.

To allow for a fair comparison, we assume both AONT-RS0 and HK0 use the systematic version of Rabin's IDA, as in Section 2.4.3.2, and that HK0 uses Shamir's threshold scheme. We also assume both AONT-RS0 and HK0 use the same symmetric-key encryption scheme $\mathcal{E}$ with the same security parameter λ.

We also assume $C \in \{0,1\}^{\omega}$ with $\omega \geq \lambda(t-1)$. This is for the benefit of AONT-RS0 as, otherwise, the scheme would be vulnerable to attacks described in Section 4.3.2.1. We assume C is as described in both schemes in order to allow for a fair comparison.

It is noted that if $\omega < \lambda(t-1)$, AONT-RS0 will need to pad the message to lengthen C, whereas HK0 can distribute C as is. This is a limitation of AONT-RS0. However, as mentioned previously, a computationally secure threshold scheme is often used instead of a perfect threshold scheme when M is large. Therefore it may be reasonable to assume that $\omega \geq \lambda(t-1)$ in general.

To illustrate the kinds of parameters that may be used, we highlight an example presented in [85]. They consider a scenario with $n = 16$ players and a threshold of $t = 10$. They choose the security parameter to be $\lambda = 128$ bits. The smallest ciphertext that could be securely distributed via AONT-RS0 in this scenario is $\lambda(t-1) = 128 \times 9 = 1152$ bits. In their scenario, the authors of [85] consider distributing a 4KB block of data, which is 32000 bits of information. This is much larger than the minimum size required to guarantee security and, in many real-life scenarios, this is likely to be the case. If less than 1152 bits were to be distributed via AONT-RS0, assuming these parameters for λ, n and t, padding would be required. If less than 1152 bits were to be distributed in the HK0 scheme under the same parameters, no padding would be required.

Under these assumptions, we now compare AONT-RS0 with HK0.

4.3.4.1 Security

AONT-RS0 achieves computational privacy, assuming $\mathcal{E}$ is ind-1 and key-1 secure, that $Hash$ is indistinguishable from a RO and the IDA is equivalent to a $(0, t; n)$-linear ramp scheme.

Likewise, HK0 achieves computational privacy, also assuming $\mathcal{E}$ is ind-1 and key-1 secure and with the additional assumption of the existence of a perfect (t, n)-threshold scheme and an IDA (with no privacy requirements).

As HK0 is computationally secure under standard assumptions, whereas AONT-RS0 is only computationally secure in the RO model, HK0 is considered to be more secure. However, being secure in the RO model is considered to be sufficient for the majority of applications [6].

4.3.4.2 Share size

In AONT-RS0, each player's share consists of

$$\left\lceil \frac{\omega + \lambda}{t} \right\rceil$$

bits. This is because $(C||c_d)$, a total of $\omega + \lambda$ bits, would be input to the IDA, which would output n shares, each consisting of $\lceil \frac{\omega+\lambda}{t} \rceil$ bits.

In HK0, each player's share would consist of

$$\left\lceil \frac{\omega}{t} \right\rceil + \lambda$$

bits. This is because each player's share consists of a ciphertext share $\boldsymbol{C}[i]$ and a key share output from Shamir's threshold scheme, $\boldsymbol{K}[i]$. The ciphertext is distributed via the IDA and so each ciphertext share would consist of $\lceil \frac{\omega}{t} \rceil$ bits. Assuming the key is distributed by an ideal, perfect threshold scheme (which Shamir's is), each key share would consist of λ bits. Thus the total size of the share is the sum of the two parts.

(Alternatively, because there are no security requirements on the IDA used in HK0, replication, as in Section 2.4.1, could be used as the IDA. This would make the share size larger, as it would total $\omega + \lambda$ bits.)

We can compare the share sizes achieved and see that

$$\left\lceil \frac{\omega + \lambda}{t} \right\rceil = \left\lceil \frac{\omega}{t} + \frac{\lambda}{t} \right\rceil \leq \left\lceil \frac{\omega}{t} \right\rceil + \left\lceil \frac{\lambda}{t} \right\rceil < \left\lceil \frac{\omega}{t} \right\rceil + \lambda,$$

for all $t > 1$. Therefore AONT-RS0 achieves smaller share sizes than HK0 for all parameters λ, n and t. In particular, the ratio between the share sizes achieved is larger if t is larger. So, if t is chosen to be close to n, the ratio between the two share sizes will be larger than if t is chosen to be small (*i.e.*, if $t = 2$).

4.3.4.3 Efficiency of share algorithms

When computing the efficiency of the share algorithms for both AONT-RS0 and HK0, the computations required for the encryption algorithm will not be taken into account, as both schemes will use the same encryption algorithm to encrypt the same size message M under the same size key.

After encrypting M, the AONT-RS0 share algorithm requires one hash function computation, $Hash : \{0,1\}^{\omega} \to \{0,1\}^{\lambda}$, and one XOR of λ bits (XORing h and k to calculate c_d). Then, $(C||c_d)$ is distributed via the IDA. Assuming the efficient version of Rabin's IDA is used, this requires $(n-t)t$ multiplications and $(n-t)(t-1)$ additions in the field $GF(2^{\lceil \frac{\omega+\lambda}{t} \rceil})$.

The HK0 share algorithm, after encrypting M, requires the distribution of the key k via a perfect threshold scheme. We have assumed Shamir's threshold scheme is used, which will require $n(t-1)$ multiplications and $n(t-1)$ additions in $GF(2^{\lambda})$ to run the share algorithm, as in Section 3.3.2.2. Other perfect threshold schemes can be used; a discussion of efficient, perfect threshold schemes is held in Section 3.3. After the key is shared, the ciphertext is distributed via the IDA. As the efficient version of Rabin's IDA is used, this requires $(n-t)t$ multiplications and $(n-t)(t-1)$ additions in the field $GF(2^{\lceil \frac{\omega}{t} \rceil})$.

(Alternatively, as before, because there are no security requirements on the IDA used in HK0, replication could be used. This would require no computation).

In general, if HK0 uses an optimal IDA (such as Rabin's IDA), AONT-RS0 requires fewer bitwise XORs than HK0. If, however, HK0 uses replication rather than an optimal IDA, HK0 has a much more efficient share algorithm as (excluding encryption) the only computation required is for the sharing of k via the perfect threshold scheme. However, in achieving a more efficient share algorithm, HK0 with replication compromises considerably on the share size.

We briefly note the effect of pre-computation in these schemes. If pre-computation is possible, AONT-RS0 will be unable to benefit as nothing can be computed prior to M being submitted. On the other hand, the HK0 scheme can only conduct the pre-computation possible in the perfect threshold scheme (a discussion of which is included in Section 3.3).

4.3.4.4 Efficiency of recover algorithms

In the AONT-RS0 recover algorithm, the players submit shares of size $\lceil \frac{\omega+\lambda}{t} \rceil$. When t players submit their shares, they are input to the IDA recover algorithm. This requires t^2 multiplications and $t(t-1)$ additions in $GF(2^{\lceil \frac{\omega+\lambda}{t} \rceil})$ in order to recover $C||c_d$. Following this, one hash computation $Hash : \{0,1\}^\omega \to \{0,1\}^\lambda$ is required. Finally, one XOR of λ bits is necessary to recover k, then C is decrypted.

For the HK0 recover algorithm, the players all have shares of size $\lceil \frac{\omega}{t} \rceil + \lambda$. After t players submit their shares, the ciphertext share is separated from the key share. The IDA is then used to recover the ciphertext from the ciphertext shares. If Rabin's IDA is used, this requires t^2 multiplications and $t(t-1)$ additions in $GF(2^{\lceil \frac{\omega}{t} \rceil})$. (As before, if replication is used instead, no computation is required in this step.) Then, the key is recovered via the perfect threshold scheme recover algorithm, which, if Shamir's threshold scheme is used, requires either $t(t-1)$ multiplications and t^2-1 additions, or $\mathcal{O}(t \log^2 t)$ operations, dependent on the method of recovery. A discussion on the efficiency of the recover algorithm of Shamir's scheme, as well as other threshold schemes, is available in Section 3.3. Finally, the ciphertext is decrypted to recover M.

In general, if HK0 uses an optimal IDA (such as Rabin's IDA), AONT-RS0 requires fewer bitwise XORs than HK0. As before, if HK0 uses replication rather than an optimal IDA, HK0 has a much more efficient recover algorithm as (excluding encryption) the only computation required is the recovery of k.

4.3.4.5 Conclusion of comparison

This analysis shows that, in assuming the random oracle model, AONT-RS0 is able to achieve a more efficient scheme than HK0, in terms of memory size and efficiency of the share and recover algorithms.

However, if the number of bitwise operations is prioritised above memory, HK0 could use replication rather than Rabin's IDA. This would result in HK0 having much larger share sizes than AONT-RS0, but requiring fewer bitwise operations to run the share algorithm.

4.4 Extending AONT-RS0 to be robust

In this section, we extend AONT-RS0 to be a robust, computationally secure threshold scheme, as in Definition 2.3.15.

We begin by discussing the use of a canary in the original AONT-RS definition and show how the canary does not extend the scheme to be robust, but does provide integrity.

We then discuss two techniques that provide robustness and consider applying them to AONT-RS0. The first technique utilises hash functions, as used by Krawczyk to extend HK0 to be robust, resulting in HK1. The second relies on commitment schemes, which were used by Bellare and Rogaway to provide an alternative robust extension to HK0, called HK2 [8]. We briefly introduce both HK1 and HK2 before adopting the techniques to extend AONT-RS0 to be robust.

4.4.1 Data integrity using a canary

In the original definition of AONT-RS, the authors use a canary [85]. The canary, denoted M_b in Figure 4.5, is a publicly known value that is appended to the plaintext before encryption. When the ciphertext is decrypted, the recovered value can be compared to the publicly known value and, if they are equal, the plaintext is accepted to be correct. If they are not equal, the recover algorithm halts, concluding there has been an error.

The canary provides the user with some guarantee that the recovered plaintext is correct. That is, the canary provides *data integrity*, one of the security goals introduced in Section 2.1.2, as it provides a means of detecting whether the data has been manipulated in an unauthorised manner.

However, the canary does not extend AONT-RS to be robust, as in Definition 2.3.15, as the scheme is unable to recover the correct plaintext, even if only one submitted share is corrupt. Although the canary is able to flag an incorrect plaintext, it cannot recover the correct plaintext. It recognises data has been altered in an unauthorised manner, but is not able to recover the original plaintext in these settings.

In summary, the use of a canary extends a threshold scheme to provide data integrity, but does not extend the scheme to be robust. A canary could be used to provide integrity in AONT-RS0, as is done in AONT-RS, but it would not make the scheme robust.

The techniques we consider next to extend AONT-RS0 to be robust (using hash functions and commitment schemes) allow recovery of M even if false shares are submitted. In addition to this, they highlight which share (or shares) are corrupt and thus allow the user to take any necessary action. However, it is noted that both hash functions and commitment schemes require more computation than the use of a canary, so which technique (either a robust extension or a canary) is used depends on the application. In practice, both techniques could be combined; the canary could first be verified and, only if the canary is incorrect, will the shares be individually verified.

4.4.2 Robust extension using hash functions

After Krawczyk proposed HK0 in [60], he proposed a robust extension that utilised hash functions. Thirteen years later, Bellare and Rogaway proved this robust extension, which they call HK1, to be computationally secure and robust, both in the RO model.

We begin by briefly introducing HK1, then apply hash functions in the same manner in order to extend AONT-RS0 to be robust.

4.4.2.1 HK1

Let $\mathcal{E}$ be a symmetric-key encryption scheme with ind-1 and key-1 security. Assume there is a perfect (t, n)-threshold scheme with share and recover algorithms $Share^{PTS}$ and $Recover^{PTS}$, and a (t, n)-IDA with algorithms $Share^{IDA}$ and $Recover^{IDA}$ (there are no security requirements for this IDA). Finally, let $Hash$ be a hash function that is indistinguishable from a RO, and let $Share^{ECC}$ and $Recover^{ECC}$ be algorithms for a (t, n)-ECC (that is, an IDA equivalent to a linear $(0, t; n)$-ramp scheme).

The HK1 share and recover algorithms are denoted by $Share^{HK1}$ and $Recover^{HK1}$ and defined in Figure 4.7.

Algorithm 4.5: $Share^{HK1}(M)$.

```
1 $k \xleftarrow{\$} KeyGen(\{0,1\}^\lambda)$;
2 $C \xleftarrow{\$} Enc_k(M)$;
3 $\boldsymbol{K} \xleftarrow{\$} Share^{PTS}(k)$;
4 $\boldsymbol{C} \xleftarrow{\$} Share^{IDA}(C)$;
5 for $i \leftarrow 1$ to $n$ do
      $h_i \leftarrow Hash(\boldsymbol{K}[i]||\boldsymbol{C}[i])$;
      $\boldsymbol{S}_i \xleftarrow{\$} Share^{ECC}(h_i)$;
6 for $i \leftarrow 1$ to $n$ do
      $\boldsymbol{V}[i] \leftarrow (\boldsymbol{K}[i]||\boldsymbol{C}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i])$;
7 return $\boldsymbol{V}$.
```

Algorithm 4.6: $Recover^{HK1}(\boldsymbol{V})$.

```
1 for $i \leftarrow 1$ to $n$ do
      $(\boldsymbol{K}[i]||\boldsymbol{C}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i]) \leftarrow \boldsymbol{V}[i]$;
2 for $i \leftarrow 1$ to $n$ do
      $h_i \leftarrow Recover^{ECC}(\boldsymbol{S}_i)$;
3 for $i \leftarrow 1$ to $n$ do
      if $\boldsymbol{V}[i] \neq \Diamond$ and $Hash(\boldsymbol{K}[i]||\boldsymbol{C}[i]) \neq h_i$ then
          $\boldsymbol{K}[i] \leftarrow \Diamond$ ; $\boldsymbol{C} \leftarrow \Diamond$;
4 $k \leftarrow Recover^{PTS}(\boldsymbol{K})$;
5 $C \leftarrow Recover^{IDA}(\boldsymbol{C})$;
6 $M \leftarrow Dec_k(C)$;
7 return $M$.
```

Figure 4.7: The share and recover algorithms defining HK1, a robust, computationally secure threshold scheme by Krawczyk [60].

The HK1 share algorithm, defined in Algorithm 4.5, begins the same as the HK0 share algorithm, as in Algorithm 2.5. The HK1 share algorithm inputs the data M to be distributed, generates a λ bit key k, and encrypts M under k to calculate the ciphertext C. The key k is shared via a (t, n)-perfect threshold scheme, which returns an n-vector $\boldsymbol{K}$. The ciphertext C is shared via a (t, n)-IDA (with no security requirements), which returns an n-vector $\boldsymbol{C}$. Now, rather than giving player P_i a share consisting of a ciphertext share and a key share, as is done in HK0, HK1 hashes each pair $(\boldsymbol{K}[i]||\boldsymbol{C}[i])$, for $1 \leq i \leq n$, to calculate the digests h_i. Each digest is then distributed via a (t, n)-ECC, which outputs an n-vector $\boldsymbol{S}_i$. As their share, player P_i is given a key share $\boldsymbol{K}[i]$, a ciphertext share $\boldsymbol{C}[i]$, and the i^{th} element from each of the vectors $\boldsymbol{S}_i$. We can recognise that this share is equivalent to a share from HK0, with an additional n other elements, where each element is a codeword of the digest of every player's ciphertext share and key share.

The recover algorithm, defined in Algorithm 4.6, inputs an n-vector $\boldsymbol{V}$, where the i^{th} element $\boldsymbol{V}[i]$ is either a string $\boldsymbol{V}[i] \in \{0,1\}^*$, or the value $\Diamond$. Each share is parsed into its components. The elements $\boldsymbol{S}_i[1], \ldots, \boldsymbol{S}_i[n]$ are then used as the vector $\boldsymbol{S}_i$ to reconstruct the values h_i via the ECC recover algorithm. The digest of each pair $(\boldsymbol{K}[i]||\boldsymbol{C}[i])$ is then computed. If $Hash(\boldsymbol{K}[i]||\boldsymbol{C}[i])$ does not equal the reconstructed value h_i, the ciphertext share and key share are replaced with $\Diamond$, meaning they are corrupt. Finally, the key k and the ciphertext C are recovered via the threshold scheme and IDA recover algorithms, then C is decrypted under k to recover the data, M.

Algorithm 4.7: $Share^{A1}(M)$.

```
1 $k \xleftarrow{\$} KeyGen(\{0,1\}^\lambda)$;
2 $C \xleftarrow{\$} Enc_k(M)$;
3 $h = H(C)$;
4 $c_d = h \oplus k$;
5 $\boldsymbol{X} \leftarrow Share^{IDA}(C||c_d)$;
6 for $i \leftarrow 1$ to $n$ do
      $h_i \leftarrow Hash(\boldsymbol{X}[i])$;
      $\boldsymbol{S}_i \xleftarrow{\$} Share^{ECC}(h_i)$;
7 for $i \leftarrow 1$ to $n$ do
      $\boldsymbol{V}[i] \leftarrow (\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i])$;
8 return $\boldsymbol{V}$.
```

Algorithm 4.8: $Recover^{A1}(\boldsymbol{V})$.

```
1 for $i \leftarrow 1$ to $n$ do
      $(\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i]) \leftarrow \boldsymbol{V}[i]$;
2 for $i \leftarrow 1$ to $n$ do
      $h_i \leftarrow Recover^{ECC}(\boldsymbol{S}_i)$;
3 for $i \leftarrow 1$ to $n$ do
      if $\boldsymbol{V}[i] \neq \Diamond$ and
       $Hash(\boldsymbol{X}[\boldsymbol{i}]) \neq h_i$ then
          $\boldsymbol{X}[i] \leftarrow \Diamond$;
4 $C||c_d \leftarrow Recover^{IDA}(\boldsymbol{X})$;
5 $h = H(C)$;
6 $k = h \oplus c_d$;
7 $M \leftarrow Dec_k(C)$;
8 return $M$.
```

Figure 4.8: The share and recover algorithms defining AONT-RS1, the robust extension of AONT-RS0 via hash functions.

4.4.2.2 AONT-RS1

Let $\mathcal{E}$ be an ind-1 and key-1 secure symmetric-key encryption scheme. Assume the existence of a (t,n)-IDA (with no security requirements), a (t,n)-ECC (that is, an IDA with security equivalent to a linear $(0,t;n)$-ramp scheme) and a hash function $Hash$ that is indistinguishable from a RO. Let Π_1 denote AONT-RS1, with share and recover algorithms as in Figure 4.8. Let the ciphertext be $C \in \{0,1\}^\omega$, where $\omega \geq (t-1)\lambda$.

We believe AONT-RS1 to be both private and recoverable in the RO model. We have not formally proved this, but the proof is likely to require minor adaptations to the somewhat cumbersome proofs of Theorem 1 in [8] (to show the scheme is private) and Theorem 3 in [8] (to show the scheme is recoverable). These adaptations are expected to be similar to those made to the proof of HK2 in order to prove AONT-RS2 is private and recoverable, which follows in Section 4.4.3.3.

4.4.3 Robust extension using commitment schemes

After proving HK1 is private and recoverable in the RO model, Bellare and Rogaway looked to construct a robust scheme that was recoverable under standard assumptions. They achieved their goal by using commitment schemes, rather than hash functions [8].

The resulting robust scheme is called HK2.

In this section, we briefly introduce commitment schemes and HK2, then apply the technique to extend AONT-RS0 to be recoverable under standard assumptions.

4.4.3.1 Committment schemes

Intuitively, a *commitment scheme* allows player P_i to commit to a message M without revealing any information about M to a different player, P_j. When P_i does want to reveal M to P_j, P_j is assured the plaintext revealed by P_i was the plaintext originally committed to.

Definition 4.4.1. *A* commitment scheme*, denoted by $\mathcal{CS}$, is a set of three algorithms, $ParGen$, Ct and Vfy, as follows.*

- *The parameter generation algorithm $ParGen$ outputs the parameters of the scheme, denoted by π.*
- *The commitment algorithm Ct takes as input the parameters of the scheme, π, and the message to be committed to, M. It outputs two values: a committal H, which is given to player P_j, and a decommittal R which is kept by P_i.*
- *The verification algorithm Vfy is run when the message M is being revealed. It takes as input the triple (H, M, R) and outputs a '1' if the inputs are valid, or a '0' otherwise.*

A commitment scheme should satisfy both the hiding and binding properties, defined in Figure 4.9 and attributed to [8]. Intuitively, if the commitment scheme has the hiding property, P_i is assured P_j learns no information about M prior to the reveal. If the scheme has the binding property, P_j is assured P_i has revealed the message that was originally committed to.

In the hiding game, defined in Game 4.9, the challenger begins by generating the parameters and randomly choosing a bit $b \in \{0, 1\}$. The adversary must submit two valid messages M_0 and M_1 during the deal procedure and the challenger will use M_b as an input to the Ct algorithm. The adversary will receive a committal H back. The adversary is

Game 4.9: *Hide*

```
Procedure Initialise
1   $\pi \xleftarrow{\$} ParGen$;
2   $b \xleftarrow{\$} \{0,1\}$;
Procedure $Deal(M_0, M_1)$
1   if $M_0, M_1 \notin \mathcal{M}$ then
        return $\perp$;
2   else $(H, R_0) \xleftarrow{\$} Ct(\pi, M_b)$;
3   return $(H)$;
Procedure $Finalise(b')$
1   if $b' = b$ then
        return true;
2   else return false;
```

Game 4.10: *Bind*

```
Procedure Initialise
1   $\pi \xleftarrow{\$} ParGen$;
Procedure $Commit(M_0)$
1   if $M_0 \notin \mathcal{M}$ then
        return $\perp$;
2   else $(H, R_0) \xleftarrow{\$} Ct(\pi, M_0)$;
3   return $(H, R_0)$;
Procedure $Finalise(M_1, R_1)$
1   if $M_1 \notin \mathcal{M}$ then
        return $\perp$;
2   if $M_0 = M_1$ then
        return $\perp$;
3   if $Vfy(H, M_0, R_0) =$
      $Vfy(H, M_1, R_1) = 1$ then
        return true;
4   else return false;
```

Figure 4.9: Hiding and binding games used to define security notions in a commitment scheme.

allowed to make up to q queries during the deal procedure. After this, the adversary must submit a guess b' for b. The adversary wins if they guess b correctly.

Call adversary $\mathcal{A}$ playing the hiding game against $\mathcal{CS}$ a *hiding adversary.* Let $\Pr[Hide^{\mathcal{A}}]$ be the probability $\mathcal{A}$ correctly guesses $b' = b$. The advantage of $\mathcal{A}$ is

$$\mathbf{Adv}_{\mathcal{CS}}^{Hide}(\mathcal{A}) = 2 \cdot \Pr[Hide^{\mathcal{A}}] - 1.$$

Say $\mathcal{CS}$ is $\epsilon(\cdot)$*-hiding* if $\mathbf{Adv}_{\mathcal{CS}}^{Hide}(\mathcal{A}) \leq \epsilon(q)$ for any adversary that makes at most q queries during the deal procedure.

In the binding game, defined in Game 4.10, the challenger generates the parameters, then the adversary submits a valid message M_0. The challenger uses M_0 as an input to Ct and returns both the committal H and the decommittal R_0 to the adversary. Once the adversary has received the pair (H, R_0), they must submit a distinct message M_1 along with a decommittal R_1. The adversary wins if the triple (H, M_1, R_1) verifies successfully.

Algorithm 4.11: $Share^{HK2}(M)$.

```
1 $k \xleftarrow{\$} KeyGen(\{0,1\}^\lambda)$;
2 $C \xleftarrow{\$} Enc_k(M)$;
3 $\boldsymbol{K} \xleftarrow{\$} Share^{PTS}(k)$;
4 $\boldsymbol{C} \xleftarrow{\$} Share^{IDA}(C)$;
5 for $i \leftarrow 1$ to $n$ do
      $(H_i, R_i) \xleftarrow{\$} Ct(\boldsymbol{K}[i]||\boldsymbol{C}[i])$;
      $\boldsymbol{S}_i \xleftarrow{\$} Share^{ECC}(H_i)$;
6 for $i \leftarrow 1$ to $n$ do
      $\boldsymbol{V}[i] \leftarrow$
        $(R_i||\boldsymbol{K}[i]||\boldsymbol{C}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i])$;
7 return $\boldsymbol{V}$.
```

Algorithm 4.12: $Recover^{HK2}(\boldsymbol{V})$.

```
1 for $i \leftarrow 1$ to $n$ do
      $(R_i||\boldsymbol{K}[i]||\boldsymbol{C}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i]) \leftarrow$
        $\boldsymbol{V}[i]$;
2 for $i \leftarrow 1$ to $n$ do
      $H_i \leftarrow Recover^{ECC}(\boldsymbol{S}_i)$;
3 for $i \leftarrow 1$ to $n$ do
      if $\boldsymbol{V}[i] \neq \Diamond$ and
        $Vfy(H_i, \boldsymbol{K}[i]\boldsymbol{C}[i], R_i) = 0$ then
          $\boldsymbol{K}[i] \leftarrow \Diamond$;
          $\boldsymbol{C} \leftarrow \Diamond$;
4 $k \leftarrow Recover^{PTS}(\boldsymbol{K})$;
5 $C \leftarrow Recover^{IDA}(\boldsymbol{C})$;
6 $M \leftarrow Dec_k(C)$;
7 return $M$.
```

Figure 4.10: The share and recover algorithms defining HK2, a robust, computationally secure threshold scheme by Bellare and Rogaway [8].

We call an adversary $\mathcal{A}$ playing *Bind* a *binding adversary.* The advantage of $\mathcal{A}$ is

$$\mathbf{Adv}_{\mathcal{CS}}^{Bind}(\mathcal{A}) = \Pr[Bind^{\mathcal{A}}].$$

There exist a number of commitment scheme that meet these security requirements. These include [19], which is based on discrete logarithms, and [31] [45], which both rely on a collision-resistant hash function.

4.4.3.2 HK2

The HK2 scheme consists of two algorithms $Share^{HK2}$ and $Recover^{HK2}$, defined in Figure 4.10. HK2 is similar to HK1 but, where HK1 uses a hash function and message digests, HK2 uses a commitment scheme and committals.

Similar to the HK1 share algorithm, HK2 begins the same as HK0. However, rather than computing the hash of $(\boldsymbol{K}[i]||\boldsymbol{C}[i])$, HK2 uses each pair as the message to be committed to in a commitment scheme. So, for each i, $(\boldsymbol{K}[i]||\boldsymbol{C}[i])$ is input to the committal algorithm Ct, which returns a committal H_i and a decommittal R_i. Each committal H_i is then shared via a (t, n)-ECC, which returns a vector $\boldsymbol{S}_i$. Each player's share then consists of

Algorithm 4.13: $Share^{A2}(M)$.

```
1 $k \xleftarrow{\$} KeyGen(\{0,1\}^\lambda)$;
2 $C \xleftarrow{\$} Enc_k(M)$;
3 $h = H(C)$;
4 $c_d = h \oplus k$;
5 $\boldsymbol{X} \leftarrow Share^{IDA}(C||c_d)$;
6 for $i \leftarrow 1$ to $n$ do
      $(H_i, R_i) \xleftarrow{\$} Ct(\boldsymbol{X}[i])$;
      $\boldsymbol{S}_i \xleftarrow{\$} Share^{ECC}(H_i)$;
7 for $i \leftarrow 1$ to $n$ do
      $\boldsymbol{V}[i] \leftarrow$
        $(R_i||\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i])$;
8 return $\boldsymbol{V}$.
```

Algorithm 4.14: $Recover^{A2}(\boldsymbol{V})$.

```
1 for $i \leftarrow 1$ to $n$ do
      $(R_i||\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i]) \leftarrow$
        $\boldsymbol{V}[i]$ ;
2 for $i \leftarrow 1$ to $n$ do
      $H_i \leftarrow Recover^{ECC}(\boldsymbol{S}_i)$ ;
3 for $i \leftarrow 1$ to $n$ do
      if $\boldsymbol{V}[i] \neq \Diamond$ and
        $Vfy(H_i, \boldsymbol{X}[i], R_i) = 0$ then
          $\boldsymbol{X}[i] \leftarrow \Diamond$;
4 $C||c_d \leftarrow Recover^{IDA}(\boldsymbol{X})$;
5 $h = H(C)$;    $k = h \oplus c_d$;
6 $M \leftarrow Dec_k(C)$;
7 return $M$ .
```

Figure 4.11: The share and recover algorithms defining AONT-RS2, the robust extension of AONT-RS0 via commitment schemes.

the key and ciphertext share $(\boldsymbol{K}[i]||\boldsymbol{C}[i])$, the decommittal for their shares R_i and an ECC contribution from each of the n committals.

As with HK1, the recover algorithm takes as input an n-vector $\boldsymbol{V}$, where the i^{th} element $\boldsymbol{V}[i]$ is either a string $\boldsymbol{V}[i] \in \{0,1\}^*$, or the value $\Diamond$. Each share is parsed into its components and each vector $\boldsymbol{S}_i$ is used to reconstruct the committal values H_i via the ECC recover algorithm. Then, for each share, the reconstructed committal H_i, the ciphertext and key share $(\boldsymbol{K}[i]||\boldsymbol{C}[i])$ and the decommittal R_i are input to the commitment scheme's verify algorithm, Vfy. If the triple successfully verify, $(\boldsymbol{K}[i]||\boldsymbol{C}[i])$ is accepted as correct. If the triple do not verify successfully, each value is replaced with a $\Diamond$ to denote a corrupted share. Then, as before, k, C and finally M are recovered.

4.4.3.3 AONT-RS2

We can use commitment schemes, as in HK2, to extend AONT-RS0 to be robust.

Let $\mathcal{E}$ be an ind-1 and key-1 secure symmetric-key encryption scheme. Assume the existence of a (t, n)-ECC, an IDA equivalent to a linear $(0, t; n)$-ramp scheme, an $\epsilon(\cdot)$-hiding commitment scheme $\mathcal{CS}$ and a hash function $Hash$ that is indistinguishable from a RO. Let Π_2 denote the AONT-RS2 scheme, with share and recover algorithms as in Figure 4.11. Let the ciphertext be $C \in \{0,1\}^\omega$, where $\omega \geq (t-1)\lambda$.

4.4.3.4 Proof of privacy

The AONT-RS2 scheme Π_2 can be proven to achieve computational privacy in the RO model by adapting the proof of privacy for the HK2 scheme by Bellare and Rogaway [8].

Theorem 4.4.2 (Privacy of AONT-RS2)**.** *Let $\mathcal{A}$ be a privacy adversary against the AONT-RS2 scheme Π_2 and let the hash function $Hash$ be indistinguishable from a random oracle. Then there is an ind-1 adversary $\mathcal{B}$ attacking the indistinguishability of $\mathcal{E}$ such that*

$$\mathbf{Adv}_{\Pi_2}^{Priv}(\mathcal{A}) \leq \mathbf{Adv}_{\mathcal{E}}^{Ind}(\mathcal{B}) \cdot 4\epsilon(n),$$

where $\mathcal{B}$ makes only one query during the deal procedure of Game Ind and the running time of $\mathcal{B}$ is that of $\mathcal{A}$ plus one execution of $Share^{A2}$.

Proof. As in [8], this proof relies on code-based game playing, a framework for which is available in [7]. A game in this case will consist of an *Initialise* procedure, and then four procedures that respond to adversary oracle queries, *Deal*, *Corrupt*, *Hash* and *Finalise*. This proof will rely on five games and, as many of the games have procedures in common, we define the procedures individually in Figure 4.12, and put next to the procedure the games in which it appears. There are six games in total, denoted $G_0, \ldots, G_5$. As an example of how to read Figure 4.12, G_0 consists of the *Initialise*, *Deal* and *Corrupt* procedures on the left-hand side and the only *Finalise* procedure (at the bottom), whilst G_3 consists of the *Initialise*, *Deal* and *Corrupt* procedures on the right-hand side, and the *Finalise* procedure.

The corrupt procedure in games G_1, G_2, G_3, G_4 refers to a probabilistic algorithm *DCt* that works as follows. On input message $\boldsymbol{X}[i]$ and committal H_i, it lets $\Omega(\boldsymbol{X}[i], H_i)$ denote the set of all coins ω such that Ct, on input $\boldsymbol{X}[i]$ and coins ω, returns a pair whose first component is H_i. If $\Omega(\boldsymbol{X}[i], H_i) = \emptyset$, then *DCt* returns $\perp$. Else, it picks ω at random from $\Omega(\boldsymbol{X}[i], H_i)$, runs Ct on input $\boldsymbol{X}[i]$ and coins ω to get a pair (H_i, R_i) and returns R_i. This algorithm is not necessarily efficiently implementable.

The advantage of the AONT-RS2 privacy adversary $\mathcal{A}$ can be defined as

$$\mathbf{Adv}_{\Pi_2}^{Priv}(\mathcal{A}) = 2 \cdot \Pr[G_0^{\mathcal{A}}] - 1.$$

```
Procedure Initialise    $G_0, G_1, G_2$
1   $k \xleftarrow{\$} KeyGen\{0,1\}^\lambda$;
2   $k' = k$;
3   $b \xleftarrow{\$} \{0,1\}$;
Procedure Initialise    $G_3, G_4, G_5$
1   $k \xleftarrow{\$} KeyGen\{0,1\}^\lambda$;
2   $k' \xleftarrow{\$} KeyGen\{0,1\}^\lambda$;
3   $b \xleftarrow{\$} \{0,1\}$;
```

```
Procedure Deal($x_0, x_1$)    $G_0, G_1, G_4, G_5$
1   $C \leftarrow Enc_k(x_b)$;
2   $h \leftarrow Hash(C)$;
3   $h \oplus k' = c_d$;
4   $\boldsymbol{X} \leftarrow Share^{ECC}(C||c_d)$;
5   for $i \leftarrow 1$ to $n$ do
        $(H_i, R_i) \xleftarrow{\$} Ct(\boldsymbol{X}[i])$;
        $\boldsymbol{S}_i \leftarrow Share^{ECC}(H_i)$
Procedure Deal($x_0, x_1$)    $G_2, G_3$
1   $C \leftarrow Enc_k(x_b)$;
2   $h \leftarrow Hash(C)$;
3   $h \oplus k' = c_d$;
4   $\boldsymbol{X} \leftarrow Share^{ECC}(C||c_d)$;
5   $\boldsymbol{C} \leftarrow Share^{ECC}(C||0)$;
6   for $i \leftarrow 1$ to $n$ do
        $(H_i, R_i) \xleftarrow{\$} Ct(\boldsymbol{C}[i])$;
        $\boldsymbol{S}_i \leftarrow Share^{ECC}(H_i)$
```

```
Procedure Corrupt($i$)    $G_0, G_5$
1   $\boldsymbol{V}[i] \leftarrow (R_i||\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i])$;
2   return $\boldsymbol{V}[i]$;
Procedure Corrupt($i$)    $G_1, G_2, G_3, G_4$
1   $R_i \xleftarrow{\$} DCt(H_i, \boldsymbol{X}[i])$;
2   $\boldsymbol{V}[i] \leftarrow (R_i||\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i])$;
3   return $\boldsymbol{V}[i]$;
```

```
Procedure Finalise($b'$)    $G_0, G_1, G_2, G_3, G_4, G_5$
1   if $b' = b$ then
        return true;
2   else return false;
```

Figure 4.12: Procedures used to construct games G_0, G_1, G_2, G_3, G_4 and G_5, which are used to prove Theorem 4.4.2, the privacy of AONT-RS2.

Game G_1 differs from G_0 only in the corrupt procedure, which re-samples R_i using DCt. Clearly,

$$\Pr[G_0^{\mathcal{A}}] = \Pr[G_1^{\mathcal{A}}] = \Pr[G_2^{\mathcal{A}}] + (\Pr[G_1^{\mathcal{A}}] - \Pr[G_2^{\mathcal{A}}]).$$

We construct an adversary $\mathcal{D}_1$ attacking the hiding property of $\mathcal{CS}$ such that

$$\Pr[G_1^{\mathcal{A}}] - \Pr[G_2^{\mathcal{A}}] = \mathbf{Adv}_{\mathcal{CS}}^{Hide}(\mathcal{D}_1). \tag{4.4.1}$$

Adversary $\mathcal{D}_1$ acts as the challenger to $\mathcal{A}$ and wishes to use $\mathcal{A}$'s advantage to gain an advantage against the hiding property of $\mathcal{CS}$. Adversary $\mathcal{D}_1$ picks $b \xleftarrow{\$} \{0,1\}$ and runs $\mathcal{A}$. When $\mathcal{A}$ submits x_0, x_1 to $\mathcal{D}_1$, $\mathcal{D}_1$ generates $k \xleftarrow{\$} \{0,1\}^{\lambda}$ and calculates $C \xleftarrow{\$} Enc_k(x_b)$. $\mathcal{D}_1$ then computes $H(C) = h$ and $h \oplus k = c_d$, then calculates both $\boldsymbol{X} \leftarrow Share^{IDA}(C||c_d)$ and $\boldsymbol{C} \leftarrow Share^{IDA}(C||0)$. For i, $1 \leq i \leq n$, $\mathcal{D}_1$ submits $\boldsymbol{C}[i], \boldsymbol{X}[i]$ (for $\boldsymbol{X}[i] \neq \boldsymbol{C}[i]$) during the deal procedure of the hiding game to its challenger. Let H_i denote the commitment value returned. Let $\boldsymbol{S}_i \leftarrow Share^{ECC}(H_i)$. When $\mathcal{A}$ makes a $Corrupt(i)$ query to $\mathcal{D}_1$, $\mathcal{D}_1$ computes its reply according to the case of the corrupt procedure of games G_1 and G_2; that is, $\mathcal{D}_1$ generates a decommittal value R_i for $\boldsymbol{X}[i]$ and the given H_i and passes $\boldsymbol{V}[i] \leftarrow (R_i||\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i])$ to $\mathcal{A}$. When $\mathcal{A}$ halts the corrupt procedure and finalises with output b', if $b' = b$, adversary $\mathcal{D}_1$ passes '1' to its challenger, guessing the commitment value H_i was computed on $\boldsymbol{X}[i]$, rather than $\boldsymbol{C}[i]$. Otherwise, $\mathcal{D}_1$ submits '0'.

Next, we have that

$$\Pr[G_2^{\mathcal{A}}] = \Pr[G_3^{\mathcal{A}}] + \left(\Pr[G_2^{\mathcal{A}}] - \Pr[G_3^{\mathcal{A}}]\right),$$

where G_3 differs from G_2 only in the initialise procedure which XORs the digest h not with the key k, but with a string k'. We claim that $\Pr[G_2^{\mathcal{A}}] = \Pr[G_3^{\mathcal{A}}]$ because the hash function is indistinguishable from a RO. After $\mathcal{A}$ has corrupted at most $t-1$ shares, they learn at most either

- no information about c_d and all of C, and so can learn $h = H(C)$; in which case $h = k \oplus c_d = k' \oplus c'_d$, where $c'_d \neq c_d$ is some unknown string; or
- all of c_d, but is missing at least λ bits of C; then $c_d = k' \oplus h'$, where $h' \neq h$ is unknown to $\mathcal{A}$.

In either case, the adversary learns either h or c_d and no information about k. Thus the known value is the XOR of two unknown strings: changing one of these strings does not affect the chances of $\mathcal{A}$ winning, thus $\Pr[G_2^{\mathcal{A}}] = \Pr[G_3^{\mathcal{A}}]$.

Next, we have

$$\Pr[G_3^{\mathcal{A}}] = \Pr[G_4^{\mathcal{A}}] + \left(\Pr[G_3^{\mathcal{A}}] - \Pr[G_4^{\mathcal{A}}]\right).$$

Construct adversary $\mathcal{D}_2$, also attacking the hiding property of $\mathcal{CS}$, such that

$$\Pr[G_3^{\mathcal{A}}] - \Pr[G_4^{\mathcal{A}}] = \mathbf{Adv}_{\mathcal{CS}}^{Hide}(\mathcal{D}_2). \tag{4.4.2}$$

The construction of $\mathcal{D}_2$ is similar to $\mathcal{D}_1$, but $\mathcal{D}_2$ generates $k, k' \xleftarrow{\$} \{0,1\}^{\lambda}$, encrypts x_b under k as before and now calculates $c_d = h \oplus k'$.

Games G_5 and G_4 differ only during the corrupt procedure. Clearly $\Pr[G_4^{\mathcal{A}}] = \Pr[G_5^{\mathcal{A}}]$.

Let $\mathcal{B}$ be an ind-1 adversary attacking $\mathcal{E}$, as in the proof of Theorem 4.3.1. The advantage of $\mathcal{B}$ is as in (4.3.3).

Now, let $\mathcal{D}$ be the hiding adversary that flips a fair coin and, if it lands heads, runs $\mathcal{D}_1$, otherwise $\mathcal{D}_2$. Clearly,

$$\mathbf{Adv}_{\mathcal{CS}}^{Hide}(\mathcal{D}) = \frac{1}{2}\left(\mathbf{Adv}_{\mathcal{CS}}^{Hide}(\mathcal{D}_1) + \mathbf{Adv}_{\mathcal{CS}}^{Hide}(\mathcal{D}_2)\right). \tag{4.4.3}$$

Since Ct is assumed to be $\epsilon(\cdot)$-hiding and $\mathcal{D}$ makes at most n queries, we have that $\mathbf{Adv}_{\mathcal{CS}}^{Hide}(\mathcal{D}) \leq \epsilon(n)$. Combining this and (4.4.1), (4.4.2), (4.4.3) gives us

$$\left(\Pr[G_1^{\mathcal{A}}] - \Pr[G_2^{\mathcal{A}}]\right) + \left(\Pr[G_3^{\mathcal{A}}] - \Pr[G_4^{\mathcal{A}}]\right) \leq 2\epsilon(n).$$

As we know that $\Pr[G_2^{\mathcal{A}}] = \Pr[G_3^{\mathcal{A}}]$ and $\Pr[G_4^{\mathcal{A}}] = \Pr[G_5^{\mathcal{A}}]$, and by substituting in the advantages of adversaries $\mathcal{A}$ and $\mathcal{B}$, we can simplify and rearrange to give

$$\mathbf{Adv}_{\Pi_2}^{Priv}(\mathcal{A}) \leq \mathbf{Adv}_{\mathcal{E}}^{Ind}(\mathcal{B}) \cdot 4\epsilon(n),$$

```
Procedure Deal(x)
1   $\ell \xleftarrow{\$} [1,n]$;   $k \xleftarrow{\$} \{0,1\}^\lambda$;
2   $C \xleftarrow{\$} Enc_k(x)$;
3   $H(C) = h$;
4   $h \oplus k = c_d$;
5   $\boldsymbol{X} \leftarrow Share^{ECC}(C||c_d)$;
6   for $i \leftarrow 1$ to $n$ do
        if $i = \ell$ then
            $(H_\ell, R_\ell) \xleftarrow{\$} Commit(\boldsymbol{X}[i])$;
7       else $(H_i, R_i) \xleftarrow{\$} Ct(\boldsymbol{X}[i])$;
        $\boldsymbol{S}_i \xleftarrow{\$} Share^{ECC}(H_i)$;
8   for $i \leftarrow 1$ to $n$ do
        $\boldsymbol{V}[i] \leftarrow$
          $(R_i||\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i])$
```

```
Procedure Corrupt(i)
1   return $\boldsymbol{V}[i]$;
```

```
Procedure Finalise(x', j)
1   for $i \leftarrow 1$ to $n$ do
        $(R_i||\boldsymbol{X}[i]||\boldsymbol{S}_1[i]||\ldots||\boldsymbol{S}_n[i]) \leftarrow$
          $\boldsymbol{V}'[i]$;
2   return $(\boldsymbol{X}[\ell], R_\ell)$
```

Figure 4.13: Procedures used by adversary $\mathcal{B}$ to respond to adversary $\mathcal{A}$ in the proof of Theorem 4.4.3, the robustness of AONT-RS2.

thus completing the proof. □

4.4.3.5 Proof of robustness

AONT-RS2 can be proven to be computationally robust by adapting the proof of robustness of HK2 by Bellare and Rogaway [85].

Theorem 4.4.3 (Robustness of AONT-RS2)**.** *Let $\mathcal{A}$ be a recoverability adversary against the AONT-RS2 scheme Π_2. Then there is an adversary $\mathcal{B}$ attacking the binding property of the commitment scheme $\mathcal{CS}$ such that*

$$\mathbf{Adv}^{Rec}_{\Pi_2}(\mathcal{A}) \leq n \cdot \mathbf{Adv}^{Bind}_{\mathcal{CS}}(\mathcal{B}),$$

where the running time of $\mathcal{B}$ is that of $\mathcal{A}$ plus overhead consisting of an execution of the $Share^{A2}$ and $Recover^{A2}$ algorithms of Π_2.

Proof. Let $\mathcal{A}$ be a recoverability adversary against Π_2. During the deal procedure, $\mathcal{A}$ submits x to $\mathcal{B}$. Let k, C, h, c_d, $\boldsymbol{X}$, $H_1, \ldots, H_n$, $\boldsymbol{S}_1, \ldots, \boldsymbol{S}_n$, $\boldsymbol{V}$ denote the quantities generated by $Share^{A2}(x)$. Let $\mathcal{A}$ corrupt at most $t-1$ shares. Let $(\boldsymbol{V}_T)$ denote the output of $\mathcal{A}$. Let k', C', h', c'_d, $\boldsymbol{X}'$, $H'_1, \ldots, H_n$, $\boldsymbol{S}'_1, \ldots, \boldsymbol{S}'_n$, $\boldsymbol{V}'$ denote the quantities recovered by $Recover^{A2}$ with input $\boldsymbol{V}'_T \cup \boldsymbol{V}_{\overline{T}}$. Consider the following events:

E_1: $\exists\, \ell \in [n]$ such that $H_\ell \neq H'_\ell$

E_2: $\exists\, \ell \in T$ such that $\boldsymbol{X}'[i] \notin \{\Diamond, \boldsymbol{X}[i]\}$

E_3: $c_d \neq c'_d$

E_4: $C \neq C'$

If $C' = C$ and $c'_d = c_d$, then the recovered secret x' equals x. This is because $h' = H(C') = H(C) = h$ and so $c'_d \oplus h' = c_d \oplus h = k$. Therefore

$$\begin{aligned}\mathbf{Adv}^{Rec}_{\Pi_2}(\mathcal{A}) &\leq \Pr[E_3 \cup E_4] \\ &\leq \Pr[E_1 \cup E_2 \cup E_3 \cup E_4] \\ &= \Pr[E_1] + \Pr[\overline{E_1} \cap E_2] + \Pr[\overline{E_2} \cap E_3] + \Pr[\overline{E_2} \cap E_4].\end{aligned}$$

We bound each term in turn. Let $E_{1,\ell}$ be the event that $H'_\ell = H_\ell$. Let T be the set of indexes of the shares corrupted by $\mathcal{A}$. If $i \notin T$, then the submission is $\boldsymbol{V}'[i]$ and the other uncorrupted shares returns $\boldsymbol{V}[i]$. Hence $\boldsymbol{S}'_\ell[i] = \boldsymbol{S}_\ell[i]$. Note that $\boldsymbol{S}_\ell$ is an output of $\mathit{Share}^{ECC}(H_\ell)$. Lemma 10 in [8] discusses perfect recoverability and, when applied to ECCs, $\mathit{Recover}^{ECC}(\boldsymbol{S}_\ell) = H_\ell$, meaning that $H'_\ell = H_\ell$. So $\Pr[E_{1,\ell}] = 0$. By the well-known union bound [18],

$$\Pr[E_1] \leq \sum_{\ell=1}^{n} \Pr[E_{1,\ell}] = 0.$$

Now we construct adversary B such that

$$\Pr[\overline{E_1} \cap E_2] \leq n \cdot \mathbf{Adv}^{Bind}_{\mathcal{CS}}(\mathcal{B}).$$

Adversary $\mathcal{B}$ runs $\mathcal{A}$, responding to its deal and corrupt calls via the procedures in Figure 4.13, where Ct is the committal algorithm of $\mathcal{CS}$ run by $\mathcal{B}$ and $Commit$ is a procedure of the $Bind$ game that $\mathcal{B}$ plays with its challenger. When $\mathcal{A}$ halts with output $\boldsymbol{X}$, $\mathcal{B}$ runs the finalise procedure.

Next, we claim both $\Pr[\overline{E_2} \cap E_3] = 0$ and $\Pr[\overline{E_2} \cap E_4] = 0$. As, if $\boldsymbol{X}'[i] = \boldsymbol{X}[i]$ for all i, then $C = C'$ and $c'_d = c_d$. So now

$$\begin{aligned}\mathbf{Adv}^{Rec}_{\Pi_2}(\mathcal{A}) &= \Pr[E_1] + \Pr[\overline{E_1} \cap E_2] + \Pr[\overline{E_2} \cap E_3] + \Pr[\overline{E_2} \cap E_4] \\ &\leq n \cdot \mathbf{Adv}^{Bind}_{\mathcal{CS}}(\mathcal{B}),\end{aligned}$$

thus completing the proof. □

4.4.4 Comparing robust extensions

The comparison between AONT-RS0 and HK0, as in Section 4.3.4, can mostly apply to the comparison of AONT-RS1 with HK1, and AONT-RS2 with HK2.

AONT-RS1 and HK1 both achieve privacy and recoverability in the RO model. The share sizes of both AONT-RS1 and HK1 remain the same as in AONT-RS0 and HK0, plus an equivalent contribution to the shares in both schemes from the digests. Similarly, the share and recover algorithms have the same complexity, with the addition of the complexity contributed from the hash functions, which is equivalent for both AONT-RS1 and HK1. So, HK1 is more efficient than AONT-RS1 (in terms of share size and the number of operations required to run the share and recover algorithms), and, unlike HK0 and AONT-RS0, achieves equivalent security, as both are private and recoverable only in the RO model. Therefore, according to the metrics considered here, AONT-RS1 is more efficient than, whilst achieving equivalent security to, HK1.

AONT-RS2 achieves privacy in the RO model and recoverability under standard assumptions, whereas HK2 achieves both privacy and recoverability under standard assumptions. The share sizes of both AONT-RS2 and HK2 remain the same as in AONT-RS0 and HK0, plus an equivalent contribution to the shares in both schemes from the commitment scheme. Similarly, the share and recover algorithms have the same complexity, with the addition of the complexity contributed from the commitment scheme, which is equivalent for both AONT-RS2 and HK2.

Of the four robust, computationally secure threshold schemes (HK1, HK2, AONT-RS1 and AONT-RS2), AONT-RS1 is the most efficient (with respect to share size and the number of operations in the share and recover algorithms). This is because both AONT-RS0 is more efficient than HK0 (implying AONT-RS1 is more efficient than HK1) and because hash functions are more efficient than commitment schemes (and so AONT-RS1 is more efficient than AONT-RS2). However, AONT-RS1 is less secure than both HK2 and AONT-RS2 (but achieves an equivalent security to HK1). In contrast, HK2 is the least efficient, but the most secure, as it is both private and recoverable under standard assumptions.

4.5 Conclusion

We generalised AONT-RS and showed information is leaked when ciphertexts are smaller than $\lambda(t-1)$ bits. We proved the generalised version of AONT-RS, called AONT-RS0, has computational privacy in the RO model. We then extended AONT-RS to be robust using two techniques: hash functions (which resulted in AONT-RS1), and commitment schemes (resulting in AONT-RS2). We suggested AONT-RS1 achieves computational privacy and recoverability, both in the RO model, with the proof being similar to the proof of HK1, as in [8]. We proved AONT-RS2 achieves computational privacy in the RO model and recoverability under standard assumptions by adapting the proof of HK2, also as in [8].

Finally, we compared AONT-R0S with HK0, considering the security, share size, and efficiency of the share and recover algorithms. We used this comparison to compare AONT-RS1 with HK1 and AONT-RS2 with HK2. We concluded that AONT-RS1 is the most efficient robust extension but is the least secure (with a security equivalent to that of HK1) whilst HK2 is the most secure, but the least efficient of the four robust schemes considered.

Chapter 5

Repairable threshold schemes

Contents

This chapter, which has been accepted as a paper by the Journal of Mathematical Cryptology [64], is joint work with Doug Stinson and considers the problem of repairable threshold *schemes, which allow players to securely repair a lost or corrupted share without the help of the dealer.*

5.1 Introduction

In this section, we motivate and define repairable threshold schemes and the efficiency metrics that will be used to analyse them.

5.1.1 Problem statement

Consider a scenario in which a player in a (t, n)-threshold scheme loses or corrupts their share and must repair it. In some settings, the player wishing to repair their share, called the *repairing player*, could communicate with the dealer and request, then receive, a copy of their share. However, the dealer may not always be accessible when a player needs to repair their share. Ideally, in this dealer-less setting, the repairing player could ask for help from its cohort of players to repair its share. A scheme in which this is possible is called a *repairable threshold scheme* (RTS).

Now, in a repairable threshold scheme, we wish to maintain the security of the underlying threshold scheme and not leak any information during a repair, and thus we must define what it means for the repair algorithm to be secure. Assume a setting in which all players execute the repair algorithm correctly. Inherited from the security definition of a (t, n)-threshold scheme, consider an adversary with access to a coalition of at most $t-1$ players, which may or may not include the repairing player. Each time the repair algorithm is executed, the coalition of at most $t-1$ players will pool their information; this includes the information stored prior to the algorithm being executed, as well as all messages sent and received during. In order to be secure, the accumulated information should yield no information about the secret distributed by the RTS.

Definition 5.1.1. *Let $d \in \mathbb{N}$ be such that $t \leq d \leq n-1$ and call d the* repairing degree*. A* (t, n, d)-repairable threshold scheme*, or a (t, n, d)-RTS, is a perfect (t, n)-threshold scheme, as in Definition 2.3.2, which, in addition to the Share and Recover algorithms, has a Repair algorithm that allows a repairing player P_r to construct their original share with* *help from some set of d other players, called the* helping players*. The Repair algorithm must be secure; that is, no player learns any new information (about either the secret or other player's shares) after the repair than was known before.*

We briefly note the bounds on the repairing degree d. First, consider the lower bound $t \leq d$. This is a necessary condition since, if a coalition of fewer than t players were able to construct a share for a player not in the coalition, then a coalition of $t-1$ players would be able to construct a t^{th} share and thus have enough shares to recover the secret s via the recover algorithm. This would contradict the privacy of the RTS, inherited from the threshold scheme, and would thus be insecure. The upper bound on the repairing degree is $d \leq n-1$. This is also obvious since, if one of the n players lost their share, there are at most $n-1$ players that could possibly help. We remark that it is desirable to have a small d, as this allows repairability to be more robust. For example, if $d = n-1$ and if two players are unavailable (they may be offline or corrupted), no players will be able to repair their shares. In contrast, if d is small, repairability would be possible even in a setting where a number of players are unavailable.

Finally, we introduce a notion, defined in [97], regarding the repairability of an RTS. As motivation for this definition, we note that in Definition 5.1.1 not every d-subset of the $n-1$ players is required to be able to help the repairing player P_r repair their share. Instead, it is necessary that at least one d-subset can help P_r. In order to distinguish between schemes in which all d-subsets, or some d-subsets, are able to help P_r, we introduce the following definition.

Definition 5.1.2. *a A (t, n, d)-RTS has* universal repairability *if all d-subsets of the potential $n-1$ helper players are able to securely repair P_r's share. A (t, n, d)-RTS has* restricted repairability *if some, but not every, d-subset of the potential helper players are able to securely repair P_r's share.*

5.1.2 Naïve solution

Consider Construction 5.1.3, which presents a naïve construction for a universally repairable (t, n, d)-RTS.

Construction 5.1.3. *Let s be the secret to be distributed. A (t, n, d)-RTS is defined by* *three algorithms, Share, Recover and Repair, as follows.*

- *Share: Distribute the secret s via a perfect (t, n)-threshold scheme, to give an n-vector $\boldsymbol{V}$. Distribute each element of $\boldsymbol{V}$ via a $(d, n-1)$-threshold scheme to give $\boldsymbol{T}_i \overset{\$}{\leftarrow} Share_d(\boldsymbol{V}[i])$, for $1 \leq i \leq n$, where $\boldsymbol{T}_i$ is an n-vector with $\boldsymbol{T}_i[i] = \perp$. As their share, player P_i receives $(\boldsymbol{V}[i], \boldsymbol{T}_1[i], \boldsymbol{T}_2[i], \ldots, \boldsymbol{T}_n[i])$*
- *Recover: A set of players pool their shares. The elements $\boldsymbol{V}[i]$ are input to the recover algorithm of the (t, n)-threshold scheme. If at least t players input valid shares, s will be recovered.*
- *Repair: If player P_r needs to repair their share, they request help from a set A of at least d players, who will each send P_r the element $\boldsymbol{T}_r[i]$, for i such that $P_i \in A$. Player P_r will recover their share via the recover algorithm of the $(d, n-1)$-threshold scheme.*

Intuitively, Construction 5.1.3 shares a secret s via a (t, n)-threshold scheme to give player P_i their share of the secret $\boldsymbol{V}[i]$. In order to enable repairability, each share is then shared via a $(d, n-1)$-threshold scheme to ensure d players can act as helping players. The scheme is secure as any $t-1$ players are unable to learn the secret due to the security of the (t, n)-threshold scheme, and each repair is secure due to the security of each of the $(d, n-1)$-threshold schemes.

5.1.3 Efficiency metrics

We are interested in the efficiency of an RTS and will consider the following metrics when analysing each scheme.

1. *Information Rate.* The first metric we consider is the information rate of the scheme, denoted by ρ. This is as defined previously in Definition 2.3.10 and measures the amount of information each player is required to store compared to the size of the secret. The information rate is such that $0 \leq \rho \leq 1$, where an RTS with $\rho = 1$ is called *ideal*.

2. *Communication Complexity.* As defined in [97], the communication complexity is the sum of the sizes (*i.e.* the bit lengths) of all messages transmitted during the repair algorithm, divided by the size of the secret. The communication complexity measures the amount of bandwidth required for each execution of the repair algorithm. We denote the communication complexity by γ.

3. *Repairability.* We define the *repairability* of an RTS, denoted by κ, to be the number of d-subsets (of the $n-1$ players) that are able to help a repairing player P_r repair their share, divided by the number of possible d-subsets (of the $n-1$ players). Note that $0 \leq \kappa \leq 1$, where $\kappa = 1$ if and only if the RTS has universal repairability, as in Definition 5.1.2.

4. *Computational complexity.* We consider the computational complexity of the share, recover and repair algorithms of the RTS.

We will see that (t, n, d)-RTSs find a compromise between these metrics and may prioritise one at the cost of the others. In particular, there appears to be an inverse relation between the information rate and communication complexity of many schemes we consider.

We now revisit Construction 5.1.3 and consider how efficient the scheme is using these metrics.

Example 5.1.1. Consider the share, recover and repair algorithms defining a (t, n, d)-RTS as in Construction 5.1.3. Assuming both underlying threshold schemes are ideal, each player is required to store n shares from a the threshold scheme as their RTS share. Therefore, the information rate of the RTS is $\rho = 1/n$. An execution of the repair algorithm requires each of the d players to send one of their shares from the $(d, n-1)$-threshold scheme, so $\gamma = d$. Finally, the scheme has universal repairability, since any d-subset of players can help a repairing player repair their scheme, hence $\kappa = 1$.

Finally, we note the share algorithm of the RTS requires the dealer to run the share algorithm of the underlying (t, n)-threshold scheme once and the share algorithm of the $(d, n-1)$-threshold scheme n times. The repair algorithm requires P_r to run the recover algorithm of the underlying $(d, n-1)$-threshold scheme once, whilst the recover algorithm requires one run of the recover algorithm for the (t, n)-threshold scheme.

5.2 Preliminaries

In this section, we will present some core ideas in combinatorial design theory and regenerating codes that we will need later.

5.2.1 Combinatorial design theory

Combinatorial design theory deals with arranging elements into finite sets with certain properties.

Definition 5.2.1. *[96]. A d*esign *is a pair $(X, \mathcal{D})$ such that X is a set of elements, called* points*, and $\mathcal{D}$ is a collection of non-empty subsets, called* blocks*, of X.*

There is no restriction on the collection $\mathcal{D}$ to have distinct blocks; this is why $\mathcal{D}$ is called a collection, rather than a set. If all the blocks are distinct, the design is called *simple*.

The *degree* of a point $x \in X$ is the number of blocks the point x occurs in. The design is called *regular* if all points have the same degree. The *rank*, k, of a design is the largest block in the collection $\mathcal{D}$. If all blocks are the same size, the design is said to be *uniform*.

Balanced incomplete block designs are a widely studied type of design and are defined as follows:

Definition 5.2.2. *A (m, k, λ)*-balanced incomplete block design*, (m, k, λ)-BIBD, is a design such that*

1. *$|X| = m$,*
2. *each block in $\mathcal{D}$ contains exactly k points, and,*
3. *every pair of distinct points is contained in exactly λ blocks.*

For convenience, blocks will be written in the form abc, rather than $\{a, b, c\}$.

Note that a BIBD is a regular, uniform, simple design.

Example 5.2.1. The pair $(X, \mathcal{D})$ is a $(9,3,1)$-BIBD, where

$$X = \{1,2,3,4,5,6,7,8,9\}, \text{ and}$$
$$\mathcal{D} = \{123, 456, 789, 147, 258, 369, 159, 267, 348, 168, 249, 357\}.$$

The following theorem, presented in [96] and given here without proof, shows the degree of every point x in a BIBD.

Theorem 5.2.3. *In an (m, k, λ)-BIBD, every point occurs in exactly*

$$\tau = \frac{\lambda(m-1)}{k-1}$$

blocks.

The value τ in Theorem 5.2.3 is called the *replication number* of the design. The following result, also from [96] and also given without proof, provides more information about the structure of a BIBD and defines how many blocks a BIBD must have.

Theorem 5.2.4. *An (m, k, λ)-BIBD has exactly*

$$b = \frac{mr}{k} = \frac{m\lambda(m-1)}{k(k-1)}$$

blocks.

Continuing from Example 5.2.1, we compute the replication number τ and the number of blocks b for the BIBD.

Example 5.2.2. Consider the $(9,3,1)$-BIBD in Example 5.2.1. The replication number of the design is $\tau = 4$ and $\mathcal{D}$ contains $b = 12$ blocks.

In [97], the concept of a repairable distribution design is introduced, which will be used later in this chapter to construct RTSs. We define the concept here and continue our example.

Definition 5.2.5. *A* (t, ℓ_1, ℓ_2)-distribution design *is a design that satisfies the following two properties:*

1. *the union of any t blocks contain at least ℓ_2 points, and*

2. *the union of any $t-1$ blocks contains at most ℓ_1 points,*

where $\ell_2 - \ell_1 \geq 1$. The distribution design is repairable *if every point in the distribution design occurs in at least two blocks.*

Example 5.2.3. The $(9, 3, 1)$-BIBD in Example 5.2.1 is a $(2, 3, 5)$-distribution design, as the union of any two blocks contains at least five points, and an individual block contains at most three points. As every point occurs in $\tau = 4$ blocks, the distribution design is repairable.

Finally, we consider the definition of a basic repairing set, also defined in [97].

Definition 5.2.6. *A subset of y blocks contained in a (t, ℓ_1, ℓ_2)-distribution design is a* basic repairing set *of size y if every point in the design is contained in at least two blocks of the subset.*

Obviously, $y \leq b$. The following theorem lower bounds y; the proof is very similar to the the proof of Theorem 5.2.4 given in [96].

Theorem 5.2.7. *A basic repairing set of a (t, ℓ_1, ℓ_2)-distribution design, constructed from an (m, k, λ)-BIBD, has at least $2m/k$ blocks.*

Proof. Let $(X, \mathcal{D})$ be a basic repairing set of an (m, k, λ)-BIBD. Define a set

$$I = \{(y, A) : y \in X, A \in \mathcal{D}, y \in A\}.$$

We will compute $|I|$ in two different ways. First, there are m ways to choose $y \in X$. For each y, there are at least two blocks A such that $y \in A$. Hence, $|I| \geq 2m$. On the other hand, there are y ways to choose a block $A \in \mathcal{D}$. For each choice of A, there are k ways to choose $y \in A$. Hence, $|I| = yk$. Combining these two equations, we see that

$$y \geq \frac{2m}{k}, \tag{5.2.1}$$

as required. □

We illustrate the concept of basic repairing sets for our ongoing example.

Example 5.2.4. For the $(9,3,1)$-BIBD in Example 5.2.1, we can calculate the lower bound on the basic repairing set to be $6 \leq y$. The set $\{123, 456, 789, 147, 258, 369\}$ is a basic repairing set of minimal size.

5.2.2 Regenerating codes

Regenerating codes are a class of distributed storage codes introduced in 2010 by Dimakis *et al.* [33]. Regenerating codes distribute data between nodes and guarantee recoverability of the data with the cooperation of a sufficient number of nodes, and regeneration of lost or corrupted shares. Regenerating codes optimally trade the bandwidth needed for the regeneration of a failed node with the amount of data stored per node in the network.

There are no security requirements for regenerating codes. However, there exists literature exploring how to secure them. Here, we present an introduction to regenerating codes followed by a system model for securing them.

5.2.2.1 Introduction to Regenerating Codes

Regenerating codes, and the notation used to describe them, is as follows.

Definition 5.2.8. *Let $\mathbb{F}_q$ be a finite field, and let D denote the data to be distributed, where $D \in (\mathbb{F}_q)^B$. Say B is the number of data symbols. Let $n \in \mathbb{N}$. Consider a distributed storage system consisting of n* nodes*, each with the capacity to store α symbols in $\mathbb{F}_q$. Let $t, d \in \mathbb{N}$, such that $t \leq d < n$. An (n,t,d)*-regenerating code *distributes D amongst n nodes such that each node stores a* share *of the data, where each share consists of α elements in $\mathbb{F}_q$. The distribution should be* recoverable*, meaning that D is recoverable by any t of the n nodes, and* repairable*, meaning that any node in the network can repair their share of the data by downloading β elements in $\mathbb{F}_q$ from each of the d repairing nodes.*

In [33], the following bound on the number of data symbols distributed by an (n,t,d)-

regenerating code is established:

$$B \leq \sum_{i=0}^{t-1} \min\{\alpha, (d-i)\beta\}. \tag{5.2.2}$$

Using this bound, it can be deduced that, when B, t and d are fixed, there is a trade-off between the size of the shares, α, and the bandwidth β required for repair. At one extreme of this trade-off we minimise β first and then α; this is called the *minimum bandwidth regenerating (MBR) point.* At the other extreme we minimise α first and then β to get the *minimum storage regenerating (MSR) point.*

This gives us the following parameters for the MBR point:

$$\beta = \frac{2B}{t(2d-t+1)} \tag{5.2.3}$$
$$\alpha = \frac{2dB}{t(2d-t+1)}. \tag{5.2.4}$$

At the MSR point, the parameters are:

$$\alpha = \frac{B}{t} \tag{5.2.5}$$
$$\beta = \frac{B}{t(d-t+1)}. \tag{5.2.6}$$

Using these extreme points, we can define MBR and MSR codes.

Definition 5.2.9. *A* minimum bandwidth regenerating (MBR) code *is an* (n, t, d)*-regenerating code with parameters* (α, β, B) *satisfying the MBR points in* (5.2.3) *and* (5.2.4). *A* minimum storage regenerating (MSR) code *is an* (n, t, d)*-regenerating code with parameters* (α, β, B) *satisfying the MSR points in* (5.2.5) *and* (5.2.6).

Since MBR codes achieve the minimum possible repair bandwidth, a replacement node downloads only what it stores, so $\alpha = d\beta$. By substituting this into (5.2.2), we can see that an MBR code must satisfy

$$B = \left(td - \binom{t}{2}\right)\beta. \tag{5.2.7}$$

Similarly, MSR codes must satisfy $B = t\alpha$ and $d\beta = \alpha + (t-1)\beta$.

There exist a number of constructions in the literature for both MBR and MSR codes. A construction of MBR codes for all possible parameters n, t and d is given in [82]. Examples of constructions of MSR codes can be found for all parameters n, t and d in [43, 100], and for $d = 2t - 2$ in [82].

Example 5.2.5 shows a $(5, 2, 3)$-MBR code, constructed according to the method in [90].

Example 5.2.5. Let $n = 5$, $t = 2$ and $d = 3$, meaning that any two of the five nodes can recover the data, and any node can regenerate their share with help from three other nodes. If we let $\beta = 1$, then (5.2.7) tells us the number of message symbols that can be distributed is $B = 5$. Let all computations be in the field with eleven elements, $\mathbb{Z}_{11}$, and let

$$u_1 = 7;\ u_2 = 3;\ u_3 = 10;\ u_4 = 6;\ u_5 = 2.$$

be the five message symbols to be distributed.

Dispersal: Use a (public) generator matrix Ψ, with properties discussed in [82], and a message matrix M to generate the code $C = \Psi M$ as follows:

$$C = \Psi M = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 4 & 1 \\ 3 & 2 & 6 \\ 4 & 2 & 1 \\ 5 & 4 & 6 \end{pmatrix} \begin{pmatrix} u_1 & u_2 & u_4 \\ u_2 & u_3 & u_5 \\ u_4 & u_5 & 0 \end{pmatrix} = \begin{pmatrix} 5 & 4 & 8 \\ 10 & 4 & 9 \\ 8 & 8 & 0 \\ 7 & 1 & 6 \\ 6 & 1 & 5 \end{pmatrix}.$$

Each node P_i is then given row i, for $1 \leq i \leq n$, of C. Note that each row, and therefore each share, consists of $\alpha = 3$ elements in F_{11}, which satisfies the MBR points in (5.2.3) and (5.2.4).

Regeneration: Say node P_4 needs to regenerate their share. This can be done with help from any $d = 3$ other nodes as follows. Assume nodes P_1, P_2 and P_5 are the helper nodes. Let Ψ_i denote row i of Ψ. Each helper node P_i must calculate the inner product $(\Psi_i M)\Psi_4^T$: note that node P_i knows $(\Psi_i M)$ as their share, and Ψ is a public matrix, so Ψ_4 is also known. Therefore, the three helper nodes P_1, P_2 and P_5 each calculate the following, respectively:

$$(\Psi_1 M)\Psi_4^T = \begin{pmatrix} 5 & 4 & 8 \end{pmatrix} \begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix} = 3 \mod 11$$

$$(\Psi_2 M)\Psi_4^T = \begin{pmatrix} 10 & 4 & 9 \end{pmatrix} \begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix} = 2 \mod 11$$

$$(\Psi_5 M)\Psi_4^T = \begin{pmatrix} 6 & 1 & 5 \end{pmatrix} \begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix} = 9 \mod 11.$$

Each helper node then sends this value to P_4. So P_4 receives the triple $(3, 2, 9)$. The regenerating node P_4 then calculates the repair matrix Ψ_{repair}, consisting of the rows of Ψ related to the helper nodes, and calculates the inverse of Ψ_{repair}. So, here, as P_1, P_2 and P_5 are helper nodes, Ψ_{repair} consists of rows $1, 2$ and 5 of Ψ, as follows:

$$\Psi_{repair} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 4 & 1 \\ 5 & 4 & 6 \end{pmatrix}, \text{ and } (\Psi_{repair})^{-1} \begin{pmatrix} 3 \\ 2 \\ 9 \end{pmatrix}. \tag{5.2.8}$$

Node P_4 then multiples $(\Psi_{repair})^{-1}$ with the triple $(3, 2, 9)$ received from the helper nodes:

$$(\Psi_{repair})^{-1} \begin{pmatrix} 3 \\ 2 \\ 9 \end{pmatrix} = \begin{pmatrix} 9 & 9 & 8 \\ 4 & 1 & 1 \\ 10 & 1 & 2 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \\ 9 \end{pmatrix} = \begin{pmatrix} 7 \\ 1 \\ 6 \end{pmatrix}, \tag{5.2.9}$$

and thus recovers their lost share.

Recover: Any $t = 2$ players are able to recover the data $u_1, \ldots, u_5$. Assume nodes P_2 and P_3 collaborate to recover the data.

Let Ψ_{DC} be the data collector matrix, constructed from rows corresponding to player P_2

and P_3. So:

$$\Psi_{DC} = \begin{pmatrix} 2 & 4 & 1 \\ 3 & 2 & 6 \end{pmatrix}, \tag{5.2.10}$$

where the shares belonging to P_2 and P_3 are:

$$\Psi_{DC}M = \begin{pmatrix} 10 & 4 & 9 \\ 8 & 8 & 0 \end{pmatrix}. \tag{5.2.11}$$

We can use the properties of the message matrix M to observe that

$$\begin{pmatrix} 2 & 4 \\ 3 & 2 \end{pmatrix} \begin{pmatrix} u_4 \\ u_5 \end{pmatrix} = \begin{pmatrix} 9 \\ 0 \end{pmatrix}, \tag{5.2.12}$$

which can be solved to give $u_4 = 6$ and $u_5 = 2$. These can then be substituted into $\Psi_{DC}M$ to give:

$$\begin{aligned} \Psi_{DC}M &= \begin{pmatrix} 2 & 4 & 1 \\ 3 & 2 & 6 \end{pmatrix} \begin{pmatrix} u_1 & u_2 & 6 \\ u_2 & u_3 & 2 \\ 6 & 2 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 2u_1 + 4u_2 + 6 & 2u_2 + 4u_3 + 2 & 9 \\ 3u_1 + 2u_3 + 3 & 3u_2 + 2u_3 + 1 & 0 \end{pmatrix} = \begin{pmatrix} 10 & 4 & 9 \\ 8 & 8 & 0 \end{pmatrix}, \end{aligned}$$

which gives us four equations in three variables:

$$\begin{aligned} 2u_1 + 4u_2 &= 4 \\ 2u_2 + 4u_3 &= 2 \\ 3u_1 + 2u_2 &= 5 \\ 3u_2 + 2u_3 &= 7, \end{aligned}$$

which can be solved to find $u_1 = 7, u_2 = 3$ and $u_3 = 10$. Thus, all five data symbols have been recovered by the two nodes, P_2 and P_3.

5.2.2.2 Securing regenerating codes

A system model for securing regenerating codes was first presented in [78] and will be introduced here.

Consider an adversary who has access to (only) the data stored on ℓ_1 nodes. In addition to these ℓ_1 nodes, the adversary also has access to the data stored on, and all data downloaded during the regeneration of, ℓ_2 nodes. Suppose ℓ_1 and ℓ_2 are such that $\ell_1 + \ell_2 < t$. Call such an adversary an (ℓ_1, ℓ_2)-adversary.

It is important to establish how many regenerations an adversary can witness because regenerating codes do not have any security requirements, so each regeneration may reveal information about the data stored.

In fact, each regeneration of an MBR code is secure. This is because $\alpha = d\beta$, which means a regenerating node stores all data downloaded during a regeneration. Therefore, an eavesdropper does not obtain any extra information by having access to the data downloaded during a regeneration, as well as the data stored. Hence, when considering MBR codes, we can assume an (ℓ_1, ℓ_2)-adversary is an $(\ell_1 + \ell_2, 0)$-adversary.

In contrast, MSR codes have insecure regenerations. This is because $\alpha < \beta d$, which means each regenerating node downloads more data during a regeneration than it ultimately stores, and thus an adversary would learn more information witnessing a regeneration than they would if they only had access to the data stored. So, when discussing secure MSR codes, it is important to define how many regenerations an adversary can witness.

As a side note, a node for which an adversary has access to the data on, but cannot witness regenerate, could model the adversary gaining only momentary access. Whereas a node for which an adversary has access to both the data on and the data downloaded during a regeneration, could model an adversary with long term access to the node.

Recall the data to be distributed in a regenerating code was denoted D, such that $D \in (\mathbb{F}_q)^B$. Let $D^{(s)}$ denote the data to be securely distributed via a regenerating code, and let $D^{(s)} \in (\mathbb{F}_q)^{B^{(s)}}$. Say $B^{(s)}$ is the number of data symbols that can be (information theoretically) securely distributed. Note that $B^{(s)} \leq B$ and, in particular, $B - B^{(s)}$ is the cost of securing the data.

Definition 5.2.10. *A* secure (n, t, d)-regenerating code *is an (n, t, d)-regenerating code that distributes $B^{(s)}$ secure data symbols such that an (ℓ_1, ℓ_2)-adversary, for $\ell_1 + \ell_2 < t$, learns no information about the $B^{(s)}$ secure data symbols.*

We conclude this introduction to regenerating codes by stating two theorems, each providing tighter upper bounds on $B^{(s)}$.

In [78] the authors consider a regenerating code where an adversary could witness the regeneration of every node they had access to, so $\ell_1 = 0$.

Theorem 5.2.11. *Consider an (n, t, d)-regenerating code with parameters α and β and a $(0, \ell_2)$-adversary. The number of data symbols that can be information theoretically secured is*

$$B^{(s)} \leq \sum_{i=\ell_2}^{t-1} \min(\alpha, (d-i)\beta).$$

In [42], the authors consider the number of data symbols that can be securely distributed by an MSR code.

Theorem 5.2.12. *Consider an (n, t, d)-MSR code with an (ℓ_1, ℓ_2)-adversary. The number of data symbols that can be information theoretically secured is*

$$B^{(s)} \leq (t - \ell_1 - \ell_2)\left(1 - \frac{1}{d-t+1}\right)^{\ell_2} \alpha.$$

5.3 Existing solutions

In this section, we consider three (t, n, d)-RTSs presented in the literature. The first two schemes, outlined in Section 5.3.1.1 and 5.3.2, were presented in [97] by Stinson and Wei. The third scheme was presented in [44] and is introduced in Section 5.3.3.

5.3.1 The enrolment RTS

We introduce the enrolment RTS, which was originally proposed in [97]. We then refine this scheme to achieve a lower communication and computational complexity. Finally, we show that the refined scheme achieves the optimal communication complexity for an ideal RTS.

5.3.1.1 Definition of the enrolment RTS

A (t, n, d)-RTS with $d = t$ constructed from the NSG enrolment protocol [75, 76] is presented in [97]. (In fact, a scheme equivalent to the NSG enrolment protocol was presented much earlier by Benaloh in [9].) We refer to the (t, n, d)-RTS as the *enrolment RTS*.

Assume there exists a Shamir (t, n)-threshold scheme defined over $\mathbb{F}_q$, with shares distributed amongst n players. The share and recover algorithms of the enrolment RTS are identical to the share and recover algorithms in Shamir's threshold scheme, as in Construction 2.3.3. The repair algorithm of the enrolment RTS is as follows.

Suppose player P_r wishes to repair their share. Without loss of generality, assume the $d = t$ helping players are players $P_1, \dots, P_t$, with $r \geq t$. Suppose the share for P_r is $\varphi_r = f(r)$, where $f(x) \in \mathbb{F}_q[x]$ is a random polynomial of degree at most $t - 1$ whose constant term is the secret s. The share φ_r can be expressed as

$$\varphi_r = \sum_{i=1}^{t} \zeta_i \varphi_i, \tag{5.3.1}$$

where ζ_i is the public Lagrange coefficients of P_i. In order to repair P_r's share, the protocol proceeds as follows.

1. For all $1 \leq i \leq t$, player P_i computes random values $\delta_{j,i}$ for $1 \leq j \leq t$, such that

$$\zeta_i \varphi_i = \sum_{j=1}^{t} \delta_{j,i}. \tag{5.3.2}$$

2. For all $1 \leq i \leq t$, $1 \leq j \leq t$, player P_i transmits $\delta_{j,i}$ to P_j using a secure channel.

3. For all $1 \leq j \leq t$, player P_j computes

$$\sigma_j = \sum_{i=1}^{t} \delta_{j,i}. \tag{5.3.3}$$

4. For all $1 \leq j \leq t$, player P_j transmits σ_j to player P_r using a secure channel.

5. Player P_r computes their share φ_r using the formula

$$\varphi_r = \sum_{j=1}^{t} \sigma_j. \tag{5.3.4}$$

It is straightforward to verify that player P_r constructs their share correctly; that is that the value computed using (5.3.2), (5.3.3) and (5.3.4) is the same value as in (5.3.1). This is demonstrated in [97].

The enrolment protocol is proven to be secure in [97]. The proof highlights two cases: the first considers a coalition of $t-1$ players in which the players are contained in $\{P_1, \ldots, P_t\}$ and excludes P_r. The second case considers a coalition of $t-1$ players which includes P_r and $t-2$ players from $\{P_1, \ldots, P_t\}$. In either case, the coalition is unable to learn anything about a t^{th} share and thus learns no information about the secret.

In the proof, for convenience, they consider a *share-exchange matrix*, E, originally defined in [75]. We note this matrix here as it will be used to define a refined scheme in Section 5.3.1.3:

$$E = \begin{pmatrix} \delta_{1,1} & \delta_{2,1} & \cdots & \delta_{t,1} \\ \delta_{1,2} & \delta_{2,2} & \cdots & \delta_{t,2} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{1,t} & \delta_{2,t} & \cdots & \delta_{t,t} \end{pmatrix}. \tag{5.3.5}$$

There are a number of observations to be made about the matrix E. The $(j,i)^{\text{th}}$ entry $\delta_{j,i}$ of E is the message player P_i sends P_j. From (5.3.2), we learn that the sum of the entries in the i^{th} row of E is equal to $\zeta_i \varphi_i$. Also, from (5.3.3), the sum of the entries in the j^{th} column is equal to σ_j. Finally, from (5.3.2), (5.3.3) and (5.3.4), the sum of all entries in E is equal to φ_r. Intuitively, player P_i computes and sends all values in row i and receives all the values in column i.

5.3.1.2 Analysis of the enrolment RTS

We evaluate the efficiency of the enrolment RTS by considering the efficiency metrics presented in Section 5.1.3.

Each player is required to store only one share from Shamir's threshold scheme. As Shamir's scheme is an ideal threshold scheme, the enrolment RTS is ideal, and so $\rho = 1$. With respect to the information rate, the enrolment RTS is optimal.

Next, we consider the communication complexity of the scheme. Messages are only exchanged in Steps 2 and 4. In Step 4, each of the t helping players are required to send one message to each of the other $t-1$ helping players, which is a total of $t(t-1)$ messages. During Step 4, each of the t helping players must send one message to the repairing player P_r, which is an additional t messages. So, in total, the repair algorithm requires $t(t-1)+t$ messages to be sent. As each message is the size of the share and therefore the size of the secret (as Shamir's scheme is ideal), the communication complexity of the scheme is $\gamma = t^2$.

Inherited from Shamir's threshold scheme, the enrolment RTS has universal repairability and thus $\kappa = 1$.

Finally, we consider the computation required for the enrolment RTS. The share and recover algorithms of the enrolment RTS are identical to the share and recover algorithms of Shamir's threshold scheme and therefore have the same complexity; the complexity of Shamir's threshold scheme is analysed in Section 3.3.2. In Step 1, the repair algorithm requires each of the t helping players to generate t random values and compute $t-1$ modular additions over $\mathbb{F}_q$. In Step 3, each player must again compute $t-1$ modular additions, and Step 5 requires the repairing player to compute $t-1$ modular additions. So, in total, each helping player must compute $2(t-1)$ modular additions and the repairing player must compute $t-1$ additions. This is a total of $2t^2-t-1$ modular additions.

5.3.1.3 Refining the enrolment RTS: the reduced enrolment RTS

We observe that not all messages in the share-exchange matrix E are necessary to enable P_r to securely repair their share. The enrolment RTS can be refined to require fewer messages being sent, and therefore achieve a lower communication complexity, whilst maintaining the optimal information rate and the security of the enrolment RTS. In fact, we can reduce the number of messages sent so that player P_i does not send P_j a message if $j > i$. We call the resulting scheme the reduced enrolment RTS. After presenting the refined scheme, the primary task is to prove it maintains the security of the enrolment RTS. The reduced enrolment RTS is as follows.

As before, let P_r be the repairing player, and let $P_1, \ldots, P_t$ be the helping players. Let $\varphi_r = f(r)$ be the share belonging to player P_r, where $f(x) \in \mathbb{F}_q[x]$ is a random polynomial of degree at most $t-1$ whose constant term is the secret s. The share φ_r can be expressed as in (5.3.1). The reduced enrolment RTS can be executed as follows:

1. For all $1 \leq i \leq t$, player P_i computes random values $\delta_{j,i}$ for $i \leq j \leq t$, such that
$$\zeta_i \varphi_i = \sum_{j=i}^{t} \delta_{j,i}.$$
2. For all $1 \leq i \leq t$, $i < j \leq t$, player P_i transmits $\delta_{j,i}$ to P_j using a secure channel.
3. For all $1 \leq j \leq t$, player P_j computes
$$\sigma_j = \sum_{i=j}^{t} \delta_{j,i}.$$
4. For all $1 \leq j \leq t$, player P_j transmits σ_j to player P_r using a secure channel.
5. Player P_r computes their share φ_r using the formula
$$\varphi_r = \sum_{j=1}^{t} \sigma_j.$$

Verifying that player P_r computes their share correctly is similar to the verification for the enrolment RTS, as in [97].

We show the reduced enrolment RTS is secure in Theorem 5.3.1. The proof is similar to the proof of the security of the enrolment RTS in [97].

Theorem 5.3.1. *The reduced enrolment RTS is information theoretically secure against a coalition A of strictly fewer than t players.*

Proof. Assume all players act honestly during the protocol.

First, we note that computing the secret, given $t-1$ shares, is equivalent to computing any additional share. This is easy to see, because any t shares allow the secret to be computed, and any $t-1$ shares along with the secret allow any other share to be computed (this is a well-known property of Shamir's threshold scheme).

As in the proof of the security of the enrolment RTS, there are two cases to consider:

Case 1: The coalition A consists of a subset of $t-1$ players in $\{P_1, \ldots, P_t\}$.

Case 2: The coalition A consists of P_r along with a subset of $t-2$ players in $\{P_1, \ldots, P_t\}$.

We consider the share-exchange matrix of the reduced enrolment RTS. The matrix here is different to the share-exchange matrix of the enrolment RTS in (5.3.5), as player P_i does not send player P_j a message if $j > i$. We can adapt E and enter a '0' into the matrix to denote no message being sent. This means values $d_{j,i} = 0$, if $j > i$. This gives us the following message-exchange matrix:

$$E' = \begin{pmatrix} \delta_{1,1} & 0 & 0 & \ldots & 0 \\ \delta_{1,2} & \delta_{2,2} & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ddots & \ldots \\ \delta_{1,t} & \delta_{2,t} & \delta_{3,t} & \ldots & \delta_{t,t} \end{pmatrix}. \qquad (5.3.6)$$

As before, the sum of the entries in the i^{th} row of E' is equal to $\zeta_i \varphi_i$, the sum of the entries of the j^{th} column is equal to σ_j, and the sum of all the entries in E' is equal to φ_r.

Consider Case 1, where A consists of a subset of $t-1$ players in $\{P_1, \ldots, P_t\}$. Assume player P_i is excluded from the coalition. Now, the coalition possess all entries in E' except for $\delta_{i,i}$. The value $\delta_{i,i}$ is completely random, and knowing this value is equivalent to knowing $\zeta_i \varphi_i$, σ_i or the secret. In fact, in order for any information to be learnt, we can use the properties of E' to deduce the following equations that would need to be solved in order to learn anything about $\delta_{i,i}$.

$$\begin{aligned} \delta_{i,i} - \zeta_i\varphi_i &= w \\ \delta_{i,i} - \sigma_i &= x \\ \zeta_i\varphi_i - \varphi_r &= y \\ \sigma_i - \varphi_r &= z, \end{aligned}$$

where

$$w = \sum_{k=1}^{i-1} \delta_{k,i} \qquad x = \sum_{k=i+1}^{t} \delta_{i,k}$$
$$y = \sum_{k=1,\, k\neq i}^{t} \zeta_k\varphi_k, \qquad z = \sum_{k=1,\, k\neq i}^{t} \sigma_k$$

are all values known to the coalition.

This leads to the following system of equations:

$$\begin{pmatrix} 1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} \delta_{i,i} \\ \sigma_i \\ \zeta_i\varphi_i \\ \varphi_r \end{pmatrix} = \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix}.$$

However, the columns of the matrix on the left are linearly dependent, and thus it is possible to choose any arbitrary value for $\delta_{i,i}$, which will then determine both σ_i and $\zeta_i\varphi_i$, and then φ_r. Therefore, in Case 1, the coalition learns no information about the individual shares $\zeta_i\varphi_i$ or φ_r, and therefore learns no information about the secret being distributed.

Now, consider Case 2, where A consists of P_r and a subset of $t-2$ players in $\{P_1, \ldots, P_t\}$. Assume players P_i and P_j are omitted from the coalition, where $i < j$. In this case, the coalition knows all entries in E' except $\delta_{i,i}, \delta_{i,j}$ and $\delta_{j,j}$. Note that player P_i does not send a message to P_j; this is known by the coalition and so they know that $\delta_{j,i} = 0$. For the coalition, learning $\delta_{i,i}, \delta_{i,j}$ and $\delta_{j,j}$ is equivalent to learning $\zeta_i\varphi_i$ or $\zeta_j\varphi_j$.

As $P_r \in A$ the values σ_i, σ_j and φ_r are known. This knowledge allows the coalition to compute $\delta_{j,j}$, as

$$\begin{aligned}\sum_{k=j}^{t} \delta_{j,k} &= \sigma_j \\ \Rightarrow \delta_{j,j} &= \sigma_j - \sum_{k=j+1}^{t} \delta_{j,k}.\end{aligned}$$

Now, the coalition are left to try to compute $\delta_{i,i}$ and $\delta_{i,j}$. Note that the sum of the these two values are known, but neither value is individually known. The following equations can be formed

$$\begin{aligned}\delta_{i,i} - \zeta_i\varphi_i &= w' \\ \delta_{i,j} - \zeta_j\varphi_j &= x' \\ \delta_{i,i} + \delta_{i,j} &= \sigma_i - y' \\ \zeta_i\varphi_i + \zeta_j\varphi_j &= \varphi_r - z,\end{aligned}$$

where

$$\begin{aligned} w' &= \sum_{k=1}^{i-1} \delta_{k,i} & x' &= \sum_{k=1,\, k\neq j}^{j-1} \delta_{k,j} \\ y' &= \sum_{k=i+1,\, k\neq j}^{t} \delta_{i,k}, & z' &= \sum_{k=i,\, k\neq i,j}^{t} \zeta_k\varphi_k \end{aligned}$$

are all known. This leads to the following system of equations, where all values in the right-hand side vector are known:

$$\begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \delta_{i,i} \\ \delta_{i,j} \\ \zeta_i\varphi_i \\ \zeta_j\varphi_j \end{pmatrix} = \begin{pmatrix} w' \\ x' \\ \sigma_i - y' \\ \varphi_r - z' \end{pmatrix}.$$

As before, the columns in the matrix on the left are linearly dependent, and thus it is possible to choose an arbitrary value for one of $\delta_{i,i}$ or $\delta_{i,j}$, which will determine the other (as the sum of $\delta_{i,i}$ or $\delta_{i,j}$ is known), which will then, in turn, determine values for $\zeta_i\varphi_i$ and $\zeta_j\varphi_j$. Similarly, we could choose arbitrary values for $\zeta_i\varphi_i$ and $\zeta_j\varphi_j$ which would determine $\delta_{i,i}$ and $\delta_{i,j}$.

In either Case 1 or 2, the coalition A of $t-1$ players would be unable to learn any information about any additional share, and thus would learn no information about the secret. $\square$

The reduced enrolment RTS maintains the information rate of the enrolment RTS, $\rho = 1$, yet manages to achieve a lower communication complexity, $\gamma = t(t+1)/2$. The reduced enrolment RTS is also universally repairable, so, as with the enrolment RTS, $\kappa = 1$. Finally, the reduced enrolment RTS has the same share and recover algorithms as the enrolment RTS, and thus has the same computational complexity for these algorithms. However, due to the reduced number of random messages generated by the helping players and the reduced number of messages sent, the repair algorithm is more efficient. Player P_i, for $1 \leq i \leq t$ must generate i random values, rather than t, and must compute $i-1$ modular additions, rather than $t-1$. As in the enrolment RTS, P_r must still compute $t-1$ additions. Therefore, the reduced enrolment protocol requires a total of $t(t+1)/2$ modular additions, rather than the $2t^2 - t - 1$ additions required in the enrolment RTS.

From the analysis, we can observe that the reduced enrolment RTS maintains or improves on all the efficiency metrics and is thus a more efficient RTS than the enrolment RTS presented in [97] and here in Section 5.3.1.1.

5.3.1.4 Optimal communication complexity of the reduced enrolment RTS

As we have reduced the communication complexity of the enrolment RTS, it is a natural question to ask whether it could be reduced any further. We show here that the communication complexity for not only the enrolment RTS, but for any scheme securely computing the sum of shares, is in fact lower bounded by $\gamma = t(t+1)/2$. Thus, in this respect, the reduced enrolment RTS is optimal.

The reduced enrolment RTS is in the setting in which t players wish for an external player to (privately) compute the sum of their t shares. Any set of at most $t-1$ of the $t+1$ players (including the external player P_r) must learn no information about either one of the player's values, or the sum of the shares. In other words, the coalition must be prevented from learning all the inputs to the sum and the output.

All known private protocols, including the reduced enrolment RTS, are *oblivious protocols*. That is, the decision whether player P_i sends a message to P_j in round h is determined by i, j and h, and does not depend on the input and random coins. By assuming a private protocol, we are able to prove the lower bound on the communication complexity whilst making fewer assumptions on the protocol, such as the number of rounds required and the exact number of messages each player either sends or receives.

We prove a lower bound on the number of messages required by any oblivious protocol allowing a $t+1^{\text{th}}$ player to compute the sum of t player's values, such that the protocol is t-private, meaning any subset of at most $t-1$ players is unable to learn all inputs and the output.

Note that the proof of Theorem 5.3.2 proves the same result as in [27]. However, the proof in [27] considers a setting where the sum of the shares is computed by a player who also contributes an input and all players learn the output. Their proof does not immediately apply to our slightly different setting, where an external player with no input is must compute the sum and the output is known only to this external player. As well as achieving slightly different goals, the protocol presented here is executed in two rounds, rather than t rounds (as is achieved in [27]).

Theorem 5.3.2. *The lower bound on the number of messages required to be sent by any oblivious protocol that allows a $t+1^{th}$ player to compute the sum of t player's values, such that the protocol is t-private, is*

$$\binom{t+1}{2} = \frac{t(t+1)}{2}.$$

Proof. Consider a graph with $t+1$ vertices. Let each vertex v_i, for $1 \leq i \leq t$, correspond to a share $\zeta_i \varphi_i$, and let the $t+1^{\text{th}}$ vertex correspond to the sum of the shares. Let the (undirected) edges of the graph correspond to inputs and outputs to the vertices. That is, an edge between vertices i and j means that either player P_i sends player P_j a value, or player P_j sends P_i a value, or both players send values. We claim that knowledge of all inputs and outputs to a vertex will uniquely define the share relating to this vertex. For the $t+1^{\text{th}}$ vertex computing the sum of the inputs, this is obvious. For the other t vertices, we observe that each of these vertices must communicate their share to the $t+1^{\text{th}}$ player, and by learning all inputs and outputs to the vertex, the share must be calculable, otherwise the $t+1^{\text{th}}$ player will not be able to use it as an input to the sum.

We will show that, in order to be secure, the graph must be complete. That is, there must be a total of $\binom{t+1}{2} = t(t+1)/2$ edges.

For the graph to be complete, there must be t edges connected to every vertex, meaning the degree of any vertex v in the graph is t. We show that, if there exists a vertex with degree less than t, a coalition of $t-1$ vertices will be able to learn more information than they should.

To show this, consider a vertex of the graph, v_x which has a degree of less than t. Specifically, let v_x have degree $t-1$. This means there exists no edge between v_x and one other node, which we denote as v_y. We do not need to specify whether either v_x or v_y are players with shares, or the player computing the sum; it does not matter. Now, consider a coalition of $t-1$ vertices A, which consists of all vertices excluding v_x and v_y. As there is no edge between v_x and v_y, all edges connected to these two vertices are also connected to vertices in A. Therefore A knows all inputs and outputs to both v_x and v_y and can thus determine the two vertices v_x and v_y (corresponding to either the shares or the sum). As A already knew the $t-1$ vertices included in the coalition, the additional information of v_x and v_y gives A knowledge of all t shares and the sum of all the shares.

Hence, if there exists a node with degree less than t, the scheme is insecure. Therefore, each node must have degree equal to at least t, meaning the minimum number of edges in the graph, and therefore the minimum number of messages sent during the protocol, is $\binom{t+1}{2} = t(t+1)/2$, as required. □

We have previously defined the reduced enrolment RTS which has a total of $\binom{t+1}{2}$ messages sent throughout. Thus the reduced enrolment RTS is one construction that meets the lower bound for the communication complexity, and is thus optimal in this respect.

5.3.2 Combinatorial repairability

Also in [97], Stinson and Wei propose a way to construct (t, n, d)-RTSs based on combinatorial designs. These schemes achieve a reasonably high (but not optimal) information rate, and a low communication complexity. However, these schemes only achieve restricted repairability, as in Definition 5.1.2.

We first present the construction of these schemes, then provide an example construction as an illustration. We then analyse the efficiency of these schemes; in particular, we apply our new metric measuring the repairability to these schemes.

5.3.2.1 Definition of scheme

The share algorithm is as follows. Suppose there exists an (m, d, λ)-BIBD with b blocks, as in Definition 5.2.2, which is also a repairable (t, ℓ_1, ℓ_2)-distribution design, as in Definiton 5.2.5. Now, use an (ℓ_1, ℓ_2, m)-ramp scheme defined over $\mathbb{F}_q$ (for $q \geq m+1$) and call the shares output by the ramp scheme *sub-shares*. Let $2m/d \leq n \leq b$, where the lower bound originates from the minimal size of the basic repairing set in (5.2.7) and b is the number of blocks in the BIBD. In [97], they construct a (t, n, d)-RTS with restricted repairability by allocating sub-shares from the ramp scheme to n players, as defined in the distribution design. Each block in the the design represents a player, and each point represents a sub-share. The recover algorithm requires t players to pool their shares and recover the shared secret via the recover algorithm of the (ℓ_1, ℓ_2, m)-ramp scheme. For the repair algorithm, a repairing player P_r must be sent d sub-shares from d players who each store one of P_r's sub-shares.

We will refer to schemes constructed in this manner as combinatorial RTSs. We illustrate this construction via an example.

Example 5.3.1. The share algorithm of a $(2, 12, 3)$-RTS is as follows. Consider the $(9, 3, 1)$-BIBD, which is a repairable $(2, 3, 5)$-distribution design, used throughout the examples in Section 5.2.1. This design consists of 12 blocks; note that $n = 12$ and so n has been chosen to be maximal. Also consider a $(3, 5, 9)$-ramp scheme over $\mathbb{F}_q$ and, for convenience, label the nine shares output from the ramp scheme $1, 2, \ldots, 9$. Using the $(2, 3, 5)$-repairable distribution design and the $(3, 5, 9)$-ramp scheme, construct a $(2, 12, 3)$-RTS with restricted repairability by allocating sub-shares from the ramp scheme to the 12 players defined by the design, as follows:

$$\begin{array}{cccc}
P_1 \leftarrow \{1,2,3\} & P_2 \leftarrow \{4,5,6\} & P_3 \leftarrow \{7,8,9\} & P_4 \leftarrow \{1,4,7\} \\
P_5 \leftarrow \{2,5,8\} & P_6 \leftarrow \{3,6,9\} & P_7 \leftarrow \{1,5,9\} & P_8 \leftarrow \{2,6,7\} \\
P_9 \leftarrow \{3,4,8\} & P_{10} \leftarrow \{1,6,8\} & P_{11} \leftarrow \{2,4,9\} & P_{12} \leftarrow \{3,5,7\}
\end{array}$$

To recover the secret, any two players can pool their shares, which will consist of at least five distinct sub-shares from the $(3, 5, 9)$-ramp scheme, and recover the distributed secret via the recover algorithm of the ramp scheme. Say player P_5 needs to repair their share. They can have assistance from players P_1, P_2 and P_3, who would each send the sub-shares $2, 5$ and 8, respectively.

In their paper, Stinson and Wei propose a number of combinatorial RTSs relying on a range of designs. They give some specific parameters for the underlying designs and the resulting RTS.

Note that these RTS schemes are secure due to the underlying properties on the (ℓ_1, ℓ_2, m)-ramp scheme and the (t, ℓ_1, ℓ_2)-distribution design: a coalition of t players will learn at most ℓ_1 sub-shares output by the ramp scheme and will therefore learn no information about the secret.

5.3.2.2 Analysis of combinatorial RTSs

A theorem in [97] states the information rate and communication complexity of combinatorial RTSs. Assume there exists an (m, d, λ)-BIBD, which is a repairable (t, ℓ_1, ℓ_2)-distribution design with b blocks that contains a basic repairing set of size y, and suppose that $q \geq m + 1$. Let $y \leq n \leq b$. Then there exists a (t, n, d)-RTS with restricted repairability that has information rate $\rho = (\ell_2 - \ell_1)/d$, and communication complexity $\gamma = d/(\ell_2 - \ell_1)$, where every share is in $(\mathbb{F}_q)^d$.

Before calculating the repairability of the scheme, we note one disadvantage of the combinatorial RTSs that arises from the necessary condition in the aforementioned theorem: a (t, d, n)-RTS can only be constructed if there exists an (m, d, λ)-BIBD which is also a repairable (t, ℓ_1, ℓ_2)-distribution design on m points with n blocks of size d. This is not true for all possible parameters t, d and n, and so this construction may not be able to build an RTS with the desired parameters.

So, we are left to calculate the repairability κ of the combinatorial RTSs and analyse the computational complexity.

Combinatorial RTSs only have restricted repairability, so not all d-subsets of players will have the information required to help a repairing player reconstruct their share. The probability that a randomly chosen set of d players can help a repairing player repair their share is described in Theorem 5.3.3.

Theorem 5.3.3. *A randomly chosen subset of d players in a (t, n, d)-RTS, constructed using an underlying $(m, d, 1)$-BIBD with $n = b$ players, has probability*

$$\kappa = \frac{(\tau - 1)^d}{\binom{n-1}{d}}$$

of successfully repairing the share of a player P_r, where τ is the replication number of the BIBD, as defined in Theorem 5.2.3.

Proof. Assume an (m, d, λ)-BIBD which is also a repairable (t, ℓ_1, ℓ_2)-distribution design on m points with n blocks of size d. Say player P_r wishes to repair their share, which consists of d sub-shares. There are a potential $n - 1$ players that could play the role of helping nodes and assist P_r in reconstructing their share. As d of these players are required to repair P_r's share, there are a total of $\binom{n-1}{d}$ d-subsets that could collaborate to help P_r. Now, we have to calculate how many of the $\binom{n-1}{d}$ d-subsets have the information required to repair P_r's share. Because of the properties of the underlying (m, d, λ)-BIBD, each sub-share occurs in exactly τ blocks, where τ is the replication number of the BIBD, calculated here as $\tau = (m - 1)/(t - 1)$. Therefore, excluding P_r's share, each sub-share is contained in $\tau - 1$ of the $n - 1$ players' shares. Now, as each pair of sub-shares occurs in one player's share, each of the d helping players must send P_r exactly one sub-share. There are $\tau - 1$ players who could contribute to recovering P_r's first sub-share, then a distinct group of $\tau - 1$ players who could contribute to recovering P_r's second sub-share, and so on, until the d^{th} sub-share. Therefore, there are $(\tau - 1)^d$ d-subsets that collectively hold the information required to help P_r recover their share. Therefore, κ is given by dividing the number of d-subsets that could successfully act as helping players, $(\tau - 1)^d$, by the total number of possible d-subsets, $\binom{n-1}{d}$, as required. □

We illustrate this proof by continuing Example 5.3.1.

Example 5.3.2. Consider the $(2, 12, 3)$-RTS in Example 5.3.1, constructed from an underlying $(9, 3, 1)$-BIBD. The replication number of the BIBD is $\tau = 4$. Each player has $d = 3$ sub-shares and each sub-share is stored by $\tau - 1 = 3$ players, excluding the repairing player. So, there are three players that can send the repairing player the first sub-share, three other players that can send the second sub-share and three other players that can send the final sub-share. This means there are $3^3 = 27$ sets of three players, out of the possible $\binom{n-1}{d} = \binom{11}{3} = 165$ 3-subsets, that have the information required to repair P_r's share. So

$$\kappa = \frac{(r-1)^d}{\binom{n-1}{d}} = \frac{27}{165} = 0.163636.$$

So, any randomly chosen set of $d = 3$ players will have a 16.3636% chance of having the information required to help the repairing player. To further illustrate this, assume P_5 is the repairing player. There are 27 sets of players that can help repair P_5's share, including $\{P_1, P_2, P_3\}, \{P_7, P_8, P_9\}$ and $\{P_2, P_{10}, P_{11}\}$. There are now $165 - 27 = 138$ sets of three players that do not have the information required to help repair P_5's share, including $\{P_4, P_6, P_7\}$, who do not know and so cannot send sub-shares 5 and 8, and $\{P_1, P_8, P_{12}\}$, who do not know sub-share 8.

We make one final comment on the reparability of the combinatorial RTSs. In [97], it is not necessary to have the number of players in the scheme n to be equal to the number of blocks b. Instead, they bound n to be $y \leq n \leq b$, where y is the size of the basic repairing set and b is the number of blocks in the design. If $n < b$, the repairability of the scheme will vary depending on the number of players, and which players, are in the RTS.

Example 5.3.3. Consider a $(2, 6, 3)$-RTS constructed from the basic repairing set in Example 5.2.4. Assume P_r wishes to repair their share. There are a possible $\binom{5}{3} = 10$ subsets of the remaining five players that could act as helping players. Of these 10 subsets, only one set has the information required to repair P_r's share. Therefore, $\kappa = 0.1$. So, a randomly chosen set of three players will have a 10% probability of being able to recover the repairing player's share.

From now on, when computing the reparability of a combinatorial RTS, we assume the number of players is maximal, so $n = b$. In Section 5.5.2, Table 5.2 shows the reparability (as well as the information rate and communication complexity) of each of the (t, n, d)-RTSs proposed in [97].

We complete the analysis of the combinatorial RTSs by commenting on the complexity of each of the RTS algorithms. Once a (t, ℓ_1, ℓ_2)-distribution design has been chosen, the RTS share algorithm requires one execution of the share algorithm of the ramp scheme.

Similarly, the RTS recover algorithm requires one execution of the recover algorithm of the ramp scheme. The RTS repair algorithm requires no computation: helping players must send sub-shares to the repairing player, but no players are required to conduct any computation. In this respect, the repair algorithm is optimal.

5.3.3 GLF scheme

In [90], the authors (Guang, Lu and Fu) present an information theoretically secure (t, n, d)-RTS which utilises MBR codes and linearised polynomials. Intuitively, they distribute a secret via a (t, t)-threshold scheme using a random, linearised polynomial with the constant term equal to the secret, then the shares are treated as the message symbols in a (t, n, d)-MBR code. Their scheme works for all parameters n, t and d.

The GLF construction achieves an information rate of $\rho = 1/d\beta$ and a communication complexity of $\gamma = d\beta$, where β is as defined by the MBR code. The GLF RTS is universally repairable and so $\kappa = 1$.

The share algorithm of the GLF RTS requires the generation of a linearised polynomial and the evaluation of t points on this polynomial; this is followed by the computation of the necessary MBR code, where the shares are treated as message symbols. The GLF recover algorithm requires recovery of the message symbols via the MBR code, then recovering the linearised polynomial from the message symbols. The repair algorithm is identical to the regeneration of a node in the underlying MBR code.

We will see how, when considering secure regeneration codes in Section 5.4, MBR codes can be used to achieve schemes with a better information rate and communication complexity than is achieved by the GLF RTS.

5.4 Solutions using regenerating codes

Secure regenerating codes, defined in Definition 5.2.10, can be directly used to construct (t, n, d)-RTSs. However, the work discussing secure regenerating codes has not been presented in the framework of threshold schemes and RTSs. In this section, we briefly explore the application of secure regenerating codes to RTSs and discuss relevant parameters. Following this, we present a number of constructions for secure regenerating codes.

5.4.1 Applying regenerating codes to RTSs

Now, we translate the language used in regenerating codes into that used by repairable threshold schemes. Each node is equivalent to a player, and the data stored by the node is the player's share. A regenerating node is equivalent to a repairing player. The strongest adversary considered in an RTS is equivalent to an (ℓ_1, ℓ_2)-adversary against a secure regenerating code, where $\ell_1 = 0$ and $\ell_2 = t-1$. So, if we wish to build a (t, n, d)-RTS, we can trivially use an information theoretically secure (n, t, d)-regenerating code.

In general, the information rate of a (t, n, d)-RTS based on a regenerating code is

$$\rho = \frac{B^{(s)}}{\alpha},$$

and the communication complexity is

$$\gamma = \frac{d\beta}{B^{(s)}}.$$

As all regenerating codes have universal repairability, all (t, n, d)-RTSs based on MBR codes have $\kappa = 1$.

Note that, because $\alpha = d\beta$ for all MBR codes, the communication complexity and information rate of these schemes are reciprocals, as was noted in [97] when discussing the combinatorial RTSs.

Prior to considering constructions of secure regenerating codes, we consider implications of the bounds given in Theorems 5.2.11 and 5.2.12 when considered in the setting of RTSs.

5.4.1.1 Using MBR codes as RTSs

The first corollary considers the information rate of an RTS based on an MBR code.

Corollary 5.4.1. *A (t, n, d)-RTS based on a secure (n, t, d)-MBR code with a $(0, t-1)$-*

adversary cannot be ideal.

Proof. The information rate of the RTS is calculated as $\rho = B^{(s)}/\alpha$. The scheme is ideal if $\rho = 1$, which is true if and only if $B^{(s)} = \alpha$. Now, substitute the MBR point from (5.2.3) and (5.2.4) into the bound given in Theorem 5.2.11. This gives the maximum number of messages symbols that can be securely distributed by an MBR code (as in [90]) to be

$$B^{(s)} = \left(td - \binom{t}{2}\right)\beta - \left(\ell_2 d - \binom{\ell_2}{2}\right)\beta. \tag{5.4.1}$$

By substituting in $\ell_2 = t-1$, we learn that $B^{(s)} = \beta(d-t+1)$. So, $B^{(s)} = \alpha$ if and only if $\beta(d-t+1) = \alpha$. But in all MBR codes, $\alpha = d\beta$, and so $d-t+1 = d$. This is only true if $t = 1$. However, in the definition of threshold schemes, given in Definition 2.3.2, t is defined to be such that $t \geq 2$ as, if $t = 1$, any individual player could recover the secret. Therefore, a (t,n,d)-RTS constructed from a secure (n,t,d)-MBR code cannot be ideal. □

5.4.1.2 Using MSR codes as RTSs

The following two corollaries consider the limitations of RTSs based on MSR codes.

Corollary 5.4.2. *A (t,n,d)-RTS based on an (n,t,d)-MSR code cannot securely distribute any messages if $d = t$.*

Proof. Consider the bound given in Theorem 5.2.12. By setting $d = t$, we can see immediately that $B^{(s)} \leq 0$. □

Corollary 5.4.3. *A (t,n,d)-RTS based on an (n,t,d)-MSR code with a $(0,t-1)$-adversary cannot be ideal. In fact, if $\ell_1 + \ell_2 = t-1$, the RTS can only be ideal against a $(t-1,0)$-adversary.*

Proof. Assume we have an optimal (in terms of $B^{(s)}$), secure (n,t,d)-MSR code. The information rate of the (t, n, d)-RTS based on this MSR code is

$$\begin{aligned}\rho &= \frac{B^{(s)}}{\alpha} \\ &= (t - \ell_1 - \ell_2)\left(1 - \frac{1}{d-t+1}\right)^{\ell_2}. \end{aligned} \tag{5.4.2}$$

As we have assumed $\ell_1 + \ell_2 = t - 1$, we can substitute this into (5.4.2), so

$$\rho = \left(1 - \frac{1}{d-t+1}\right)^{\ell_2}. \tag{5.4.3}$$

For the (t, n, d)-RTS to be ideal, ρ must equal 1. This is true if and only if $\ell_2 = 0$, as required. □

In fact, (5.4.3) illustrates how the information rate of a (t, n, d)-RTS based on an MSR code with an (ℓ_1, ℓ_2)- adversary, such that $\ell_1 + \ell_2 = t - 1$, decreases as ℓ_2 increases. This is because, as previously explained, the repair algorithm for MSR codes leaks information as $\alpha < \beta d$. Because of this, MSR codes may not be the best way to construct RTSs as they achieve a small information rate when considering a maximal adversary with $\ell_1 = 0$ and $\ell_2 = t - 1$.

Now, we consider secure regenerating codes as RTSs and analyse results.

5.4.2 Constructions of secure regenerating codes for RTSs

We consider three constructions for secure regenerating codes. The first of the three, [90], is a secure MBR code and can be constructed for all parameters t, n, d such that $t \leq d < n$. The next two constructions we consider, from [83] and [90], are secure MSR constructions and are for specified values of d.

5.4.2.1 SRK's secure MBR construction [90]

In [90], the authors (Shah, Rashmi and Kumar) present an information theoretically secure (n, t, d)-MBR code based on a matrix product construction for MBR codes, presented in [82] and used in Example 5.2.5. Without loss of generality, they construct codes for the case where $\beta = 1$, and codes for any higher value of β can be obtained by a simple concatenation of the code with $\beta = 1$ (this technique is called *striping* and is detailed in [82]). We call this construction the SRK-MBR construction.

The SRK-MBR construction achieves a secure code by replacing a carefully chosen set of $B - B^{(s)}$ message symbols with symbols which are chosen uniformly and independently at random from $\mathbb{F}_q$. If these random values are treated as message symbols, the secure regenerating code is identical to the standard regenerating code, and so distribution, recovery and repair are as in the regeneration code defined in [82]. They prove the secure construction is information theoretically secure. The SRK-MBR scheme achieves the maximum bound for $B^{(s)}$, as in Theorem 5.2.11, with $\beta = 1$.

Consider this secure (n, t, d)-regenerating code as a (t, n, d)-RTS, and consider a $(0, t-1)$-adversary. As the repair algorithm of an MBR code is inherently secure, a $(0, t-1)$-adversary is equivalent to a $(t-1, 0)$-adversary. We can substitute $\ell_2 = 0$ and $\beta = 1$ into (5.4.1) to calculate

$$\begin{aligned} B^{(s)} &= \left(td - \binom{t}{2}\right) - \left((t-1)d - \binom{t-1}{2}\right) \\ &= d - t + 1. \end{aligned}$$

Now, as each player stores α elements in $\mathbb{F}_q$, where $\alpha = d\beta$ is as in (5.2.4), and as $\beta = 1$, the information rate and communication complexity metrics can be computed as follows:

$$\begin{aligned} \rho &= \frac{d - t + 1}{d}, \\ \gamma &= \frac{d}{d - t + 1}. \end{aligned}$$

As regenerating codes have universal repairability, $\kappa = 1$ for all secure regenerating codes treated as an RTS.

We present a brief example of this secure construction, which is a continuation from Example 5.2.5.

Example 5.4.1. Consider the code in Example 5.2.5. The number of symbols that can be securely distributed via this $(5, 2, 3)$-MBR code is $B^{(s)} = 2$. Replace u_1, u_2 and u_4 with random elements in the field, and let u_3 and u_4 be the secure message symbols to be distributed. This is then a secure $(5, 2, 3)$-RTS against a $(1, 0)$-adversary and has information rate $\rho = 2/3$, communication complexity $\gamma = 3/2$ and repairability $\kappa = 1$.

Finally, we briefly comment on the complexity of the scheme. All three algorithms, share, repair and recover, require all players to compute linear computations.

5.4.2.2 Rawat's MSR construction [83]

In [83], Rawat proposes an information theoretically secure (n, t, d)-MSR code for $d = n-1$. The secure code is based on a construction for general MSR codes proposed in [100] which is valid for all parameters n, t and d. Rawat's MSR construction can be treated as a $(t, n, n-1)$-RTS with a $(0, t-1)$-adversary.

Rawat's construction is optimal with respect to $B^{(s)}$, as given in Theorem 5.2.12. Thus, by substituting in $d = n - 1$, the information rate of the scheme can be calculated as

$$\rho = \frac{B^{(s)}}{\alpha} = \left(1 - \frac{1}{n-t}\right)^{t-1}.$$

To calculate the communication complexity, it is useful to calculate $d\beta$ using the value for β and $B = \alpha t$, given in the MSR trade-off point. This gives that

$$d\beta = \frac{dB}{t(d-t+1)} = \frac{\alpha t(n-1)}{t(n-t)} = \frac{\alpha(n-1)}{(n-t)},$$

where α is defined to be $\alpha = (n-t)^{n-1}$ in the scheme. Then,

$$d\beta = \frac{(n-t)^{n-1}(n-1)}{n-t} = (n-t)^{n-2}(n-1).$$

Then the communication complexity of the scheme is

$$\gamma = \frac{d\beta}{B^{(s)}} = \frac{(n-t)^{n-2}(n-1)}{\left(1 - \frac{1}{n-t}\right)^{t-1}(n-t)^{n-1}} = \frac{(n-1)}{\left(1 - \frac{1}{n-t}\right)^{t-1}(n-t)}.$$

As before, $\kappa = 1$. Also, as with other schemes based on regenerating codes, all three algorithms, share, repair and recover, require computations of linear combinations.

5.4.2.3 SRK's secure MSR construction [90]

In [90], the authors present a secure (t, n, d)-RTS based on MSR codes, for $d = 2t-2$. We call this the SRK-MSR construction. SRK-MSR is similar to SRK-MBR and is also based on the constructions given in [82], which are suitable for when $d = 2t-2$ (they say the schemes can be extended so $d > 2k-2$ via shortening), and thus the SRK-MBR is also for parameters $d = 2t-2$. As in SRK-MBR, SRK-MSR consists of replacing a carefully selected subset of the message symbols with random values.

The SRK-MSR scheme is able to distribute $B^{(s)} = (t-\ell_1-\ell_2)(\alpha-\ell_2\beta)$ messages securely. The authors claim their scheme is optimal when $\ell_2 = 0$, but say it is unknown whether it is optimal when $\ell_2 > 0$. We answer here that the MSR scheme is optimal if $\ell_2 = 0$ or 1, but is not optimal for $\ell_2 > 1$.

Corollary 5.4.4. *The (t, n, d)-RTS constructed from the SRK-MSR construction presented in [90] is optimal with respect to the number of messages that can be securely distributed, $B^{(s)}$, if $\ell_2 \leq 1$. If $\ell_2 > 1$, the construction is not optimal with respect to $B^{(s)}$.*

Proof. Denote the number of message symbols the SRK-MSR construction in [90] can securely distribute as

$$B^{(s)}_{SRK} = (t-\ell_1-\ell_2)(\alpha-\ell_2\beta).$$

We can substitute in the values for $\beta = B/t(d-t+1)$ and $B = \alpha t$, given at the MSR trade-off point in (5.2.5) and (5.2.6), to get

$$\begin{aligned} B^{(s)}_{SRK} &= (t-\ell_1-\ell_2)\left(\alpha - \frac{\ell_2\alpha t}{t(d-t+1)}\right) \\ &= (t-\ell_1-\ell_2)\alpha\left(1-\frac{\ell_2}{(d-t+1)}\right). \end{aligned}$$

Then we can compare this to the bound $B^{(s)}$ for all MSR codes given in Theorem 5.2.12. If we divide both values by $\alpha(t-\ell_1-\ell_2)$, we can see that

$$\begin{aligned} \frac{B^{(s)}_{SRK}}{\alpha(t-\ell_1-\ell_2)} &= 1-\frac{\ell_2}{d-t+1} \\ &\leq \left(1-\frac{1}{d-t+1}\right)^{\ell_2} = \frac{B^{(s)}}{\alpha(t-\ell_1-\ell_2)}, \end{aligned}$$

with an equality when ℓ_2 equals either zero or one, but strictly greater than when $\ell_2 > 1$, as required. □

Consider the SRK-MSR construction as a $(t, n, 2t-2)$-RTS with a $(0, t-1)$-adversary. We can substitute $\beta = 1$ and $d = 2t-2$ into the MSR point to give that $B = \alpha(\alpha+1)$ and $\alpha = t-1$, so $d = 2\alpha$. Now, we can calculate, $B^{(s)} = \alpha - (t-1) = (t-1) - (t-1) = 0$. Therefore the SRK-MSR construction cannot be used to securely distribute any symbols when a $(0, t-1)$-adversary is considered.

5.5 Comparison of techniques

In this section, we compare the RTS constructions introduced throughout this chapter. We begin by comparing MBR and MSR based schemes. Then, we compare schemes that prioritise communication complexity above information rate, then schemes prioritising information rate.

5.5.1 Comparing MBR and MSR codes

Here, we highlight some similarities and differences between (t, n, d)-RTSs that are constructions of secure MBR and MSR codes.

Firstly, all (t, n, d)-RTSs based on either MBR or MSR codes have universal repairability, so $\kappa = 1$, always.

However, there are a number of major differences. MBR codes prioritise bandwidth and thus the (t, n, d)-RTSs based on them achieve a lower communication complexity than schemes based on MSR codes. In contrast, MSR codes prioritise storage and thus the (t, n, d)-RTSs based on MSR codes generally achieve higher information rates. However, secure MSR codes cannot be ideal RTSs unless $\ell_2 = 0$, meaning the adversary witnesses no regenerations, which may be unrealistic in the RTS setting.

Importantly, the repair algorithm for RTSs based on MBR codes is secure: the adversary is able to witness any number of regenerations and no information will be learnt. However, crucially, the repair algorithm for RTSs based on MSR codes is insecure: with each distinct regeneration, the adversary learns more information. Thus, the number of regenerations the adversary can witness affects the number of message symbols that can be securely distributed; the more regenerations witnessed, the fewer message symbols can be secured. In settings where $\ell_2 = t - 1$, which is what is considered here for an RTS, MSR codes may not be very useful.

Finally, secure MBR based (t, n, d)-RTSs exist for all valid parameters n, t and d. In the current literature, there does not appear to be any secure construction based on MSR codes for all valid parameters, and a secure construction for $d = t$ is impossible.

Thus, between MBR based and MSR based RTSs, MBR based schemes appear to be the most applicable to RTSs, mainly because of the secure repair algorithm. In particular, the secure MBR construction presented in [90] appears to be the most applicable construction from the field of regenerating codes. This is because it achieves the upper bound for the value of $B^{(s)}$ and is therefore optimal and the repair algorithm is secure, meaning the adversary can witness multiple repairs.

Table 5.1 compares the information rate and communication complexity for all universally repairable schemes considered for an example set of parameters. Due to the restrictions of the repairing degree d for the secure MSR constructions, the table considers the metrics for a $(4, 6, 6)$-RTS using the SRK-MBR [90], Rawat [83], SRK-MSR [90] and GLF [44] constructions. The metrics for a $(4, 7, 4)$-RTS from the enrolment and reduced enrolment RTSs are also included. It is possible to see from Table 5.1 that, out of all RTSs based on regenerating codes, the SRK-MBR construction achieves the best information rate and the best communication complexity for the given parameters.

(t, n, d)-RTS	Construction	ρ	γ	κ
$(4, 7, 6)-$RTS	SRK-MBR [90]	1/2	2	1
	Rawat [83]	8/27	27/4	1
	SRK-MSR [90]	*Not possible*		
	GLF [44]	1/6	6	1
$(4, 7, 4)-$RTS	Enrolment [97]	1	16	1
	Reduced Enrolment	1	10	1

Table 5.1: Comparing metrics for universally repairable RTS constructions.

	Combinatorial Schemes [97]					**MBR Schemes** [90]		
(t,n,d)-RTS	$(m,d,1)$-BIBDs	n	ρ	γ	κ	ρ	γ	κ
$(2,n,3)$	$(9,3,1)$	$6 \leq n \leq 12$	$2/3$	$3/2$	0.1636	$2/3$	$3/2$	1
$(2,n,3)$	$(15,3,1)$	$10 \leq n \leq 35$	$2/3$	$3/2$	0.0361	$2/3$	$3/2$	1
$(2,n,3)$	$(21,3,1)$	$14 \leq n \leq 70$	$2/3$	$3/2$	0.0139	$2/3$	$3/2$	1
$(2,n,4)$	$(16,4,1)$	$8 \leq n \leq 20$	$3/4$	$4/3$	0.0060	$3/4$	$4/3$	1
$(2,n,4)$	$(28,4,1)$	$14 \leq n \leq 63$	$3/4$	$4/3$	0.0073	$3/4$	$4/3$	1
$(2,n,4)$	$(40,4,1)$	$20 \leq n \leq 130$	$3/4$	$4/3$	0.0019	$3/4$	$4/3$	1
$(2,n,5)$	$(25,5,1)$	$10 \leq n \leq 30$	$4/5$	$5/4$	0.0263	$4/5$	$5/4$	1
$(3,n,5)$	$(25,5,1)$	$10 \leq n \leq 30$	$2/5$	$5/2$	0.0263	$3/5$	$5/3$	1
$(2,n,5)$	$(65,5,1)$	$26 \leq n \leq 208$	$4/5$	$5/4$	0.0003	$4/5$	$5/4$	1
$(3,n,5)$	$(65,5,1)$	$26 \leq n \leq 208$	$2/5$	$5/2$	0.0003	$3/5$	$5/3$	1
$(2,n,8)$	$(64,8,1)$	$16 \leq n \leq 72$	$7/8$	$8/7$	0.0016	$7/8$	$8/7$	1
$(3,n,8)$	$(64,8,1)$	$16 \leq n \leq 72$	$5/8$	$8/5$	0.0016	$3/4$	$4/3$	1
$(4,n,8)$	$(64,8,1)$	$16 \leq n \leq 72$	$1/4$	4	0.0016	$5/8$	$8/5$	1
$(2,n,4)$	$(13,4,1)$	$9 \leq n \leq 13$	$3/4$	$4/3$	0.0136	$3/4$	$4/3$	1
$(3,n,4)$	$(13,4,1)$	$9 \leq n \leq 13$	$1/2$	2	0.0136	$1/2$	2	1
$(2,n,5)$	$(21,5,1)$	$12 \leq n \leq 21$	$4/5$	$5/4$	0.0660	$4/5$	$5/4$	1
$(3,n,5)$	$(21,5,1)$	$12 \leq n \leq 21$	$3/5$	$5/3$	0.0660	$3/5$	$5/3$	1
$(4,n,5)$	$(21,5,1)$	$12 \leq n \leq 21$	$1/5$	5	0.0660	$2/5$	$5/2$	1
$(2,n,6)$	$(31,6,1)$	$15 \leq n \leq 31$	$5/6$	$6/5$	0.0263	$5/6$	$6/5$	1
$(3,n,6)$	$(31,6,1)$	$15 \leq n \leq 31$	$2/3$	$3/2$	0.0263	$2/3$	$3/2$	1
$(4,n,6)$	$(31,6,1)$	$15 \leq n \leq 31$	$1/3$	3	0.0263	$1/2$	2	1

Table 5.2: Comparison of $(t,n,d)-$RTSs based on $(m,d,1)-$BIBDs, as in [97], and secure MBR codes, as in [90].

5.5.2 Comparison of techniques prioritising communication complexity

Both secure MBR schemes and combinatorial RTSs prioritise communication complexity over information rate. Here, we compare the SRK-MBR construction with the combinatorial RTSs given in [97].

Table 5.2 shows how the combinatorial RTSs in [97] compare to those based on secure MBR codes in [90] for certain parameters. The comparison shows the information rate ρ, communication complexity γ and repairability κ (assuming n is maximal for the combinatorial schemes), with highlighted rows showing the RTSs with different ρ and γ. The chosen parameters relate to proposed combinatorial RTSs in [97]; note that MBR schemes exist for all valid parameters of n, t and d, but the combinatorial RTSs rely on the existence of an underlying design with relevant parameters.

From Table 5.2, we can see the schemes achieve equal information rate and communication complexities in most cases. In some cases, which have been highlighted, the SRK-MBR scheme achieves a better information rate and a better communication complexity than the combinatorial RTSs. In no case do the combinatorial RTSs achieve better results than the SRK-MBR construction. In fact, when the concept of restricted repairability was introduced in [97], it was suggested that compromising repairability may enable more efficient schemes. However, this is not the case.

As well as achieving similar or better values for ρ and γ in all defined parameters for t and d in [97], the SRK-MBR construction has two advantages over the combinatorial RTSs:

1. The MBR schemes in [90] achieve universal, rather than restricted, repairability.
2. The combinatorial RTSs in [97] depend on the existence of certain combinatorial constructions. This is in contrast to MBR schemes, which can be constructed for all valid parameters n, t and d.

However, one advantage of the combinatorial RTSs in [97] is the computational complexity of repairing a share. In schemes based on either MBR or MSR codes, every repair requires helping nodes and the repairing node to execute linear computations. The combinatorial RTSs, in contrast to this, just require the helping nodes to send the repairing node a sub-share, with no computation being required for any of the players.

5.5.3 Comparison of techniques prioritising information rate

Both schemes based on MSR codes and the enrolment and reduced enrolment RTS prioritise information rate and offer universal repairability.

However, because of the insecure repair protocol, secure MSR codes are unable to achieve an information rate as good as the reduced enrolment RTS. Even though the communication complexity of the enrolment RTS was reduced in the reduced enrolment RTS, the communication complexity remains much higher than any of the other schemes.

Chapter 6

Localised multi-secret sharing schemes

Contents

This chapter is based on joint work with Keith Martin, Maura Paterson and Doug Stinson and is published in [63].

6.1 Introduction

A *multi-secret sharing scheme* is a generalisation of a secret sharing scheme in which several secrets are shared according to different access structures on the same set of players [13, 14, 47, 55, 72]. In this chapter, we consider a specific class of multi-secret sharing schemes, which we call *localised threshold multi-secret sharing schemes*, that are suited to an application in RFID security proposed by Juels, Pappu and Parno in USENIX 2008 [56].

We motivate the definition of localised threshold multi-secret sharing schemes initially through the following toy example:

Example 6.1.1. A learned society is lead by a committee with seven members. The members each serve a seven year term on the committee. Each year one member leaves the committee and a new member is elected to replace them. Every year the society holds a conference in a different city and committee meetings occur at these conferences. A meeting is deemed quorate as long as at least three committee members are present.

The society wishes to distribute a signing key among the committee members that allows them to sign the reports of their meetings at which at least three members are present. The key will need to change each year to reflect the changed membership of the committee. However, they wish to avoid the need to change the members' shares, since the shares are handed out at the committee meetings, but not every member attends each meeting. Furthermore, the shares belonging to any members who have left the committee should not reveal any information about the current value of the committee's signing key.

With this example as our initial motivation, we first introduce multi-secret sharing schemes, then define localised threshold multi-secret sharing schemes (LMSS). In Section 6.3 we provide theoretical results on the security properties of LMSSs, including bounds on the share sizes, and give constructions that meet these bounds. Section 6.4 considers how the security definition of an LMSS can be relaxed in order to permit schemes with smaller share sizes. Finally, in Section 6.5, we show how these ideas can be applied in a natural way when designing a scheme suitable for application in RFID enabled supply chain management, motivated by a proposal of Juels, Pappu and Parno [56].

6.2 Localised threshold multi-secret sharing schemes

6.2.1 Multi-secret sharing

Many authors have studied a generalisation of secret sharing in which several secrets are shared according to different access structures on the same set of participants [13, 14, 47, 55, 72]. There are various equivalent ways of defining security in such a setting; the following is due to Herranz, Ruiz and Sáez [47]:

Definition 6.2.1. *Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of n players. For $i = 1, 2, \dots, \ell$ let Γ_i be a monotone access structure on $\mathcal{P}$. Let ℓ secrets $s_1, s_2, \dots, s_\ell$ be chosen from some secret space $\mathcal{S}$. A scheme that allocates to each player $P_i \in \mathcal{P}$ a share v_i from some set V of potential shares is a* weak information theoretic secure multi-secret sharing scheme *if it satisfies the following properties:*

- (correctness) *for $i = 1, 2, \dots, \ell$ and for any set $A \subseteq \mathcal{P}$ of players we have that if $A \in \Gamma_j$ then the shares of the players in A can be used to recover the secret s_j. In terms of entropy,* $\mathrm{H}(\boldsymbol{\mathcal{S}_j} \mid \mathbf{A}) = 0$ *whenever $A \in \Gamma_j$.*
- (weak information theoretic security) *if $A \notin \Gamma_j$ then the shares of the players in A reveal no information about s_j. That is,* $\mathrm{H}(\boldsymbol{\mathcal{S}_j} \mid \mathbf{A}) = \mathrm{H}(\boldsymbol{\mathcal{S}_j})$.

If, in addition, any set $A \notin \Gamma_j$ together with a set T of secrets with $s_j \notin T$ reveals no information about s_j that is not already revealed by T alone (i.e. if $\mathrm{H}(\boldsymbol{\mathcal{S}_j} \mid \mathbf{A}, \mathbf{T}) = \mathrm{H}(\boldsymbol{\mathcal{S}_j} \mid \mathbf{T})$*) then the scheme is said to have* strong information theoretic security.

We present the following theorem from [47], without proof, which will be used later in our analysis of LMSSs.

Theorem 6.2.2. *Let $\Gamma_1, \dots, \Gamma_\ell$ be ℓ access structures on $\mathcal{P}$ and consider player $P_i \in \mathcal{P}$. Assume there exists subsets of players $A_1 \subset A_2 \subset \dots \subset A_\ell \subset \mathcal{P} \setminus \{P_i\}$ satisfying, for all $j = 1, 2, \dots, \ell$, the following three conditions:*

1. *$A_j \in \Gamma_{j-1}$ whenever $j > 1$;*
2. *$A_j \notin \Gamma_j$;*

3. $A_j \cup \{P_i\} \in \Gamma_j$.

Then, for any weak information theoretic secure multi-secret sharing scheme for access structures $\Gamma_1, \Gamma_2, \ldots, \Gamma_\ell$*, it holds that* $\mathrm{H}(\boldsymbol{V_i}) \geq \sum_{i=1}^{\ell} \mathrm{H}(\boldsymbol{\mathcal{S}_i})$.

6.2.2 Localised threshold multi-secret sharing schemes

We now have the necessary notions to proceed. The requirements of Example 6.1.1 lead us to the following definition:

Definition 6.2.3. *Let* $\mathcal{P} = P_0, P_1, P_2, \ldots$ *be an ordered set of players. A* window of length *n consists of a set of n consecutive players. Denote by W_i the window* $\{P_i, P_{i+1}, \ldots, P_{i+n-1}\}$. *To each window W_i we assign a secret s_i from some finite secret space $\mathcal{S}$. Let V be a finite set of* shares*. A scheme that associates a share $v_i \in V$ to each player P_i is a* localised threshold multi-secret sharing scheme with window length n and threshold t *(denoted (t,n)-LMSS) if it satisfies the following properties:*

- *for any $i = 0, 1, 2, \ldots$ the set of shares associated with the players in a set $P \subseteq \mathcal{P}$ allow the secret s_i to be recovered whenever $|P \cap W_i| \geq t$;*
- *for any $i = 0, 1, 2, \ldots$ if $P \subseteq \mathcal{P}$ is a set of players with the property that $|P \cap W_i| < t$ then the shares associated with the players in P reveal no information about s_i.*

We can see that an LMSS satisfies the requirements of Example 6.1.1.

Example 6.2.1. The use of a $(3,7)$-LMSS to distribute their signing keys would allow the committee in Example 6.1.1 to satisfy their requirements: the first seven committee members are the players $P_0, P_1, \ldots, P_6$, and the remaining committee members are ordered by the year in which they join the committee. Each player P_i is given a share v_i. As illustrated in Figure 6.1, window W_2 contains players P_2, P_3, P_4, P_5, P_6, P_7 and P_8, which correspond to the members who are in the committee during year two. Any three or more of these members can combine their shares to recover the signing key s_2 for year two. For example, the set of members $\{P_3, P_6, P_8\}$ is an authorised set for year two, so shares v_3, v_6 and v_8 can be used to reconstruct s_2. On the other hand, the set $\{P_7, P_8, P_9, P_{10}\}$ only contains two members from within window W_2 and hence the set of shares $\{v_7, v_8, v_9, v_{10}\}$

do not reveal any information about s_2. (They would, however, allow s_3, s_4, s_5, s_6, s_7 or s_8 to be recovered.)

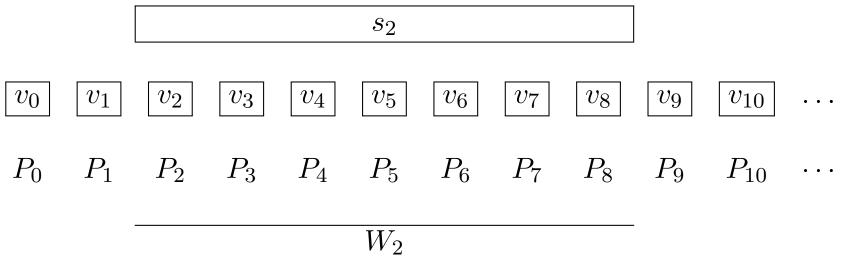
s_2
v_0 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} $\cdots$
P_0 P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 P_{10} $\cdots$
W_2

Figure 6.1: Depiction of window W_2 in a $(3, 7)$-LMSS.

A (t, n)-LMSS can be regarded as a weak information theoretic secure multi-secret sharing scheme, as in Definition 6.2.1, with access structures $\Gamma_0, \Gamma_1, \Gamma_2, \ldots$, where Γ_i is a (t, n)-threshold access structure for the set of players within window W_i. In general, we say a (t, n)-LMSS has weak information theoretic security, or *weak security*, if it corresponds to a weak information theoretic secure multi-secret sharing scheme. Similarly, we say a (t, n)-LMSS is strongly information theoretic secure, or has *strong security*, if it corresponds to a strong information theoretic secure multi-secret sharing scheme.

The key feature of an LMSS is that the set of players is ordered and every minimal authorised set of players is contained within a window of n consecutive players for some n. That is, authorised subsets are sets of a sufficient number of players that are sufficiently 'close' to each other in the underlying ordering.

6.3 Bounds and constructions for LMSSs

6.3.1 Optimal constructions

The most trivial construction for a (t, n)-LMSS with weak security is to deploy a perfect (t, n)-threshold scheme in each window, as observed in [56] (the same approach has been mentioned previously in the literature as a way to construct multi-secret sharing schemes for various other combinations of threshold access structures, *e.g.* [13]). We present this trivial approach as the following construction:

Construction 6.3.1. *Let $\mathcal{P} = P_0, P_1, P_2, \ldots$ be an ordered set of players, and denote by*

W_i the window $\{P_i, P_{i+1}, \ldots, P_{i+n-1}\}$. Let p be a prime with $p > n$. For each window W_i we share a secret $s_i \in \mathbb{Z}_p$ among the n players in W_i using a perfect (t, n)-threshold scheme (such as Shamir's scheme from Construction 2.3.3). This is done independently for each window; a player P_j is thus assigned shares corresponding to each window that contains it, i.e. windows $W_{j-n+1}, W_{j-n+2}, \ldots, W_j$. (The secrets may or may not be independent, but the randomness used in each of the threshold schemes is chosen independently.)

It is straightforward to see that Construction 6.3.1 gives rise to a (t, n)-LMSS in which the total share size for each player is $n \log p$. In the case where $t < n$ this turns out to be optimal, as shown by the following theorem:

Theorem 6.3.2. *Let $\mathcal{P} = P_0, P_1, P_2, \ldots$ be an ordered set of players. Suppose $t < n$ and let Σ be a weakly secure (t, n)-LMSS that associates a share $v_i \in V_i$ to each player P_i. Then for any $j \geq n - 1$ we have that*

$$\mathrm{H}(\boldsymbol{V_j}) \geq \sum_{i=j-n+1}^{j} \mathrm{H}(\boldsymbol{\mathcal{S}_i}).$$

Proof. For ease of notation we prove the result for $j = n - 1$, but the proof applies analogously to any $j \geq n - 1$. Consider the sequence of players $\mathcal{P} = \{P_0, P_1, \ldots, P_{2n-2}\}$. Restricting Σ to these players yields a weakly secure multi-secret sharing scheme on $\mathcal{P}$ where, for $i = 0, \ldots n - 1$, the secret s_i is shared according to the access structure $\Gamma_i = \{A \subset W_i \mid |A| \geq t\}$.

For $r = 1, 2, \ldots, n$, let B_r consist of the first $t - 2 + r$ elements of $\mathcal{P} \setminus \{P_{n-1}\}$. Then the sets B_i satisfy the conditions for Theorem 6.2.2, and so $\mathrm{H}(\boldsymbol{V_{n-1}}) \geq \sum_{i=0}^{n-1} \mathrm{H}(\boldsymbol{\mathcal{S}_i})$, as required. □

If the secrets and shares are all uniformly distributed, we obtain the result that the size of the share given to P_j, for $j \geq n - 1$, in a (t, n)-LMSS with $t < n$ is at least $n \log |s|$, which implies that Construction 6.3.1 is optimal. (We address the case $t = n$ separately, next).

The fact there do not exist constructions for a (t, n)-LMSS with share sizes shorter than those of Construction 6.3.1 means these schemes are not suitable for applications requiring small shares. In Section 6.4 we consider various approaches to relaxing the security definitions for these schemes in a controlled manner so as to allow more efficient constructions.

On the positive side, we observe that if the secrets are generated independently of each other, then the optimal schemes of Construction 6.3.1 are in fact strongly secure.

Interestingly, the restrictions of Theorem 6.3.2 do not apply in the case where $t = n$, as the following construction demonstrates.

Construction 6.3.3. *Let $\mathcal{P} = P_0, P_1, P_2, \dots$ be an ordered set of players, and denote by W_i the window $\{P_i, P_{i+1}, \dots, P_{i+n-1}\}$. Suppose that for $i \geq 0$, a secret s_i is generated uniformly at random from the set $\mathbb{Z}_2^\lambda$, and that this is done independently for each i. For $i = 0, 1, \dots, n-2$ we assign a share v_i to player P_i by generating v_i uniformly at random from $\mathbb{Z}_2^\lambda$. For $i \geq n-1$ we set $v_i = s_{i-n+1} \oplus v_{i-n+1} \oplus v_{i-n+2} \oplus \cdots \oplus v_{i-1}$.*

To intuitively understand Construction 6.3.3, the last player in a window is given the XOR of all previous shares in the window XORed with the window key. Specifically, take window W_{i-n+1}. The last player in this window is P_i, who is given share v_i, which is the XOR of the window secret s_{i-n+1} with all other shares in the window $v_{i-n+1}, v_{i-n+2}, \dots, v_{i-1}$. Then, to recover a window secret, all players in a window need only XOR their shares.

Theorem 6.3.4. *Construction 6.3.3 results in an (n, n)-LMSS that has weak security and optimal share size, but which does not possess strong security.*

Proof. In an (n, n)-LMSS, the access structure Γ_j has a single minimal authorised subset A, which is the set consisting of all n players in the window W_j:

$$A = \{P_j, P_{j+1}, ..., P_{j+n-1}\}.$$

Note that the secret s_j can be recovered from the shares belonging to players in A by XORing all their shares: $s_j = v_j \oplus v_{j+1} \oplus \cdots \oplus v_{i+n-2} \oplus v_{j+n-1}$.

For the access structure Γ_j, the largest unauthorised subsets have the form

$$B_{\hat{i}} = \{P_0, P_1, \dots, P_{i-1}, P_{i+1}, \dots, P_{j+n-1}, \dots\},$$

for some i with $j \leq i \leq j + n - 1$. That is, $B_{\hat{i}}$ is the set of all players excluding one player P_i who is contained in the window W_j. The set $B_{\hat{i}}$ is unauthorised because not all n players contained in the window W_i are present in $B_{\hat{i}}$.

Suppose an adversary who wishes to determine s_j possesses all the shares in $B_{\hat{i}}$. By the definition of Construction 6.3.3, the share v_{j+n-1} is constructed as

$$v_{j+n-1} = s_j \oplus v_j \oplus v_{j+1} \oplus \cdots \oplus v_{i-1} \oplus v_i \oplus v_{i+1} \oplus \cdots \oplus v_{j+n-2},$$

hence

$$s_j = v_j \oplus v_{j+1} \oplus \cdots \oplus v_{i-1} \oplus v_i \oplus v_{i+1} \oplus \cdots \oplus v_{j+n-2} \oplus v_{j+n-1}. \tag{6.3.1}$$

The adversary knows all terms on the right-hand side of (6.3.1) apart from the share v_i (as v_i belongs to P_i, who is not contained in $B_{\hat{i}}$). But the share v_i can be expressed as

$$v_i = s_{i-n+1} \oplus v_{i-n+1} \oplus v_{i-n+2} \oplus \cdots \oplus v_{i-1}. \tag{6.3.2}$$

With this is mind, we show that Construction 6.3.3 does not have strong security, but does have weak security. We also show the construction is optimal.

Strong security. Construction 6.3.3 does not have strong security. In the strong security setting, in addition to the shares possessed by the players in $B_{\hat{i}}$, we assume the adversary also knows all secrets other than s_j. So the adversary knows all secrets but s_j and all shares but v_i.

Consider (6.3.2). The adversary knows all shares $v_{i-n+1}, v_{i-n+2}, \ldots, v_{i-1}$, and all secrets except s_j. If $j \neq i-n+1$, the adversary knows s_{i-n+1} and can immediately use (6.3.2) to calculate v_i. They then know all shares in the window W_j and can recover s_j via (6.3.1).

If $j = i-n+1$, the secret s_{i-n+1} in (6.3.2) is unknown (as s_j is unknown and $j = i-n+1$) and the adversary cannot proceed as before. Note this is the case where the player excluded from $B_{\hat{i}}$ is the last player in the window W_j. Here, the adversary does not know the share $v_i = v_{j+n-1}$ or the secret $s_j = s_{i-n+1}$. However, we observe that:

$$v_{j+n} = s_{j+1} \oplus v_{j+1} \oplus v_{j+1} \oplus \ldots \oplus v_{j+n-2} \oplus v_{j+n-1},$$

hence

$$v_{j+n-1} = s_{j+1} \oplus v_{j+1} \oplus v_{j+1} \oplus \ldots \oplus v_{j+n-2} \oplus v_{j+n}. \tag{6.3.3}$$

The adversary possesses all terms on the right-hand side of (6.3.3) and so can calculate $v_{j+n-1} = v_i$. Once v_i has been calculated, the adversary once again knows all the shares in the window W_j and can use (6.3.1) to calculate s_j, as before.

Weak security. In the weak security setting, the adversary does not know any secrets others than those it is able to compute using the shares in its possession. So, assume the adversary has only the shares belonging to players in $B_{\hat{i}}$ and wants to calculate the secret s_j; the adversary has all information necessary apart from the share v_i. The same is true for the secrets corresponding to every window containing the share v_i. Consider the following system of equations:

$$\begin{aligned} s_{i-n+1} &= v_{i-n+1} \oplus v_{i-n+2} \oplus \cdots \oplus v_{i-1} \oplus v_i, \\ s_{i-n+2} &= v_{i-n+2} \oplus v_{i-n+3} \oplus \cdots \oplus v_i \oplus v_{i+1}, \\ &\vdots \\ s_i &= v_i \oplus v_{i+1} \oplus \ldots \oplus v_{i+n-2} \oplus v_{i+n-1}. \end{aligned}$$

This is a system of n linear equations. The adversary knows all the values except for the $n+1$ values in the set $S = \{v_i, s_{i-n+1}, s_{i-n+2}, \ldots, s_i\}$ (the n secrets and the share v_i belonging to player $P_i \notin B_{\hat{i}}$). For every possible choice of $s_j \in \mathbb{Z}_2^\lambda$ there is a choice of the values in $S \setminus \{s_j\}$ consistent with the set of shares corresponding to players in $B_{\hat{i}}$. Hence these shares reveal no new information about the secret s_j, meaning this construction has weak security.

Optimality. To show this scheme is optimal with respect to share size, we note that restricting this construction to any window gives a perfect (n, n)-threshold scheme for that window in which the size of each share is the same as the size of the corresponding secret. Since for a perfect (n, n)-threshold scheme the size of the shares must be at least as large as the size of the secret, this is optimal. $\square$

The scheme of Construction 6.3.3 thus gives an example of a (t, n)-LMSS that is weakly secure, but not strongly secure. We note that it has independently generated secrets, thus demonstrating that having independently generated secrets is a necessary but not sufficient condition to guarantee a scheme with weak security will also have strong security.

6.3.2 Time dependent schemes

In many applications, such as our motivating Example 6.1.1, the secrets $s_0, s_1, \ldots$ of an LMSS have a natural interpretation as a sequence of secrets that change over time. In such a setting, it makes sense to consider a security model that is intermediate between the strong and weak models. Hence, we introduce the following notion of security:

Definition 6.3.5. *A weakly secure (t, n)-LMSS is said to have* perfect backward secrecy *if the shares possessed by any set $A \notin \Gamma_j$ together with the set of the first j secrets $T_j = \{s_0, s_1, \ldots, s_{j-1}\}$ reveals no information about the secret s_j other than that already revealed by T_j. In entropy terms,* $\mathrm{H}(\boldsymbol{\mathcal{S}_j} \mid \mathbf{A}, \mathbf{T_j}) = \mathrm{H}(\boldsymbol{\mathcal{S}_j} \mid \mathbf{T_j})$.

Perfect backward secrecy ensures that the exposure of past secrets does not affect the security of future secrets. Note that, by definition, a (t, n)-LMSS with perfect backward secrecy is also a weakly secure (t, n)-LMSS. Furthermore, a strongly secure (t, n)-LMSS necessarily has perfect backward secrecy. Thus, perfect backward secrecy can be seen as an intermediate requirement between weak and strong security. A scheme with strong security in fact possesses both perfect backward secrecy and perfect forward secrecy, where compromise of future secrets does not affect the security of past secrets. This is interesting from the point of view of motivating the strong security model, given that the scenario in Example 6.1.1 seemed *a priori* only to require weak security.

It is interesting to consider whether the scheme of Construction 6.3.3 has the property of perfect backward secrecy. The following result shows that, in fact, it only has quite limited backward secrecy:

Theorem 6.3.6. *Construction 6.3.3 does not have perfect backward secrecy: an adversary possessing the secrets $s_0, s_1 \ldots s_{j-1}$ who also has shares of a maximal set of players $B_{\hat{i}} \notin \Gamma_j$ can determine s_j except in the case where $i = j + n - 1$, where no information about s_j is revealed.*

Proof. Consider the proof of Theorem 6.3.4. In showing that an adversary who has access to all secrets other than s_j can recover s_j in the case where $i \neq j + n - 1$, we in fact only made use of secrets s_ℓ with $\ell < j$. Hence the same argument demonstrates that this construction does not give a backward secure scheme. Interestingly, when $i = j + n - 1$ the adversary in fact learns nothing about s_j in the backward secure setting. To see this, we consider the following set of equations:

$$\begin{aligned} s_{i-n+1} &= v_{i-n+1} \oplus v_{i-n+2} \oplus \cdots \oplus v_{i-1} \oplus v_i, \\ s_{i-n+2} &= v_{i-n+2} \oplus v_{i-n+3} \oplus \cdots \oplus v_i \oplus v_{i+1}, \\ &\vdots \\ s_i &= v_i \oplus v_{i+1} \oplus \cdots \oplus v_{i+n-2} \oplus v_{i+n-1}. \end{aligned}$$

This is a set of n linear equations with $n+1$ unknowns, namely $s_{i-n+1}, s_{i-n+2}, \ldots, s_i, v_i$. For every possible secret s_{i-n+1} there exists a choice for the remaining elements of this set that is consistent with the view of the adversary. Therefore, the construction does ensure that an adversary lacking share v_{i-n+1} learns no information about s_{i-n+1}. $\square$

6.4 Relaxing security requirements to construct more efficient schemes

The bounds on share sizes implied by Theorem 6.3.2 mean that in order to construct a more efficient (t, n)-LMSS it is necessary to relax the security definition. There are various ways in which this could be done. The *SWISS schemes* proposed by Juels *et al.* are one example [56]; we discuss this scheme and limitations of it in Section 6.5.2.2. In this section, we systematically consider a range of techniques that can be applied while still working in the setting of information theoretic security.

Recall the essential aims of a (t, n)-LMSS are to ensure that any t suitably close users are able to recover a secret, and that each secret should only be accessible to players within a bounded window. The techniques considered here allow us to maintain these goals, while relaxing the strict requirements of Definition 6.2.3 in ways that gives a well understood trade-off between the security compromises and the resulting efficiency gains.

6.4.1 Shifting to a non-perfect model of secret sharing

Section 2.3.1 indicated that the shares of a perfect (t, n)-threshold scheme have to be at least as large as the secret. But in Section 2.3.2 we showed that share sizes could be reduced by the use of a $(t_0, t_1; n)$-ramp scheme, with the increased efficiency being traded against the relaxation of the security in that sets of players of sizes between t_0 and t_1 can now gain partial information about the secret.

The exact same technique can be applied in the context of a (t, n)-LMSS, by replacing the use of Shamir's threshold scheme in Construction 6.3.1 with a $(t_0, t_1; n)$-ramp scheme, such as that in Construction 2.3.13. That is, we maintain the same correctness requirement for the (t, n)-LMSS but relax the security requirement. Reducing the sizes of sets of players that are excluded from learning any information about the secret from $t_1 - 1$ to t_0 in this manner allows us to decrease the size of the shares by a factor of $t_1 - t_0$.

6.4.2 Changing the access structures

One consequence of Definition 6.2.3 is that the functionality (in terms of which keys a given player can contribute to recovering) is inextricably tied directly to the security (in terms of the sizes of the windows of players that can contribute to recovering a particular key). Specifically, if we wish to consider windows of length n, then, according to this definition, each player is necessarily a participant in n distinct windows. As this connects directly to the storage overheads for each player, this restriction limits the scope for improving the efficiency of such schemes.

It would be desirable to have greater flexibility in varying the parameters of a scheme. One way to achieve this is to decouple the access structure (*i.e.*, the definition of which sets of players are authorised to access keys) from the pattern of key distribution (*i.e.*, which keys are able to be accessed by particular authorised sets). In this section, we set up a framework for analysing this more general setting and explore the resulting consequences in terms of security and practicality.

Consider the access structure and the key windows as two separate notions.

- *The access structure in isolation*: As previously, we consider an ordered set of players denoted by $\mathcal{P} = P_0, P_1, P_2, \ldots$. We define an access structure Γ on $\mathcal{P}$ by specifying that the authorised sets in Γ are all those subsets of $\mathcal{P}$ that contain a subset of the form $S = \{P_{i_1}, P_{i_2}, \ldots, P_{i_t}\}$ where $i_1 < i_2 < \cdots < i_t$ and $i_t - i_1 \leq n-1$. That is, any subset of $\mathcal{P}$ that contains t or more players from within a window of n consecutive players is authorised. We think of the authorised sets in Γ as being those that have the right to reconstruct at least one key.

- *Key windows*: The defining property of a localised threshold multi-secret sharing scheme is that we want any given key to be accessible by sufficiently large sets of players that are suitably close. For a given key s we suppose there is a specific *key window* W^s consisting of the players $P_i, P_{i+1}, \ldots P_{i+\ell-1}$ that we think of as having the potential to be involved in recovering s. Note that, unlike in Section 6.3, we no longer require $\ell = n$ but we can also allow $\ell \geq n$.

We now consider the restriction of the access structure Γ to the window W^s. This gives us an access structure Γ^s on the players in W^s whose authorised sets are all those of the form $A \in \Gamma$ with $A \subseteq W^s$. This is illustrated in Figure 6.2 for the case $\ell = 4$ and $n = t = 2$.

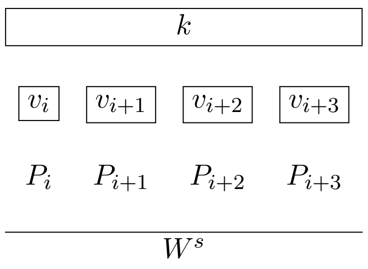

Figure 6.2: When $\ell = 4$ and $n = t = 2$, any pair of adjacent players are authorised to recover the secret.

For $\ell > n$ the access structure Γ^s is no longer a threshold access structure. It is possible to construct a secret sharing scheme assigning shares to the players in W^s that enable authorised sets in Γ^s to recover s while preventing the shares belonging to any set $A \notin \Gamma^s$ from gaining any information about s [53]. However, this may require the share sizes to be larger than the size of the secret. For example, in the case where $\ell = 4$ and $n = t = 2$ (as depicted in Figure 6.2), Capocelli *et al.* show that it is necessary for the largest share to be at least 50% larger than the size of the secret [22].

One approach to avoiding this issue (as suggested in [56]) is simply to use a (t, ℓ)-threshold scheme to share s among the players in W^s, since this allows shares that are no larger than the size of the secret (and permits a straightforward trade-off between security and share size through the use of a suitable ramp scheme if desired). However, in the case where ℓ is significantly larger than n (the schemes proposed in [56] have $\ell \geq 2n$) this results in many subsets of W^s that are not authorised having shares that allow them to recover the secret, namely any set of t shares $\{v_{i_1}, v_{i-2}, \dots, v_{i_t}\}$ with $i_1 < i_2 < \cdots < i_t$ and $i_t - i_1 > n$. This can be substantially mitigated at no extra cost by the use of the following construction:

Construction 6.4.1. *Generate shares of the secret s for the players in W^s as follows:*

- *Share s using a perfect (t, n)-threshold scheme and assign the resulting shares $v_1, v_2, \dots, v_n$ to the first n players in the window.*
- *Assign $v_1, v_2, \dots, v_n$ in turn to the next n players in turn and so on, cycling through the shares as necessary throughout the rest of the window.*

This construction ensures the shares possessed by any n consecutive players are precisely those of a (t, n)-threshold scheme, and so the shares of any subset of t or more of those n consecutive players can recover the secret, as required by Γ^s. There is potential for a small saving in the size of the shares relative to using a (t, ℓ)-threshold scheme since it is possible to use a field of size n rather than ℓ. However the main advantage of this construction is in reducing the number of sets $A \notin \Gamma^s$ that can recover s. For a (t, ℓ)-threshold scheme, there are $\binom{\ell}{t}$ subsets of size t that can recover the secret, whereas Γ^k contains only $(\ell - n)\binom{n-1}{t-1} + \binom{n}{t}$ authorised subsets of size t (as there are $\binom{n}{t}$ ways of choosing such a subset from among the last n players in the window; for subsets not wholly contained within the last n players there are $\ell - n$ possible choices for the first player in the subset and $\binom{n-1}{t-1}$ ways to choose the rest of the subset from the $n - 1$ subsequent players).

For example, if $\ell = 4$ and $n = t = 2$, and if a $(t = 2, \ell = 4)$-threshold scheme is used, there are $\binom{4}{2} = 6$ pairs of shares that can recover the secret, whereas Γ^s only has $2\binom{1}{1} + \binom{2}{2} = 3$ authorised pairs. This implies that half of the pairs of players enabled by the threshold scheme to access the secret are not in Γ^s. For the scheme given in Construction 6.4.1, on the other hand, if $\ell = \alpha n$ there are $\alpha^t\binom{n}{t}$ subsets of size t that can recover the secret.

For $\ell = 4$ and $n = t = 2$ this gives $2^2\binom{2}{2} = 4$ pairs that can recover the secret, so only a quarter of these are not in Γ^s.

We note that Construction 6.4.1 can also be instantiated with a ramp scheme in place of the threshold scheme, if desired.

6.4.3 Staggering key windows

If we were to fix $\ell = n$ and require a new key window of length n starting with player P_i for every $i = 1, 2, \ldots$ then we would recover Definition 6.2.3. However, by allowing distinct values of ℓ and n, and allowing more flexibility in the distribution of the key windows, we can obtain a family of schemes that have the potential for much greater flexibility in tailoring their properties to suit our application requirements.

One way to do this is to introduce a parameter d that describes the offset between consecutive key windows, so secret s_0 is associated with the window $W^{s_0} = \{P_0, P_1, P_2, \ldots, P_{\ell-1}\}$, whilst secret s_1 is associated with the window $W^{s_1} = \{P_d, P_{d+1}, \ldots, P_{d+\ell-1}\}$, and so on. In general, the secret s_i is associated with the window $W^{s_i} = \{P_{id}, P_{id+1}, \ldots, P_{id+\ell-1}\}$.

The following lemmas describe basic properties of the scheme that arises from staggering the key windows in this fashion.

Lemma 6.4.2. *An authorised set $A = \{P_{i_1}, P_{i_2}, \ldots, P_{i_m}\}$ with $i_m - i_1 = c$ for some $c \leq n - 1$ can reconstruct either $\lfloor \frac{\ell - c}{d} \rfloor$ or $\lceil \frac{\ell - c}{d} \rceil$ secrets.*

Proof. The shares corresponding to the players in set A allow them to recover the secret for any window W with $A \subseteq W$. A window of length ℓ contains A if it starts with a player between $P_{i_{m-\ell}}$ and P_{i_1}. This is a range of $\ell - c$ possible starting points. If the windows have offset d then for any value of i_m at least $\lfloor \frac{\ell - c}{d} \rfloor$ windows, and up to $\lceil \frac{\ell - c}{d} \rceil$ windows will start in this range. $\square$

The share storage requirements for any individual player are given by the following lemma:

Lemma 6.4.3. *Any single player is associated with either $\lfloor \frac{\ell}{d} \rfloor$ or $\lceil \frac{\ell}{d} \rceil$ shares.*

The proof is a direct analogue of that of Lemma 6.4.2.

Example 6.4.1. Suppose we take $\ell = 2n$ and $d = n$. Then every player is associated with shares from two distinct key windows, and any authorised set is able to recover either one or two distinct window keys.

6.4.4 Combining techniques

Combining all the techniques discussed in this section gives us the following construction:

Definition 6.4.4. *Let $\mathcal{P} = P_0, P_1, P_2, \dots$ be an ordered set of players, and denote by W_i the window $\{P_i, P_{i+1}, \dots, P_{i+\ell-1}\}$. Let p be a prime with $p > n$. For each window W_i for $i = 0, d, 2d, 3d, \dots$ we share a secret $s_i \in \mathbb{Z}_p$ among the ℓ players in W_i using Construction 6.4.1 implemented with a $(t_0, t_1; n)$-ramp scheme. The resulting construction is called a flexible localised multi-secret sharing scheme, denoted $(t_0, t_1; n, \ell, d)$-*fLMSS.

The following properties of this construction follow directly from the earlier results in this section.

Theorem 6.4.5. *A $(t_0, t_1; n, \ell, d)$-*fLMSS *has the following properties:*

- *If the secrets are all independent and identically distributed according to the uniform random variable $\boldsymbol{\mathcal{S}}$ then the size of each share is $\left\lceil \frac{\ell}{d} \right\rceil \frac{\log_2 |s|}{t_1 - t_0}$ bits.*
- *Any set of players $\{P_{i_1}, P_{i_2}, \dots, P_{i_m}\} \subseteq W_i$ for $i = 0, d, 2d, \dots$ with $m \geq t_1$ and $i_m - i_1 \leq n$ is able to recover the secret s_i.*

Example 6.4.2. Consider secret windows of length $\ell = 150$, with an offset of $d = 40$ between consecutive window keys. Let $n = 100$, $t_1 = 50$ and $t_0 = 30$, so any 50 players $P_{i_1}, P_{i_2}, \dots, P_{i_{50}}$ with $i_{50} - i_1 \leq 99$ are able to fully construct a common secret, whereas sets of 30 or fewer players learn no information about the secret. Using Lemma 6.4.2, an authorised set $A = \{P_{i_1}, P_{i_2}, \dots, P_{i_{50}}\}$ with $i_{50} - i_1 = c$ for some $c \leq 99$ is able to reconstruct $\left\lfloor \frac{150-c}{40} \right\rfloor$ or $\left\lceil \frac{150-c}{40} \right\rceil$ secrets. For example, when $c = 50$ we have

$$\frac{\ell - c}{d} = \frac{150 - 50}{40} = 2.5,$$

so each authorised subset is always able to construct at least two, and potentially up to three, window secrets. To illustrate this, consider the authorised set of players $A_0 =$

$\{P_{120}, P_{121}, \ldots, P_{169}\}$ depicted in Figure 6.3. The shares in A_0 are capable of reconstructing the secrets for windows W_{40}, W_{80} and W_{120}. This is the maximum possible number of secrets a set of 50 players can recover. Consider a different authorised set $A_1 = \{P_{119}, P_{120}, \ldots, P_{168}\}$. The players in A_1 can reconstruct the secrets for windows W_{40} and W_{80}, but no longer have sufficient shares to compute the secret for window W_{120}.

Lemma 6.4.3 implies that any single player is associated with either $\lfloor \frac{150}{40} \rfloor$ or $\lceil \frac{150}{40} \rceil$ shares. In this example,

$$\frac{\ell}{d} = \frac{150}{40} = 3.75,$$

so each individual player is associated with either three or four shares. For example, consider the player P_{210}. This player must hold shares for the secrets of windows W_{80}, W_{120}, W_{160} and W_{200} and hence is an example of a player who holds four shares. On the other hand, player P_{235} must hold a share for the secret for windows W_{120}, W_{160} and W_{200} and thus holds three shares. Both players are illustrated in Figure 6.3.

Theorem 6.4.5 states that the total size of the shares for each player in a $(t_0, t_1; n, \ell, d)$-fLMSS is

$$\left\lceil \frac{\ell}{d} \right\rceil \frac{\log_2 |s|}{t_1 - t_0}$$

bits. Here, we have a $(30, 50; 100, 150, 40)$-fLMSS, so if we are wishing to have a uniformly generated 32 bit secret (for example), we obtain

$$\left\lceil \frac{\ell}{d} \right\rceil \frac{\log_2(s)}{t_1 - t_0} = \left\lceil \frac{150}{40} \right\rceil \frac{32}{50 - 30} = 4 \times \frac{32}{20} < 7,$$

so we would require each player to store at most 7 bits. For a secret of 64 bits, we would require at most 13 bits of storage. In comparison, were we to use Construction 6.3.1, each player would be required to store 100 shares each of size 32 bits, leading to a total storage of 3200 bits, or 6400 bits in the case of 64 bit secrets.

6.5 Application to key distribution in RFID enabled supply chains

In this section, we progress from our initial motivation, given in our toy Example 6.1.1, and define the main motivation for LMSSs, that of RFID enabled supply chains. We explore

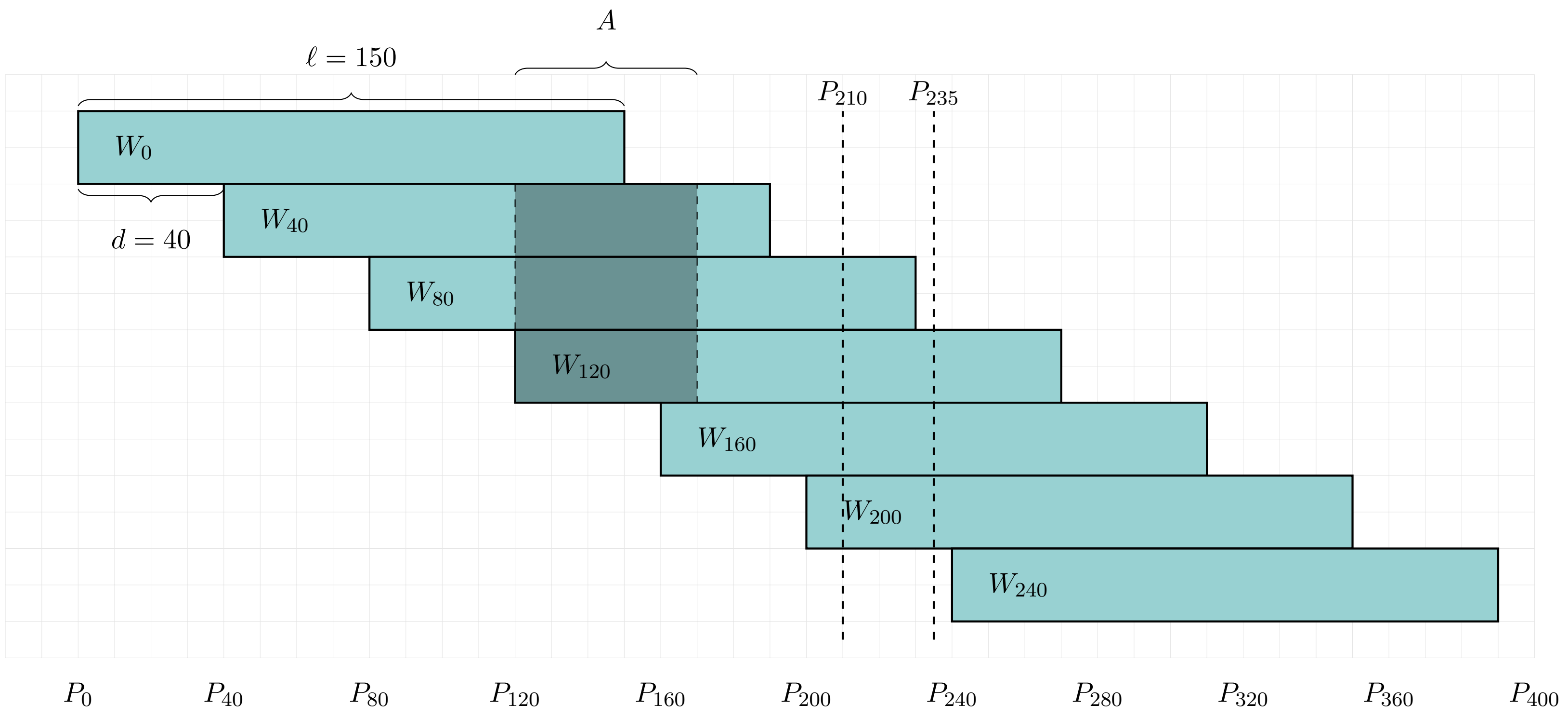

Figure 6.3: Illustration of properties of the $(30, 50; 100, 150, 40)$-fLMSS of Example 6.4.2.

the problem then highlight a number of ways in which threshold schemes have previously been used in RFID enabled supply chains. We introduce the SWISS schemes in [56], which inspired the definition of LMSS, and highlight the similarities and differences between the SWISS schemes and our fLMSS. Finally, we consider the application of fLMSS to RFID enabled supply chains.

6.5.1 RFID enabled supply chains

Radio frequency identification (RFID) is a technology that uses radio signals to identify objects [1]. An RFID system consists of a tag, a reader and a back-end server [1]. RFID devices are used in transportation, logistics, manufacturing and processing and typical applications include inventory control, animal tagging, postal tracking, airline baggage management, access control, and manufacturing processes [88].

RFID tags enable the identification, tracking and verification of products in a supply chain both automatically and in real time [1], and have the potential to store information such as batch numbers and date of manufacture, hence their use is becoming more prevalent in manufacturing. Ultimately, it is expected that items will be tagged on an individual level, enabling them to be tracked from the factory to the point of sale.

In supply chains, the predominant RFID standard is known as the Electronic Product Code (EPC) [34] and, in particular, Class-1 Gen-2 EPC, henceforth referred to as Gen2. Gen2 tags can be regarded as new generation bar-codes that emit some data, called an EPC code, that contains four elements [56]: a *header*, which denotes the EPC version number; a *domain manager* that details the manufacturer; an *object class* that specifies the item type; and a *serial number*, which is a universal identifier for the item. After a tag is read by a reader, the unique serial number enables the tag to be linked to a database containing other vital information related to the product. Storing the EPC code rather than all the information relevant to the product requires less memory on the tag, which is ideal as tags have a limited memory of up to 2KB of data [56].

However, the use of these tags in supply chains creates new security and privacy challenges. For instance, consider a consumer who is in possession of a tagged product. They run the risk that a passer-by could scan the tag and thereby determine they are carrying the product. There are many items, such as medications, for which this is potentially undesirable. Additionally, if the same EPC tag was read at multiple locations, the passer-by would be able to track the consumer's movements. Hence there are privacy concerns for the consumer with the use of RFID tags.

Another challenge is that of key distribution. In particular, there are two features in the Gen2 standard that require secret keying material:

- Each tag's memory can be either permanently or temporarily *locked*, which prevents alteration of the tag's memory. With the use of a write-access key, a user can unlock the tag and write to its memory. We note that even if a tag is locked, the memory can still be read, just not written to.
- The *kill command* is a function that forces the tag to permanently disable itself.

Generally, the tag's memory may be written and locked at the point of manufacture, then read at different stages throughout the supply chain. Finally, at the point of sale, the kill command may be executed in order to address the privacy concerns of the consumer. However, there may not exist any prior secure channels between the manufacturer and merchant, who legitimately requires access to the tags' contents (in order to execute the kill function, for example). From leaving the factory to arriving on the shelf, the RFID tag may pass between several different organisations and, at the point in which the RFID's memory is written, the factory may not know where the RFID tag is being sent.

A solution that prevents a casual adversary from reading the contents of a tag, whilst allowing a legitimate user to read and potentially lock or kill the tag, is required. One popular approach is to exploit differences in the way in which a legitimate user is able to access the data on a tag, as opposed to the potentially more restricted access available to a casual adversary.

For example, in [65], the authors assume an adversary only has momentary access to an RFID tag, whilst a legitimate user has access for an extended period of time. Correspondingly, they suggest distributing the data on a tag via a perfect (n, n)-threshold scheme, for some value n. All the shares are then stored on the tag and, each time the tag is queried, it will emit a different share. Thus, a casual adversary with access for a limited time period will be unable to read all n shares and so will learn no information about the data stored on the tag, whereas a legitimate user will be able to query the tag a sufficient number of times in order to learn all the shares and thus read the tag. However, this solution requires each tag to store n times the size of the original data which, given the limited memory available on these tags, may be too much.

We now discuss two solutions, both proposed in [56] by Juels, Pappu and Parno, that leverage the different behaviours and abilities of an adversary and a legitimate user.

6.5.2 Existing solutions

In [56], Juels, *et al.* suggest two ways in which threshold schemes can be utilised in RFID enabled supply chains. They call their techniques *secret sharing across space* and *secret sharing across time*. Both techniques address slightly different scenarios in the RFID enabled supply chain, with secret sharing across time addressing the same problem as the LMSS scheme. We begin, however, by introducing the secret sharing across space solution, and discuss its limitations.

6.5.2.1 Secret sharing across space

In [56], the authors consider the following scenario:

Example 6.5.1. Suppose a manufacturer places tags on items that are shipped in cases (where one case consists of many items) to a merchant who then sells them individually to customers. The merchant (who has a legitimate need to read the tags) has access to multiple tags. On the other hand, after the items are sold, an adversary (who should be prevented from recovering the tags' identities) is expected to only have access to a small number of tags.

In this scenario, Juels *et al.* suggest using one case-wide key for all items in a case, which can be combined with each tag's serial number to compute a unique key for each tag; this unique key may be a write-access key, or a key that allows execution of, for example, the kill command. In order to secure the case-wide key, they suggest distributing it across all the tags in the case using a threshold scheme, with each tag receiving a single share. This ensures the merchant who possesses the entire case can gather enough shares to recover the case-wide key (and therefore the unique key for each tag), but once the items have been sold they are sufficiently dispersed that the case-wide key, and therefore the unique key for a given tag, can no longer be recovered.

In order to keep the share sizes of the case-wide key small, Juels *et al.* propose using a variation of Krawczyk's HK0 scheme, defined in Figure 2.3, which they call their *tiny secret sharing scheme (TSS).* Like Krawczyk's scheme, their TSS scheme provides computational (rather than information theoretic) security [60] but is only secure in the RO model. Recall that, in Krawczyk's HK0 scheme, a large secret is encrypted with a (relatively) short key and the resulting ciphertext is shared using an IDA (with no security requirements), while the key is shared using a perfect (t, n)-threshold scheme. For the TSS, Juels *et al.* encrypt the large secret with a hash of the encryption key, then share both the key and the ciphertext with an IDA based on an ECC, as discussed in Section 2.4.2. They claim this enables them to *'make the size of our shares independent of the secret'.* In fact, this is not correct, nor is their assertion that Krawczyk's HK0 scheme has *'shares with lengths independent of the secret's size'.* Rather, as Krawczyk states in his abstract [60], his scheme is *'an m-threshold scheme... in which shares corresponding to a secret s are of size $|s|/m$ plus a short piece of information whose length does not depend on the secret size but just on the security parameter'.* (The bound of $|s|/m$ is clearly optimal if the secret is to be recovered from m shares.)

In some cases, the authors say, only a key need be distributed, and not a large secret. In this case, the key can be directly distributed via an IDA based on an ECC. For example, it is known (and has already be stated in Section 2.4.2), that a length n code with minimum distance d and dual distance d^* gives a $(d^* - 2, n - d + 1; n - 1)$-ramp scheme. Thus, if the dual distance of the code is small, t_0 is also small and the resulting scheme only guarantees protection of the key against small coalitions of players. In Krawczyk's HK0 scheme the computational security is ensured by the fact a perfect threshold scheme is used to share the key. Replacing the threshold scheme with an information dispersal algorithm based on an ECC, as is done in the TSS construction, leads to a similar reduction in security as would be caused by simply using a ramp scheme to share the message directly, does not reduce the storage relative to this more straightforward approach, and offers only computational rather than information theoretic security guarantees. As such, this ECC approach does not appear to offer any clear advantages in this context.

Thus, we conclude by commenting that the use of a threshold scheme to distribute a case-wide key across all the tags in a case of items, with each tag receiving a single share, is a novel and interesting idea, but that the scenario given in Example 6.5.1 may be better served by the straightforward use of either Krawczyk's HK0 scheme, or a linear $(t_0, t_1; n)$-ramp scheme, rather than the proposed TSS, dependent on whether a large secret or only a key is to be distributed.

6.5.2.2 Secret sharing across time

An adversary that gains access to an entire case of items in Example 6.5.1 will be able to learn the case-wide key, and thus derive the unique key to each tag. This would then enable the adversary to use the unique keys to write to the tags' memories, or execute the kill command, for example. Neither of which are ideal. For this reason, Juels *et al.* consider a further scenario, in which a key is distributed across not one case, but across multiple cases. As before, the distributed key is used for multiple tags and is combined with each tag's serial number in order to derive a unique key for each tag. For ease, we refer to the key begin distributed simply as the key. Given that cases arrive at staggered times in a supply chain, Juels *et al.* refer to this method as secret sharing across time, and give the following example.

Example 6.5.2. Alice is shipping RFID tagged items to Bob and wishes to communicate the key for the tags to Bob as securely as possible. Alice employs trucks that hold up to ten cases and is concerned an adversary may gain access to at least one case, if not the whole truck. She wishes to prevent such an adversary from gaining access to the relevant keys.

Juels *et al.* suggest Alice does as follows. She selects a sequence of eleven cases P_j, $P_{j+1}, \ldots, P_{j+10}$, called a window, designated for delivery to Bob and creates a key s from which, along with the knowledge of each tag's unique key, it is possible to derive the unique key for any tag within the window of cases. She distributes s into eleven shares $v_1, v_2, \ldots, v_{11}$ via a perfect $(11, 11)$-threshold scheme, and writes share v_i to case P_{j+i-1}. With this solution, an adversary that gains access to the contents of a small collection of cases, or even an entire truckload, is unable to reconstruct s or obtain the unique keys for any RFID tag. On the other hand, Bob can reconstruct s once he receives the full sequence of eleven constituent cases.

Example 6.5.2 is a very specific example given by Juels *et al.* We present a generalised scenario that builds on this motivating example.

Example 6.5.3. Suppose a manufacturer attaches RFID tags to items as they come off the production line. The items are then packed and shipped to meet orders coming in from wholesale customers. The manufacturer wishes to distribute keys across items in an order using a (t, n)-threshold scheme, as in Example 6.5.1. However, the customers may order differing numbers of items, and at the time when the data is being placed on the tags the manufacturer does not yet know what these orders are going to be (either in terms of their sizes or to which customer they will be shipped). The shares on the tags in a single order must enable the wholesaler to recover a suitable key, yet adversaries who obtain fewer than t shares from a given order should learn no information about the key. (In particular, this means that the tags from a certain wholesaler's orders should not allow that wholesaler to learn the key corresponding to another wholesale customer's order.)

As a solution to this problem, Juels *et al.* propose using multiple overlapping windows and attaching to each window a key, which can be recovered if a sufficient number of cases within a window are received and their shares learnt. This is the basis of their *sliding window information secret sharing (SWISS) scheme.* Their basic SWISS scheme, which is a $(t, 2n)$-SWISS scheme, uses windows of length $2n$, that overlap by an offset of $d = n$, and distribute each window key via a perfect $(t, 2n)$-threshold scheme. In their basic SWISS scheme, each case is included in two windows and is thus given two shares. Any t cases within n of each other in the underlying order will be able to recover either two or three window keys.

After defining their basic SWISS scheme, they observe that it can be generalised to a (t, x)-SWISS scheme by introducing some parameter Ψ. The (t, x)-SWISS scheme has windows of length $(\Psi + 1)x/\Psi$ which overlap by Ψ cases, and distributes each window key via a $(t, (\Psi + 1)x/\Psi)$-threshold scheme. In this generalisation, each case in included in $\Psi + 1$ windows and is thus given $\Psi + 1$ shares. Any t cases within $(\Psi + 1)x/\Psi$ of each other in the underlying order will be able to recover either $\Psi + 1$ or $\Psi + 2$ window keys.

Finally, Juels *et al.* mention that one could use their TSS scheme in place of a perfect threshold scheme in order to further reduce share sizes.

6.5.2.3 Applying the fLMSS

We observe that the requirements of Example 6.5.3 are in fact essentially the same as those of Example 6.1.1; hence, a $(t_0, t_1; n, \ell, d)$-fLMSS is an appropriate solution for the manufacturer's needs in this situation. We relate our previous Example 6.4.2 to RFID enabled supply chains:

Example 6.5.4. In Example 6.4.2, we saw that the use of a $(30, 50; 100, 150, 40)$-fLMSS allowed a 64 bit secret to be distributed while only requiring each player to store a 13 bit share. This is well within the capacity of a Gen2 tag, and would be suitable, for example, in a situation where merchants order shipments of at least 100 items at a time.

6.5.2.4 Relating the SWISS scheme and LMSS

In defining their SWISS construction, Juels *et al.* did not explore any constructions that provide perfect security, such as is achieved in Construction 6.3.1. This motivated our LMSS definition and the exploration of constructions and bounds.

We then recognised the overlapping of windows as one technique to make efficiency gains, and identified a number of other techniques that could be used. In particular, we recognised the fact that Ψ and d completely determines both n and ℓ (or alternatively, the choice of Ψ and n completely determines d and ℓ) as unnecessarily restrictive and hence we decoupled the access structure from the window size.

Another initiative to make efficiency gains was to use a ramp scheme, rather than a perfect threshold scheme. This remains in the information theoretic security setting, but leaks a measurable amount of information in return for smaller share sizes. In a similar vein, Juels *et al.* suggest that, instead of a perfect threshold scheme, their TSS scheme could be used. As is argued in Section 6.5.2.1, either a ramp scheme or Krawczyk's HK0 would be a more straightforward approach than the TSS, dependent on whether a large secret or just a key is being dispersed. If a ramp scheme is used instead of the TSS, this would align with our fLMSS more tightly. Alternatively, a computationally secure threshold scheme could be used in place of either the perfect threshold scheme (in our LMSS), or the ramp scheme (in our fLMSS). We note that the definition of a (t, x)-SWISS scheme with parameter Ψ, used with a perfect threshold scheme is a $(t-1, t; n, n, x/\Psi)$-fLMSS, where $n = (\Psi + 1)x/\Psi$.

Thus, our fLMSS is a more flexible solution than the SWISS scheme proposed by Juels *et al.* in [56].

6.6 Conclusion

In this chapter we introduced the concept of LMSSs, which are a special case of multi-secret sharing schemes and a natural concept with a range of potential applications. We have presented a number of constructions, analysed the security and considered their bounds.

We then discussed relaxing the security requirements in order to construct flexible LMSSs. These schemes provide a flexible and lightweight tool for approximating the ideal behaviour of a (t, n)-LMSS in a restricted environment, such as for use in RFID enabled supply chains. We highlight the advantages of using a fLMSS for this application to be:

- The direct use of a ramp scheme rather than an arbitrary ECC (as in the SWISS scheme [56]), explicitly gives the values of the important parameters, so that the resulting trade-off between security and efficiency is entirely clear. Furthermore, the use of Construction 2.3.13 in the fLMSS gives the essential property that both the sharing and the secret recovery can be efficiently performed.
- The fLMSS provides information theoretic security rather than relying on computational assumptions (in contrast to the TSS, which is secure only in the RO model).
- The use of Construction 6.4.1 reduces the number of unauthorised sets who can access a given secret relative to a SWISS scheme of analogous parameters.
- By separating the window length ℓ from the offset d, we have enabled a more flexible choice of parameters that allows for the appropriate security/efficiency trade-off to be chosen to directly suit application requirements.

The systematic analysis of the various components of the fLMSS in Section 6.4 ensures the trade-offs inherent in the selection of parameters are explicit and well-understood, making the fLMSS a widely applicable tool for applications of this nature.

Chapter 7

Concluding Remarks

Throughout this thesis, we focused on threshold schemes in different security settings. We have considered enhancements to threshold schemes, including robust extensions, repairable threshold schemes and localised threshold multi-secret sharing schemes, and explored their efficiencies, which included analysing the memory and computational requirements and communication bandwidth. We considered a number of ways in which security can be compromised in order to achieve more efficient schemes and analysed this trade-off in different settings.

We began with Chapter 2, where the necessary security notions and definitions were presented. This included defining symmetric-key encryption schemes, secret sharing schemes, threshold schemes, information dispersal algorithms and all relevant security notions.

In Chapter 3, we presented a perfect, ideal threshold scheme, called the modified HP scheme, based on an element of a computationally secure threshold scheme from a patent by HP [21]. We defined the scheme, then analysed both the security and efficiency. We then evaluated a number of other ideal, perfect threshold schemes in the literature, including Shamir's threshold scheme [91] and a scheme by Kurihara *et al.* [62]. We showed the modified HP share algorithm required fewer bitwise operations than Shamir's share algorithm, and fewer bitwise operations for the recover algorithm than both the schemes by Shamir and Kurihara *et al.*, assuming pre-computation is unavailable for recovery.

Going forwards, it would be interesting to implement the different schemes on different hardware and compare the running times for both share and recover algorithms over a range of parameters. It would also be constructive to widen the range of schemes included in the comparison. In particular, Chapter 3 focused on perfect threshold schemes and compared the modified HP scheme to other ideal, perfect threshold schemes. Rather than considering relaxing the security in order to gain more efficient schemes, as is explored in other chapters, we could explore the trade-offs between different efficiency metrics. For example, there may be non-ideal, perfect threshold schemes requiring fewer bitwise operations than the modified HP scheme, or perfect threshold schemes requiring more than minimal randomness with fewer bitwise operations. A full survey considering all efficient, perfect threshold schemes, not just ideal ones, would be worthwhile.

Chapter 4 explored a computationally secure threshold scheme proposed by Plank and Resch in [85], called AONT-RS. We presented a generalised version of the scheme, called AONT-RS0, that allowed for flexibility over choice of IDA and the block mode of operation and specified the necessary assumptions on the symmetric-key encryption scheme, the size of the ciphertext, the hash function and the IDA in order for the scheme to be secure. We proved AONT-RS0 to be computationally secure in the RO model. After analysing the security, we proceeded to analyse the efficiency of AONT-RS0 and compared it to Krawczyk's HK0 scheme, presented in [60]. We concluded AONT-RS0 is more efficient, in terms of memory requirements and computational complexity, than HK0, but not as secure, due to the assumptions made in the proof of AONT-RS0 (in particular, the assumption that the hash function is indistinguishable from a random oracle) compared to the standard assumptions made in the proof of HK0.

We then briefly discussed extending AONT-RS0 to be robust by using two techniques: first hash functions, then commitment schemes. The extension using hash functions is private and recoverable in the RO model, whereas the technique using commitment schemes is private in the RO model and recoverable under standard assumptions. As before, the scheme proven to be secure in the RO model is more efficient, but is less secure than the scheme proven under standard assumptions.

Again, implementing these schemes, both the non-robust and robust versions, and comparing their running times under a range of parameters would be an interesting activity. Also, we considered two methods of extending AONT-RS0 to be robust. There exist other methods of robust extensions in the literature; a methodical survey considering a wider range of techniques and comparing their efficiencies would be worthwhile.

Repairable threshold schemes were considered in Chapter 5, which are threshold schemes that enable a player to securely reconstruct a lost share with help from their peers. We summarised and refined existing RTSs in the literature and introduced a new parameter for analysis, called the repair metric. We highlighted the direct trade-off between the information rate and communication complexity, first noted in [97], then explored using regenerating codes as RTSs and found them to be immediately applicable. In particular, MBR codes appeared to present the best candidate solutions for RTSs that prioritised communication complexity, whilst our refined version of the enrolment RTS from [97] was the best solution for an RTS that prioritised information rate.

All RTSs proposed achieve perfect security. Defining computationally secure RTSs and finding constructions in this security model would be interesting. As an example, one could consider extending computationally secure threshold schemes, such as either HK0 or AONT-RS0 from Chapter 4, to be repairable. To then explore the memory requirements and communication complexity and define how the trade-off between these two metrics translates to the computational security model, could be a worthwhile research topic.

The concept of a localised threshold multi-secret sharing scheme (LMSS) was introduced in Chapter 6. Intuitively, an LMSS allows a sufficient number of players that are sufficiently close in an underlying order to reconstruct a secret. After presenting a formal definition, we explored the bounds on such schemes and presented a number of constructions achieving these bounds. We then methodically scrutinised ways to relax the security definition in order to beat these bounds. Finally, we combined these techniques to give a flexible LMSS, which is a a flexible and lightweight tool for approximating the ideal behaviour of an LMSS in a restricted environment. Inspired by Juela, Pappu and Parno [56], we then introduced the use of RFID tags in the supply chain and explored a couple of ways threshold schemes have been used in this setting. We briefly introduced the SWISS schemes suggested by Juels *et al.*, which were originally suggested for use in RFID enabled supply chains, and showed how our fLMSS was a more flexible version of the SWISS schemes.

Considering further applications of LMSSs would be interesting; we imagine that, given the natural interpretation of an LMSS as a sequence of secrets that change over time, there are a number of interesting applications besides RFID enabled supply chains. Additionally, further considering the use of a computationally secure threshold scheme in our fLMSS, in place of the ramp scheme, would be interesting as it is yet another way the security of the LMSS could be relaxed in order to achieve better bounds.

Given the increasing prevalence of constrained devices and the unusual applications they offer, there has been growing interest in efficient threshold schemes and their applications, making this a fascinating and worthwhile area of research. However, there is still much exploration to be done in this field, exploring both threshold schemes and their enhancements (including extensions mentioned, such as robust and repairable schemes, and those we have not mentioned, such as verifiable and proactive schemes).

Bibliography

[1] S. Abughazalah, K. Markantonakis, and K. Mayes. Enhancing the key distribution model in the RFID-enabled supply chains. In *Proceedings of the International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 871–878. IEEE, 2014.

[2] A. V. Aho and J. E. Hopcroft. *The design and analysis of computer algorithms.* Pearson Education India, 1974.

[3] N. Alon and M. Luby. A linear time erasure-resilient code with nearly optimal recovery. *IEEE Transactions on Information Theory*, 42(6):1732–1736, 1996.

[4] ARM. Cortex-M3, Revision r2p0: Technical reference manual, 2010. Available at `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0337i/CHDDIGAC.html`.

[5] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 394–403. IEEE, 1997.

[6] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and Communications Security (CCS)*, pages 62–73. ACM, 1993.

[7] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Proceedings of EUROCRYPT: Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer Berlin, 2006.

[8] M. Bellare and P. Rogaway. Robust computational secret sharing and a unified account of classical secret-sharing goals. In *Proceedings of the ACM conference on Computer and Communications Security (CCS)*, pages 172–184. ACM, 2007.

[9] J. C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Proceedings of Advances in Cryptology (CRYPTO): Conference on the Theory and Application of Cryptographic Techniques*, volume 263 of *Lecture Notes in Computer Science*, pages 251–260. Springer, 1986.

[10] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference (AFIPS)*, volume 48, pages 313–317, 1979.

[11] G. R. Blakley and C. A. Meadows. Security of ramp schemes. In *Proceedings of Advances in Cryptology (CRYPTO): Workshop on the Theory and Application of Cryptographic Techniques*, volume 196 of *Lecture Notes in Computer Science*, pages 242–268. Springer, 1984.

[12] M. Blaum and R. Roth. New array codes for multiple phased burst correction. *IEEE Transactions on Information Theory*, 39(1):66–77, 1993.

[13] C. Blundo, A. De Santis, G. D. Crescenzo, A. G. Gaggia, and U. Vaccaro. Multi-secret sharing schemes. In *Proceedings of Advances in Cryptology (CRYPTO): Annual International Cryptology Conference*, volume 839 of *Lecture Notes in Computer Science*, pages 150–163. Springer, 1994.

[14] C. Blundo, A. De Santis, and U. Vaccaro. Efficient sharing of many secrets. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 665 of *Lecture Notes in Computer Science*, pages 692–703. Springer, 1993.

[15] C. Blundo, A. De Santis, and U. Vaccaro. Randomness in distribution protocols. *Information and Computation*, 131(2):111–139, 1996.

[16] A. Bogdanov, S. Guo, and I. Komargodski. Threshold secret sharing requires a linear size alphabet. In *Proceedings of Theory of Cryptography Conference*, volume 9986 of *Lecture Notes in Computer Science*, pages 471–484. Springer, 2016.

[17] A. Bogdanov, Y. Ishai, E. Viola, and C. Williamson. Bounded indistinguishability and the complexity of recovering secrets. In *Proceedings of Advances in Cryptology*

(CRYPTO): Annual International Cryptology Conference, volume 9816 of *Lecture Notes in Computer Science*, pages 593–618. Springer, 2016.

[18] G. Boole. On the theory of probabilities. *Philosophical Transactions of the Royal Society of London*, 152:225–252, 1862.

[19] J. F. Boyar, S. A. Kurtz, and M. W. Krentel. A discrete logarithm implementation of perfect zero-knowledge blobs. *Journal of Cryptology*, 2(2):63–76, 1990.

[20] E. F. Brickell. Some ideal secret sharing schemes. In *Proceedings of EUROCRYPT: Workshop on the Theory and Application of Cryptographic Techniques*, volume 665 of *Lecture Notes in Computer Science*, pages 468–475. Springer, 1989.

[21] P. Camble, L. Chen, I. Henry, and M. Watkins. Utilizing error correction (ECC) for secure secret sharing. Hewlett Packard Enterprise Development LP, 2016. World Intellectual Property Organisation Patent Number WO2016048297.

[22] R. M. Capocelli, A. D. Santis, L. Gargano, and U. Vaccaro. On the size of shares for secret sharing schemes. *Journal of Cryptology*, 6(3):157–167, 1993.

[23] A. Chandrasekara, R. Bala, and G. Landers. Critical Capabilities for Object Storage. Technical report, Gartner, 2016. Available at `https://www.gartner.com/doc/reprints?ct=160413&id=1-33E2S6I&st=sb`.

[24] H. Chen and R. Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In *Proceedings of Advances in Cryptology (CRYPTO): Annual International Cryptology Conference*, volume 4117 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2006.

[25] L. Chen, T. M. Laing, and K. M. Martin. Efficient, "XOR"-based, ideal (t, n)-threshold schemes. In *Proceedings of International Conference on Cryptology and Network Security (CANS)*, volume 10052 of *Lecture Notes in Computer Science*, pages 467–483. Springer, 2016.

[26] L. Chen, T. M. Laing, and K. M. Martin. Revisiting and extending the AONT-RS scheme: a robust computationally secure secret sharing scheme. In *Proceedings of AFRICACRYPT: International Conference on Cryptology in Africa*, volume 10239 of *Lecture Notes in Computer Science*, pages 40–57. Springer, 2017.

[27] B. Chor and E. Kushilevitz. A communication-privacy tradeoff for modular addition. *Information Processing Letters*, 45(4):205–210, 1993.

[28] R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Proceedings of EUROCRYPT: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.

[29] R. Cramer, I. B. Damgård, N. Döttling, S. Fehr, and G. Spini. Linear secret sharing schemes from error correcting codes and universal hash functions. In *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 313–336. Springer, 2015.

[30] J. Daemen and V. Rijmen. *The design of Rijndael: AES - the Advanced Encryption Standard*. Springer Science & Business Media, 2002.

[31] I. B. Damgård, T. P. Pedersen, and B. Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In *Proceedings of Advances in Cryptology (CRYPTO): Annual International Cryptology Conference*, volume 773 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 1993.

[32] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[33] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551, 2010.

[34] EPCglobal. EPC radio-frequency identity protocols: Class-1 generation-2 UHF RFID protocol for communications at 860 MHz–960 MHz, version 1.0.9. *Specification for RFID Air Interface*, 2004.

[35] O. Farràs, T. B. Hansen, T. Kaced, and C. Padró. On the information ratio of non-perfect secret sharing schemes. *Algorithmica*, 79(4):987–1013, 2017.

[36] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings of Advances in Cryptology CRYPTO: Conference on the Theory and Application of Cryptographic Techniques*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[37] FIPS. SHA-3 standard: Permutation-based hash and extendable-output functions. *Federal Information Processing Standard, NIST FIPS 202*, 2015. Available at `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf`.

[38] E. Fujisaki and T. Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *Proceedings of EUROCRYPT: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 1403 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1998.

[39] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 464–479. IEEE, 1984.

[40] O. Goldreich, S. Goldwasser, and S. Micali. On the cryptographic applications of random functions. In *Proceedings of Advances in Cryptology (CRYPTO): Workshop on the Theory and Application of Cryptographic Techniques*, volume 196 of *Lecture Notes in Computer Science*, pages 276–288. Springer, 1985.

[41] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, 1996.

[42] S. Goparaju, S. El Rouayheb, R. Calderbank, and H. V. Poor. Data secrecy in distributed storage systems under exact repair. In *Proceedings of International Symposium on Network Coding (NetCod)*, pages 1–6. IEEE, 2013.

[43] S. Goparaju, A. Fazeli, and A. Vardy. Minimum storage regenerating codes for all parameters. In *Proceedings of IEEE International Symposium on Information Theory (ISIT)*, pages 76–80, 2016.

[44] X. Guang, J. Lu, and F. W. Fu. Repairable threshold secret sharing schemes. *Computing Research Repository (CoRR): arXiv preprint, arXiv:1410.7190*, 2014.

[45] S. Halevi and S. Micali. Practical and provably-secure commitment schemes from collision-free hashing. In *Proceedings of Advances in Cryptology (CRYPTO): Annual International Cryptology Conference*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 1996.

[46] J. Halpern and V. Teague. Rational secret sharing and multiparty computation. In *Proceedings of the annual ACM symposium on Theory of computing (STOC)*, pages 623–632. ACM, 2004.

[47] J. Herranz, A. Ruiz, and G. Sáez. New results and applications for multi-secret sharing schemes. *Designs, Codes and Cryptography*, 73(3):1–24, 2013.

[48] IBM. IBM Cloud Object Storage, 2016. *Available at:* `https://www.cleversafe.com/platform/why-ibm-cloud-object-storage`.

[49] S. Iftene. General secret sharing based on the Chinese remainder theorem with applications in E-voting. *Electronic Notes in Theoretical Computer Science*, 186:67–84, 2007.

[50] ISO/IEC 10116. Information technology – Security techniques – Modes of operation for an n-bit block cipher algorithm.

[51] ISO/IEC 10118-3. Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions.

[52] ISO/IEC 19592-2. Information technology – Security techniques – Secret sharing, 2017.

[53] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *Proceedings of IEEE Globecom*, pages 99–102, 1987.

[54] W. A. Jackson and K. M. Martin. A combinatorial interpretation of ramp schemes. *Australasian Journal of Combinatorics*, 14:51–60, 1996.

[55] W. A. Jackson, K. M. Martin, and C. M. O'Keefe. Multisecret threshold schemes. In *Proceedings of Advances in Cryptology CRYPTO: Annual International Cryptology Conference*, volume 773 of *Lecture Notes in Computer Science*, pages 126–135, 1993.

[56] A. Juels, R. Pappu, and B. Parno. Unidirectional key distribution across time and space with applications to RFID security. In *Proceedings of USENIX Security Symposium*, pages 75–90, 2008.

[57] A. A. Karatsuba. The complexity of computations. *Proceedings of the Steklov Institute of Mathematics*, 211:169–183, 1995.

[58] E. D. Karnin, J. W. Greene, and M. E. Hellman. On secret sharing systems. *IEEE Transactions on Information Theory*, 29(1):35–41, 1983.

[59] D. Knuth. *The Art of Computer Programming vol. 2: Seminumerical Algorithms, 3/E.* Pearson Education, 1998.

[60] H. Krawczyk. Secret sharing made short. In *Proceedings of Advances in Cryptology CRYPTO: Annual International Cryptology Conference*, volume 773 of *Lecture Notes in Computer Science*, pages 136–146. Springer, 1993.

[61] J. Kurihara, S. Kiyomoto, K. Fukushima, and T. Tanaka. A fast $(3, n)$-threshold secret sharing scheme using exclusive-or operations. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 91(1):127–138, 2008.

[62] J. Kurihara, S. Kiyomoto, K. Fukushima, and T. Tanaka. A new (k, n)-threshold secret sharing scheme and its extension. In *Proceedings of ISC: International Conference on Information Security*, volume 5222 of *Lecture Notes in Computer Science*, pages 455–470. Springer, 2008.

[63] T. M. Laing, K. M. Martin, M. B. Paterson, and D. R. Stinson. Localised multisecret sharing. *Cryptography and Communications*, 9(5):581–597, 2016.

[64] T. M. Laing and D. R. Stinson. A survey and refinement of repairable threshold schemes. *Journal of Mathematical Cryptology.* In press.

[65] M. Langheinrich and R. Marti. Practical minimalist cryptography for RFID privacy. *IEEE Systems Journal*, 1(2):115–128, 2007.

[66] S. J. Lin and W. H. Chung. An efficient (n, k) information dispersal algorithm based on Fermat number transforms. *IEEE Transactions on Information Forensics and Security*, 8(8):1371–1383, 2013.

[67] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.

[68] C. Lv, X. Jia, J. Lin, J. Jing, L. Tian, and M. Sun. Efficient secret sharing schemes. In *Proceedings of Secure and Trust Computing, Data Management and Applications*, volume 665 of *Communications in Computer and Information Science*, pages 114–121. Springer, 2011.

[69] C. Lv, X. Jia, L. Tian, J. Jing, and M. Sun. Efficient ideal threshold secret sharing schemes based on EXCLUSIVE-OR operations. In *Proceedings of International Conference on Network and System Security (NSS)*, pages 136–143. IEEE, 2010.

[70] F. MacWilliams and N. Sloane. *The theory of error correcting codes.* Elsevier, 1977.

[71] J. Massey. Minimal codewords and secret sharing. In *Proceedings of the Joint Swedish-Russian International Workshop on Information Theory*, pages 276–279, 1993.

[72] B. Masucci. Sharing multiple secrets: Models, schemes and analysis. *Designs, Codes and Cryptography*, 39(1):89–111, 2006.

[73] R. J. McEliece and D. V. Sarwate. On sharing secrets and Reed-Solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.

[74] NIST. Recommendation for block cipher modes of operation. National Institute of Standards and Technology Special Publication 800-38A, 2001.

[75] M. Nojoumian. *Novel Secret Sharing and Commitment Schemes for Cryptographic Applications*. PhD thesis, University of Waterloo, 2012.

[76] M. Nojoumian, D. R. Stinson, and M. Grainger. Unconditionally secure social secret sharing scheme. *IET Information Security*, 4(4):202–211, 2010.

[77] M. B. Paterson and D. R. Stinson. A simple combinatorial treatment of constructions and threshold gaps of ramp schemes. *Cryptography and Communications*, 5(4):229–240, 2013.

[78] S. Pawar, S. El Rouayheb, and K. Ramchandran. On secure distributed data storage under repair dynamics. In *Proceedings of the IEEE Symposium on Information Theory Proceedings (ISIT)*, pages 2543–2547. IEEE, 2010.

[79] F. P. Preparata. Holographic dispersal and recovery of information. *IEEE Transactions on Information Theory*, 35(5):1123–1124, 1989.

[80] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

[81] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the annual ACM symposium on Theory of Computing*, pages 73–85. ACM, 1989.

[82] K. V. Rashmi, N. B. Shah, and P. V. Kumar. Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction. *IEEE Transactions on Information Theory*, 57(8):5227–5239, 2011.

[83] A. S. Rawat. A note on secure minimum storage regenerating codes. *Computing Research Repository (CoRR): arXiv preprint, arXiv:1608.01732*, 2016.

[84] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[85] J. K. Resch and J. S. Plank. AONT-RS: blending security and performance in dispersed storage systems. In *Proceedings of FAST-2011: a USENIX Conference on File and Storage Technologies.*

[86] R. L. Rivest. All-or-nothing encryption and the package transform. In *Proceedings of the International Workshop on Fast Software Encryption (FSE)*, pages 210–218. Springer, 1997.

[87] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[88] C. M. Roberts. Radio frequency identification (RFID). *Computers & Security*, 25(1):18–26, 2006.

[89] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Proceedings of Advances in Cryptology CRYPTO: Annual International Cryptology Conference*, volume 1666 of *Lecture Notes in Computer Science*, pages 148–164. Springer, 1999.

[90] N. B. Shah, K. Rashmi, and P. V. Kumar. Information-theoretically secure regenerating codes for distributed storage. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5. IEEE, 2011.

[91] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[92] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[93] R. Singleton. Maximum distance q-nary codes. *IEEE Transactions on Information Theory*, 10(2):116–118, 1964.

[94] A. Soro and J. Lacan. FNT-based Reed-Solomon erasure codes. In *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, pages 1–5. IEEE, 2010.

[95] D. R. Stinson. An explication of secret sharing schemes. *Designs, Codes and Cryptography*, 2(4):357–390, 1992.

[96] D. R. Stinson. *Combinatorial Designs: Constructions and Analysis*. Springer-Verlag, New York, Inc., 2004.

[97] D. R. Stinson and R. Wei. Combinatorial repairability for threshold schemes. *Designs, Codes and Cryptography*, 86(1):1–16, 2017.

[98] M. Tompa and H. Woll. How to share a secret with cheaters. *Journal of Cryptology*, 1(3):133–138, 1989.

[99] Y. Wang and Y. Desmedt. Efficient secret sharing schemes achieving optimal information rate. In *Proceedings of the IEEE Information Theory Workshop (ITW)*, pages 516–520. IEEE, 2014.

[100] M. Ye and A. Barg. Explicit constructions of high-rate MDS array codes with optimal repair bandwidth. *IEEE Transactions on Information Theory*, 63(4):2001–2014, 2017.