Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Master in Advanced Mathematics and Mathematical Engineering
Master's thesis

On publicly verifiable secret sharing schemes

Jérémy Lavollée

Supervised by Carles Padró

January 2023

I thank Carles Padró for his invaluable guidance in the research and presentation of this thesis. I also thank Ignacio Cascudo for inviting me to an enlightening week of research with him and his team at IMDEA Software.

Abstract

Secret sharing allows a dealer to distribute shares of a secret to a set of parties such that only so-called authorised subsets of these parties can recover the secret, whilst forbidden sets gain at most some restricted amount of information. This idea has been built upon in verifiable secret sharing to allow parties to verify that their shares are valid and will therefore correctly reconstruct the same secret. This can then be further extended to publicly verifiable secret sharing by firstly considering only public channels of communication, hence imposing the need for encryption of the shares, and secondly by requiring that any party be able to verify any other parties shares from the public encryption.

In this thesis we work our way up from the original secret sharing scheme by Shamir to examples of various approaches of publicly verifiable schemes. Due to the need for encryption in private communication, different cryptographic methods allow for certain interesting advantages in the schemes. We review some important existing methods and their significant properties of interest, such as being homomorphic or efficiently verifiable. We also consider recent improvements in these schemes and make a contribution by showing that an encryption scheme by Castagnos and Laguillaumie allows for a publicly verifiable secret sharing scheme to have some interesting homomorphic properties. To explore further we look at generalisations to the recently introduced idea of Abelian secret sharing, and we consider some examples of such constructions. Finally we look at some applications of secret sharing schemes, and present our own implementation of Schoenmaker's scheme in Python, along with a voting system on which it is based.

Keywords

secret sharing, publicly verifiable, homomorphic, voting

Contents

1. Introduction

Secret sharing, as the name suggests, is the idea of sharing a secret among a group of parties. The party sharing the secret, known as the dealer, first generates a share for each party it will be sharing the secret with. Each share on its own should give no information about the secret, but if an adequate subset of parties work together then the combination of their shares will recover the secret. There is little restriction on what an 'adequate' subset can be here, and it is up to the dealer to decide which sets of parties are qualified and which are forbidden. One of the most common approach to this is simply to set a threshold number such that any subset with sufficiently many parties can recover the secret.

Secret sharing schemes are useful when the secret at hand is of high importance but the owner of the secret does not fully trust any of the parties to which it could give it. That is, instead of entrusting a single party with a secret, it sends to many parties some shares and it knows that they may only recover the secret if enough of them work together. For example when storing a password, maybe for a bank account that is too risky to write down but also too important to hope to remember it in one's heads. Then we could break this password into say five shares which could each be written down and stored in different locations. Supposing we generated the shares such that the threshold number to recover the secret was three, then even if two of the shares happen to be found by an intruder, they would still not be able to recover the password.

The first secret sharing schemes were actually two independently proposed schemes introduced in the same year in 1979 by Blakley [5] and Shamir [28]. The former took a geometric approach by assigning the secret to a point in space and setting the shares as hyperplanes, the idea being that the secret point is recoverable only by intersecting sufficiently many share hyperplanes. Shamir's approach on the other hand finds a random polynomial of sufficiently large degree such that its evaluation at zero is the secret. Then the shares are evaluations of said polynomial at different points and the recovery of the secret can be carried out by Lagrange interpolation on these points. In chapter 3 we consider these methods in more detail and note that of the two, Shamir's has become a very important basis from which to study secret sharing.

A natural extention to secret sharing schemes is to consider some of the parties and dealer to be dishonest or corrupt. For example a corrupt dealer could send shares which reconstruct to different secrets depending on the set of parties taking part in the reconstruction. In 1985 Chor introduced in [15] the idea of a verifiable secret sharing scheme, where parties could verify the validity of their shares and was soon followed with different methods by Benaloh [4] in 1986 and Feldman [16] in 1987. In chapter 4 we consider this type of scheme and how the parties can check that their shares are in-fact valid. We also consider the different types of security that these schemes may enjoy. In two main categories we have the information-theoretic security which means even computationally unbounded adversaries cannot recover the secret, and the weaker computational security which says that the scheme is secure against adversaries with a certain bounded computational power. For the rest of this chapter we consider a specific example of a verifiable secret sharing scheme [13] which extends the idea of the Lagrange interpolation to allow for verifiability. This scheme is secure in the information-theoretic sense, whereas the rest of the schemes considered in the later chapters are computationally secure. The scheme works in a process of multiple rounds in which parties can complain about both the dealer and each other if some of their broadcasted values do not fit with their own. For each complaint there is a resolution process where sharing the right information allows the honest parties to weed out the corrupt ones. We see that once the whole process is finished then the honest parties have all recovered the same secret, which is the dealer's original secret if the latter is honest.

We present only a single example of verifiable secret sharing since the main interest of this thesis lies in the next iteration of secret sharing schemes that we consider. The aptly named publicly verifiable secret sharing model takes the idea of verifiable schemes and requires that they be usable in public spaces where no party trusts any other. Introduced in 1999 by Schoenmakers [27], publicly verifiable schemes offer two important changes. The first is that any party, actively taking part in the scheme or not, should be able to carry out the verifications that the parties are tasked with due to the verifiability of the scheme. That is any party, with only publicly available information should be able to verify that the shares that any other party receives are valid. This is often done by consider a new object in the scheme called a verifier whose only job is to carry out these checks. This requirement of public verifiability allows the schemes to potentially be used in real implementations, for example in a voting system, where it is ideal to allow anyone to check for themselves that the scheme is being carried out as it should be. In line with this proximity to real implementation, the other change coming from publicly verifiable schemes is to make use of an encryption scheme to send messages. Instead of how the previous schemes were run with the assumption that there were private channels of communication between any parties and the dealer, here we require messages to be encrypted. This allows the schemes defined in this way to be truly implementable. However we do note that it does not make the job of the verifier easier since the latter must now check that shares are valid from an encryption of these shares. A clever yet potentially costly method of making this possible is to ask the dealer to send, along with the shares, some proof of the validity of each share. In this way the verifier need only check that the proofs are correct and this confirms the validity of the shares.

In chapter 5 we define the publicly verifiable schemes and reconsider the correctness and privacy requirements in line with the new scheme. The first scheme we consider is Schoenmaker's [27] which is inspired by Shamir and is a basis of many later schemes. Then in order to present the Ruiz and Villar scheme introduced in [26] we must first consider a specific cryptosystem from which we use the encryption scheme. The Paillier cryptosystem [24] considers the difficulty of computing N-th residue classes to construct a homomorphic encryption scheme. The scheme was later adapted to a publicly verifiable secret sharing scheme by Ruiz and Villar in 2005, making use of the homomorphic property. The scheme is based on the decisional composite residuosity assumption from the Paillier cryptosystem and boasts non-interactive verification without the need for the Random Oracle Model assumption or other add-hoc methods. Finally we end the chapter with another interesting publicly verifiable scheme, this one making use of bilinear maps. That is, a well-behaved and efficiently computable bilinear map between groups from which the equivalence to the Diffie-Hellman problem is considered hard, namely the Bilinear Diffie-Hellman assumption. The scheme presented in [20] in 2009 makes clever use of the bilinearity of the map for efficient verifications which do not require zero-knowledge proofs or Fiat-Shamir heuristics. We also use this scheme to introduce the idea of an adaptive adversary which can corrupt parties while the scheme is running. Finally we consider the homomorphic property of this scheme which is made surprisingly simple thanks to the bilinear map, meaning that any linear combination of shares is valid to recover the same linear combination of their respective secrets.

In chapter 6 we consider recent improvements in the study of publicly verifiable schemes and make our own contribution. First we follow the ideas presented in SCRAPE [7] to adapt the verification method of the Schoenmaker scheme. The idea makes use of the close relation between Shamir's secret sharing and Reed-Solomon codes to allow for efficient verification of the validity of shares. This boils down to a single equation between commitments and a randomly chosen codeword in the dual of the code which is equivalent to an inner product between the code and its dual. We then consider an adaptation of ElGamal encryption introduced in [11] by Castagnos and Laguillaumie which we refer to as CL15. This adaptation comes from the fact that linearly homomorphic encryption is possible with exponential ElGamal where the messages are in the exponent and hence taking the product results in the encryption for the sum of messages, but it is limited by the fact that to recover the message we must then compute the discrete log of the encryption. This is viable if it is known that the message in the exponent is small, but otherwise it has no computable solution. To remedy this the CL15 encryption considers an adapted setup with a group where the Diffie-Hellman problem is hard but which has a subgroup where the discrete log is easy. This allows for easy decryption by encrypting to the given subgroup. The CL15 encrytion scheme also allows for the interesting property that the decryption is linear in the secret keys. This is used by the parties to prove to each other that they have correctly decrypted their shares. Once we have all these adapted methods we can consider the DHPVSS scheme with our contribution of using the CL15 encryption instead of ElGamal, although the two work in very similar ways.

Chapter 7 returns to the standard secret sharing scheme and considers how these can be generalised. First we consider a generalisation by Schoenmaker from a threshold scheme to an arbitrary linear scheme, making use of the close relation between secret sharing schemes and linear codes. We then look at Abelian secret sharing which allows us to construct schemes from a set of Abelian groups and a subgroup code. We consider three such related constructions which make use of the duals of the code. Of particular interest here is to see how the authorised and forbidden sets can be expressed in terms of the Abelian groups and their duals. This use of generalising provides a better understanding of the existing schemes, and with further work could provide new methods for secret sharing.

The final chapter in this thesis considers some ways in which secret sharing schemes can be applied. First we look at examples of schemes being used in protocols such as the average consensus problem and time-locking secrets. Secret sharing schemes are often used in protocols where it cannot be assumed that parties trust each other or that there is a trusted third party. Finally we consider an implementation of a publicly verifiable secret sharing scheme and its adaptation to construct a voting system. To bridge the gap between the theory and the more applied side we chose to write Schoenmaker's scheme in Python to see how the application fairs in practice. It is implemented in a straight-forward way with minimal use of libraries so that we can see the inner workings of the code. In the chapter we present the most important parts of the code and explain the processes that it follows, the full code being available at https://github.com/JeremyLvl/TFM. The second implementation is from the same Schoenmaker paper which presents a voting system based on the publicly verifiable scheme. In this case the voters are dealers which distribute their shares to talliers, or the parties in the secret sharing scheme. Using the homomorphic properties of the encryption allows the talliers to recover the sum of votes without having to decypher the votes directly. That is, the talliers can put together the shares they receive for each vote to produce a single share for the sum of votes. For both implementations we also present a minimal working example to run through and show how the whole program works.

2. Cryptographic prerequisites

For the coming discussions it will be useful to first introduce some basic cryptographic concepts. In this section we present the very basics of the necessary topics for the schemes that we will consider later. As we will see, secret sharing is very close to codes and we will make use specifically of Reed-Solomon codes. For publicly verifiable secret sharing schemes we will also need hashing to be able to use zero-knowledge proofs, and public key encryption for the parties to communicate privately.

2.1 Encryption

Let us first define symmetric and public key encryption. Following convention we call the sender of the message Alice, denoted A, and the receiver Bob, denoted B. In short, an encryption scheme allows for A to publicly send an encrypted version of a message B, such that only the parties with the correct decryption keys may recover the original message. As the name suggests, a symmetric key encryption scheme assumes that both A and B share the same key with which they encrypt the message and decipher the ciphertext.

Let us fix some generic domains for simplicity. Let $\mathcal{M}$ be the domain of messages, $\mathcal{C}$ the domain of ciphertexts and $\mathcal{K}$ the domain for keys. For the encryption and decryption processes, we also need to differentiate between deterministic and randomised algorithms. So let a deterministic algorithm be one which always produces the same result given the same inputs, whereas a randomised algorithm has some form of randomisation and may produce a different result for the same inputs.

Definition 2.1. A **symmetric key encryption** scheme is a pair of algorithms (Enc, Dec) defined over $(\mathcal{M}, \mathcal{C}, \mathcal{K})$ which are as follows:

- $Enc(m, k)$: a randomised algorithm which encrypts a message $m \in \mathcal{M}$ to a ciphertext in $\mathcal{C}$,
- $Dec(c, k)$ a deterministic algorithm which decrypts a ciphertext $c \in \mathcal{C}$ to a message in $\mathcal{M}$.

The decryption algorithm returns the original message if the same key is used on the encrypted version of the message, that is, $Dec(Enc(m, k), k) = m$.

The requirement that both parties already know a shared key k is strong, in order to avoid it, public key encryption instead uses two different keys for encryption and decryption. A public key is known to both parties and a secret key is known only to the receiver. This way, encryption using the public key can only be decrypted using the adequate private key.

Definition 2.2. A **public key encryption scheme** ε, is a set of three algorithms as follows:

- $Gen()$: a randomised algorithm that outputs a pair (pk, sk) of public/private key,
- $Enc(m, pk)$: a randomised algorithm which encrypts a message $m \in \mathcal{M}$ given a public key and outputs the ciphertext $c \in \mathcal{C}$,
- $Dec(c, sk)$: a deterministic algorithm which decrypts the ciphertext c and outputs m if $c = Enc(m, pk)$ and (pk, sk) is a valid key pair from Gen.

Now for the security of these encryption schemes. Using the idea of attack games we will define an encryption scheme to be secure if an adversary cannot tell the difference between two encryptions. More specifically, an attack game runs as follows. An adversary chooses any two messages and sends them to the challenger, who will choose one at random, encrypt it and send it back. The adversary then has to guess which of the two messages was encrypted. If the adversary can do this with a probability significantly different to a half, then the encryption scheme is insecure, since clearly some information is visible through the encryption. We define only the attack game for the public key encryption scheme, from which the symmetric equivalent is clear.

A note on notation in the attack game. Writing $a \leftarrow Algo()$ means running the algorithm $Algo()$ and setting a to the output. Assigning $b \in_R S$ means setting b to some value in S uniformly at random.

Definition 2.3. The semantic security attack game between the challenger *Chal* and the adversary *Adv* runs as follows, where *Adv* wins if $\hat{b} = b$.

$$\begin{array}{ccc}
Chal & & Adv \\
\hline
(pk, sk) \leftarrow Gen() & & \\
 & \xrightarrow{pk} & \\
 & & m_0, m_1 \in \mathcal{M} \\
 & \xleftarrow{m_0,\, m_1} & \\
b \in_R \{0,1\} & & \\
c \leftarrow Enc(m_b, pk) & & \\
 & \xrightarrow{c} & \\
 & & \hat{b} \in \{0,1\}
\end{array}$$

Finally we can define the security of the scheme. Note that this depends on the computational power of the adversary. In short we allow any **polynomial time** adversary, ie. one who's algorithms runs in polynomial time on the size of the input. So we can claim semantic security if such an adversary cannot say which message was encrypted better than simply guessing.

Definition 2.4. A public key encryption scheme is **semantically secure** if for any polynomial time adversary then $|\mathbb{P}(\hat{b} = b) - 1/2| < \varepsilon$.

2.2 Hashing

Let us now introduce the very useful concept of a **hash function**. We keep the definition of a hash function quite open to allow for it to be use in many different contexts. The idea is to have a deterministic one-way function whose output appears random. By one-way here we mean that it is infeasible for a polynomial time algorithm given $H(x)$ to recover x. By the randomness of the output we mean that any small change in the input should produce a large change in the output, such that the similitude of the two inputs is not easily seen.

A common heuristic to consider is the **Random Oracle Model**, or ROM. This says that it is possible to model a hash function as having a truly random output, that is, it is uniformly random in the image set. It is an idealisation of the idea of hashing and clearly a proof in the random oracle model does not imply security in the standard cryptography model, where hashing is not truly random. However we consider it anyway since it is useful to be able to construct proofs of security in the ROM.

2.3 Zero knowledge proofs

A **zero-knowledge proof** is a protocol run between a prover P and a verifier V such that the prover convinces the verifier of a specific claim, but from which the verifier gains no other information other than the knowledge that the claim is true. The interactive model is carried out by both parties sending each other messages, the final list of which is called the **transcript** of the proof.

Definition 2.5. A zero-knowledge proof with prover P and verifier V must satisfy:

- correctness: V accepts the proof in the case that both parties are honest,
- soundness: an honest V accepts a proof from a dishonest P with bounded probability,
- zero-knowledge: V learns nothing from the proof other than that the claim is true.

For the zero-knowledge requirement we consider a **simulator** S of the prover which attempts to prove the claim is true but without any of the knowledge the prover has. If such a simulation of the proof can be sucessfully carried out without the required knowledge then indeed the verifier can learn nothing other than the truth of the claim, since it could itself have formed the simulated proof.

A **proof of knowledge** is zero-knowledge proof where the prover proves to the verifier that they know a certain value, often called a 'witness' of a claim. We will only consider proofs of knowledge of the form of sigma protocols, as seen in Figure 1. For such proofs the soundness property is called special soundness

$$\begin{array}{ccc} \textit{Prover} & & \textit{Verifier} \\ \hline \text{commitment} & \xrightarrow{a} & \\ & \xleftarrow{e} & \text{challenge} \\ \text{response} & \xrightarrow{z} & \end{array}$$

Figure 1: A generic sigma protocol proof of knowledge.

and uses the idea of a **knowledge extractor**. It is assumed that by exchanging messages with a prover, the extractor is able to recover a witness for the claim. More specifically, given two transcripts (a, e, z) and (a, e', z') with the same commitments and different challenges and responses, then the knowledge extractor can recover the witness from the prover.

Finally we consider the **Fiat-Shamir heuristic** which is often used in the construction of non-interactive zero-knowledge proofs, or NI-ZK for short. The idea is to take an interactive proof of knowledge and convert it into a digital signature, that is, a single message which convinces the verifier in the same way as the proof. More formally, consider the interactive proof in Figure 1, the Fiat-Shamir heuristic says that this can be done in a single signature from the prover, as in Figure 2. Note that this only works to convince

$$\begin{array}{ccc} \textit{Prover} & & \textit{Verifier} \\ \hline \text{commitment } a & & \\ e = H(a) & & \\ \text{response } z & & \\ & \xrightarrow{(a, z)} & \end{array}$$

Figure 2: A non-interactive zero-knowledge proof.

the verifier if it believes that the challenge the prover constructs is in fact random, and hence the verifier did not construct a specific e to allow themself to cheat. So this requires that hashing is truly random, since $e = H(a)$, and hence it is a heuristic in the random oracle model. As we will see, this can also be aided by sending the commitment and having the verifier check that the hash is correct.

2.4 Codes

Codes and especially linear codes will be useful constructions to have when considering secret sharing. We introduce here the basics that we will later need to make a parallel between secret sharing schemes and codes. Referring to a set $\mathbb{F}_q$ as an alphabet, a **codeword** is a sequence of symbols in $\mathbb{F}_q$, and a **code** C is a set of codewords. A **linear code** of length n is a linear subspace of a vector space $\mathbb{F}_q^n$, that is, any linear combination of codewords is also a codeword. The dimension of a linear code, $\dim(C)$, is the dimension of the subspace it is defined by. A code is s-**error-correcting** if given a codeword with up to s of its symbols changed, it can still recover the original codeword. The minimum distance of a code is the least number of symbols needed to change one codeword to another. We will focus on error-correcting linear codes and their duals, which is the set of codewords which are orthogonal to all words in the code.

Definition 2.6. An $[n, k, d]$ code is a linear error-correcting code over $\mathbb{F}_q$ of length n, dimension k and minimum distance d.

Definition 2.7. For an $[n, k, d]$ code C, the dual code $C^\perp$ is the vector space of $v \in \mathbb{F}_q^n$ such that $\langle v, c\rangle = 0$ for all $c \in C$.

Note that $\dim(C) + \dim(C^\perp) = n$ and the dual code is actually an $[n, n-k, d']$ code for some d'.

Reed-Solomon codes are $[n, k, n-k+1]$ codes introduced in [25] which can be described as having evaluations of a polynomial for as codewords,

$$C = \{(p(\alpha_1), \dots, p(\alpha_n)) : p(x) \in \mathbb{F}_q[x], \deg(p) \leq k-1\}.$$

And its dual code $C^\perp$ is an $[n, n-k, k+1]$ code described as

$$C^\perp = \{(v_1 f(\alpha_1), \dots, v_n f(\alpha_n)) : f(x) \in \mathbb{F}_q[x], \deg(f) \leq n-k-1\},$$

where $v_i = \prod_{j\neq i} \frac{1}{i-j}$.

3. Secret sharing

3.1 Introductory background

The basic idea of secret sharing is intuitively well-explained by its name. In short, it is the sharing of a secret between parties, which as is often assumed, do not necessarily trust each other, such that the secret is only recovered if a correct subset of the parties work together. In other words, consider a set of $n \geq 2$ parties or players and another special party which we call the dealer. The dealer has some secret, denoted s, which it wants to share in a certain way among the parties. It requires that on its own the share a party receives gives it no information about the secret, but specific subsets of the n parties can work together, combining their shares, to find the original secret s. A commonly used approach is to allow any set of parties to recover the secret if they are sufficiently numerous, that is, setting a threshold size for the minimum number of parties.

Secret sharing schemes, were first independently introduced in 1979 by Blakley [5] and Shamir [28], and both in the specific case of threshold secret sharing. Blakley approached this geometrically by considering the intersection of hyperplanes. That is, each share given to a party is a hyperplane such that the intersection of sufficiently many describes just a single point. The secret is one of the coordinates of this intersection point. Note that the intersection of too few hyperplanes gives a subspace of dimension at least two, from which the secret coordinate cannot be found. Although an interesting approach, we will not go further into it. The main problem is the size of the shares, which have to be multiple times larger than the secret. This can be improved by choosing appropriate planes to be shared, although this in turn transforms the scheme to be equivalent to the one described instead by Shamir. The latter approach is closer to the common methods of today and we will describe it detail shortly. The idea is simple, for a secret s, a polynomial $q(x)$ is constructed to have the secret as $s = q(0)$. The shares sent to players are distinct points evaluated on this polynomial, sufficiently many of which can be used by a Lagrange interpolation to recover the polynomial, and hence to recover the secret.

3.2 A secret sharing model

To define a secret sharing scheme we first need to more formally define the access structures which say which subsets of parties should be able to recover the secret. Let D be the dealer who wants to share its secret and $\mathcal{P} = \{P_1, \dots, P_n\}$ be the set of parties which will receive shares. A subset of parties is called **authorised** if it should be able to recover the secret, and **forbidden** otherwise. The collection of all authorised parties is called the **access structure** and denoted $\Gamma \subset 2^{\mathcal{P}}$. One thing that is clear immediately is that an access structure must be monotone. Naturally if $\mathcal{A}$ can find the secret then any set of parties which contain $\mathcal{A}$ can too. That is, for a set of parties $\mathcal{A} \in \Gamma$, if $\mathcal{A} \subset \mathcal{B}$ then $\mathcal{B} \in \Gamma$. Finally we say that for an (t, n)-threshold secret sharing scheme, the access structure can be described as $\Gamma = \{\mathcal{A} \subset 2^{\mathcal{P}} : |\mathcal{A}| \geq t\}$.

Following Beimel [2] we now define a secret sharing scheme by first defining distribution schemes. Note that we assume for now that there is a private communication channel for any pair of parties, including with the dealer.

Definition 3.1. A **distribution scheme** is a mapping $\Pi : S \times R \to S_0 \times \cdots \times S_n$, where S is a domain of secrets, R is a domain of random strings, and S_i is a domain of shares for P_i.

For ease of notation, given a set of parties $\mathcal{A} \subset \Gamma$, let $\Pi_{\mathcal{A}}(s, r)$ be the restriction of $\Pi(s, r)$ to the shares that parties of $\mathcal{A}$ receive.

Definition 3.2. A **secret sharing scheme** is a protocol between a dealer D and a set of parties $\mathcal{P} = \{P_1, \dots, P_n\}$. For a secret s the protocol runs as follows:

1. D randomly selects some $r \in R$,
2. D computes shares $\Pi(s, r) = (s_0, \dots, s_n)$,
3. D privately sends s_i to each party P_i respectively.

The scheme is said to realise an access structure Γ if the two following properties are satisfied:

- **correctness**: for any authorised set of parties $\mathcal{A} \in \Gamma$ there is a reconstruction function Rec : $\Pi_{\mathcal{A}}(s, r) \to S$ such that $\mathrm{Rec}(\Pi_{\mathcal{A}}(s, r)) = s$.
- **privacy**: for two secrets $s_0, s_1 \in S$ and a forbidden set of parties $\mathcal{B} \notin \Gamma$ the distributions of $\Pi_{\mathcal{B}}(s_0, r)$ and $\Pi_{\mathcal{B}}(s_1, r)$ are identical over the choice of r.

The correctness property here requires that, as expected, any authorised set of parties can reconstruct the secret from the shares they have received, hence the *Rec* function. The privacy property says that any forbidden set of parties should have no information about the secret from their collective shares. This is done by requiring that their shares are as likely to come from any two secrets.

Finally we define two often considered properties of secret sharing schemes which will be the case in most of the schemes in this work.

Definition 3.3. A secret sharing scheme is **perfect** if every possible set of parties is either authorised or forbidden.

Note that since we assume that a set is forbidden if it is not authorised, then we are already working with perfect schemes. This property essentially gets rid of the idea of sets of parties which gain some information about the secret from their shares, but not enough to fully recover the secret.

Definition 3.4. A secret sharing scheme is **ideal** if the size of the shares are equal to that of the secret, that is, $|S_i| = |S|$ for $i = 1, \dots, n$.

In this case the scheme is ideal because the secret does not get larger in size while being shared. As we will see immediately below, this is the case for Shamir's scheme which uses the same field $\mathbb{F}_q$ for both the secrets and the shares.

3.3 Shamir's threshold scheme

We can now describe Shamir's well-known (t, n)-threshold secret sharing scheme as introduced in [28]. Recall the access structure here authorises any set with t or more parties. Consider a secret s in some field $\mathbb{F}_q$, and note that for the rest of this section we will be working in this field. The dealer selects a random polynomial $q(x)$ of degree $t-1$ such that $q(0) = s$. The shares are evaluations of $q(x)$ at distinct points, for example it is common to give to party P_i the share $s_i = q(i)$. Note that in this case Shamir's scheme is a linear secret sharing scheme meaning that the secret can be written as a linear combination of the shares, we will see how later in this section. By Lagrange interpolation, and since $q(x)$ has degree $t-1$, a set of at least t shares are enough to uniquely determine $q(x)$, from which the secret can be recovered. This last statement will be proved in the general case, which is nothing more than the Lagrange interpolation theorem.

For ease of notation, when we consider a set of say k parties and their shares, we will assume that these are the first k parties $P_1, \dots, P_k$, and hence their shares are $s_1, \dots, s_k$. This will save us from double indexing.

Note that this can simply be thought of as a re-indexing of all the parties and their shares and is without loss of generality.

Definition 3.5. The (t, n) Shamir threshold secret sharing scheme is a protocol with the distribution scheme

$$\Pi : \mathbb{F}_q \times \mathbb{F}_q^t \to \mathbb{F}_q \times \cdots \times \mathbb{F}_q$$

where $q > n$ and the access structure is $\Gamma = \{\mathcal{A} \subset \mathcal{P} : |\mathcal{A}| \geq t\}$. For a secret $s \in \mathbb{F}_q$ the sharing protocol for the dealer is as follows:

1. D randomly samples $t-1$ elements $\alpha_1, \dots, \alpha_{t-1} \in \mathbb{F}_q$,
2. D constructs the polynomial $q(x) = s + \sum_{i=1}^{t-1} \alpha_i x^i$,
3. D sets the share $s_i = \Pi(s, (\alpha_1, \dots, \alpha_{t-1})) = q(i)$,
4. D privately sends to each party P_i their respective share s_i.

For an authorised set of parties $\mathcal{A} = \{P_1, \dots, P_k\}$ (with $k \geq t$) the reconstruction protocol is as follows:

1. Each P_i shares its share s_i with all other parties in $\mathcal{A}$,
2. Each P_i computes $\sum_{j=1}^t s_j \prod_{\ell \neq j} \frac{\ell}{\ell - j} = s$.

Let us now prove the Lagrange interpolation theorem to show that the reconstruction of shares is indeed correct. This requires showing that for any set of t points constructed as in the scheme, then there is a polynomial of degree at most $t-1$ which fits these points, and that this polynomial is in fact unique.

Theorem 3.6. *Given a set of t pairs $\{(i_1, s_{i_1}), \dots, (i_t, s_{i_t})\}$ in a field $\mathbb{F}_q$ with all i_j's distinct, there is a unique polynomial $q(x)$ of degree at most $t-1$ and such that $q(i_j) = s_{i_j}$ for all $j \in [t]$.*

Proof. The existence of such a polynomial $q(x)$ comes directly from the Lagrange interpolation formula, that is, we can construct

$$q(x) = \sum_{k=1}^{t} s_{i_k} \prod_{\ell \neq k} \frac{x - i_\ell}{i_k - i_\ell}.$$

Then $q(i_j)$ has an $i_j - i_j$ term in the product of each element of the sum except at $k = j$, where $s_{i_j} \prod_{\ell \neq j} \frac{i_j - i_\ell}{i_j - i_\ell} = s_{i_j}$, and hence $q(i_j) = s_{i_j}$. Also $q(x)$ has degree at most $t-1$ since the product in each element of the sum takes at most $t-1$ elements.

Now to show the uniqueness of this polynomial, suppose there are two distinct $p(x), q(x)$ both satisfying the required properties. Then let $r(x) = q(x) - p(x)$ be their difference and notice that $r(i_j) = 0$ for all $j \in [t]$. So $r(x)$ has at least t distinct roots, but must also have degree at most $t-1$ since both $p(x)$ and $q(x)$ do. This means $r(x)$ must be the zero polynomial and hence $p(x) = q(x)$. □

It is clear that each party in $\mathcal{A}$ can clearly recover the secret $q(0)$ from t shares. This handles the correctness requirement of the scheme, but what about the privacy. That is, why does a set of at most $t-1$ shares gain no information about the secret? For such a set, the same Theorem 3.6 says that for any secret is as likely to be correct as any other. To see this take any secret $s \in \mathbb{F}_q$ and consider it as an additional share to the set of $t-1$ shares an adversary already has. Then by the theorem there is a unique polynomial $q(x)$ of degree at most $t-1$ fitting the $t-1$ shares and the chosen secret s. Since this holds for any secret, they are each as likely as each other to have been the original secret. Of course if we have no information about the secret with $t-1$ shares then we certainly have no information with less shares, so we can conclude that the secret is recoverable only with t or more shares.

3.4 Linear secret sharing from codes

We now consider how a secret sharing scheme can be modelled as an $[n, k, d]$ linear code C, as was first explained in [23]. Consider the generating matrix of such code, $G = (v_0, \dots, v_n)$, of dimensions $(n+1) \times k$ since the secret is included. Note that the rows form a basis for the code and the codewords in C are all possible linear combinations of the rows. To generate the secret and shares of such a scheme we can take any $c \in \mathbb{F}_q^{n+1}$ and find $cG = (s_0, s_1, \dots, s_n)$. For a description of the sets which are authorised and forbidden we consider the definition of a secret sharing scheme, saying that an authorised set $A \in \Gamma$ is one which can recover the secret s_0 from its set of shares $(s_i)_{i \in A}$. For ease of notation, we simply denote a set of shares $(s_i)_{i \in A}$ as s_A. So a set of parties A is authorised if and only if $v_0 = \sum_{i \in A} \lambda_i v_i$ for some λ_i, from which it is clear that A can recover the secret as $\sum_{i \in A} \lambda_i s_i = s_0$ for any generated secret and shares $(s_0; s_1, \dots, s_n)$. The vectors in the generating matrix therefore define exactly which parties are authorised and which are not, hence the scheme is perfect. Finally we consider the duality of these schemes. The dual scheme can be generated by considering the dual code C^T which is generated by the parity-check matrix H, ie. the $(n-k) \times n$ matrix such that $cH^T = 0$ for all $c \in C$. In this way the dual code is generated as $dH = (x_0, \dots, x_n)$, and the validity of codewords can therefore be checked probabilistically using the dual code, as we will consider in detail in section 6.1.

It can be seen that Shamir's scheme is a particular case of this, where the code is the Reed-Solomon code introduced in section 2.4. That is, for a secret $s \in \mathbb{F}_q$, the code

$$C = \{(p(1), \dots, p(n)) : p(x) \in \mathbb{F}_q[x], \deg(p) \leq k-1\}.$$

describes the set of shares given a polynomial $p(x)$ with $p(0) = s$, as constructed in the scheme. So a (t, n) Shamir threshold scheme can be described by a $[n, t, n-t+1]$ Reed-Solomon code. This code can correct $n-t$ errors and hence as we have seen, a set of t shares is sufficient to recover the secret.

4. Verifiable secret sharing

4.1 Introduction

A secret sharing scheme as described in the previous section is clearly at risk of a corrupt or dishonest dealer. For example the dealer may choose to send shares which when reconstructed by different sets of parties give to each of them different secrets. The dealer could also share arbitrary shares that will reconstruct no secret at all, just a seemingly random element in the domain of secrets. Apart from their intuition and context about what the secret should look like, the parties would have no way of knowing this is the case. The verifiability in verifiable secret sharing schemes is there to overcome such problems. Simply put it requires that any honest and authorised set will reconstruct the same secret and that they can check this is the case. Here we must make a distinction about the honesty of the adversary. In secret sharing schemes an adversary is assumed to be able to see the shares of a forbidden set of parties. Although the privacy property of the scheme means that the adversary has no information about the secret, even after seeing these shares. It is also assumed that the adversary cannot control the parties or the dealer, they simply follow the schemes protocols. These adversaries are generally referred to as **passive** adversaries.

In verifiable secret sharing we allow for **active** adversaries that not only see, but also control a forbidden set of parties. That is, they see their respective shares and choose what messages they send to other parties or to the dealer. Similarly the dealer may be corrupt, meaning that it may send shares that do not reconstruct correctly to a unique secret. For these reasons we are interested in a scheme that would allow honest parties to verify the validity of their shares with regards to reconstructing the correct secret. For this we need the idea of a consistent set of shares.

Definition 4.1. A set of parties $\mathcal{B}$ has a **consistent** set of shares if there is a fixed secret $s' \in S$, not necessarily equal to the dealer's original secret s, such that for any authorised subset $C \subset \mathcal{B}$ the reconstruction function gives $\mathrm{Rec}(\Pi_{\mathcal{B}}(s,r)) = s'$.

Definition 4.2. A secret sharing scheme is **verifiable** if for any authorised set of parties $\mathcal{B}$ there is a protocol $\mathit{Verify}(\Pi_{\mathcal{B}}(s,r))$ which proves to the parties in $\mathcal{B}$ that their shares are consistent.

In short, a verifiable secret sharing scheme must be able to convince its authorised parties that their shares are consistent. We should also adjust the properties of correctness and privacy to account for the case that the dealer is controlled by the adversary. For the correctness, even if the dealer is corrupt we want all honest parties in any authorised set to recover some common secret. Of course in the case of a corrupt dealer we cannot speak of an original secret since the dealer may not have a secret at all and broadcast values at random. So we simply require that their is some fixed secret which they all recover.

Definition 4.3. A verifiable secret sharing scheme has the correctness property if the entire set of parties $\mathcal{P}$ has a consistent set of shares. Hence any party in an authorised set recovers the same secret.

Similarly we alter the security property. In the scenario that the dealer is controlled by the adversary it makes little sense to expect the adversary to not know the secret. So we re-frame the security to require that under an honest dealer, and denoting as $\mathit{View}_{\mathit{Adv}}$ the collection of all the information that the adversary knows, then the adversary does not gain any information on the dealer's secret from $\mathit{View}_{\mathit{Adv}}$.

Definition 4.4. A verifiable secret sharing scheme has the privacy property if given an honest dealer then $\mathit{View}_{\mathit{Adv}}$ is independent of the secret s.

4.2 Background

In 1985 Chor [15] introduced verifiable secret sharing based on Shamir's (t, n) secret sharing scheme and the factorisation problem. Interestingly it allowed the parties to verify any share, even those of other parties. This, we will see, is very close to publicly verifiable secret sharing, although it was not carried forward in the verifiable schemes that followed. We will not go into the details of Chor's scheme because it heavily use ideas such as oblivious transfers and broadcast networks which are not of direct interest in this thesis. Soon after, in 1986 Benaloh [4] considered a different approach to allow for verifiability. This time using homomorphic secret sharing to break shares further into 'sub-share' and allow reconstruction of a 'master-share', but notably without sharing the respective 'sub-secrets'. In 1987 Feldman [16] proposed the first non-interactive verifiable secret sharing scheme, where a party can be convinced of consistency using only their own share and without communicating with any other party. Note that in general it can be allowed for the Verify protocol to require interaction between the parties. We say a verifiable secret sharing scheme is **non-interactive** if a party can run Verify on its own, with no communication with other parties.

4.3 Security types

Like in most areas of study in cryptography there is an important divide between the assumed powers of the models we consider. For secret sharing we can speak of conditionally or unconditionally secure schemes. The latter being what we have so far studied. Also called **information-theoretic security**, it is defined by the fact that even a computationally unbounded adversary with unlimited computation power cannot break the security. We will study in the next section a verifiable secret sharing scheme with this type of security. The weaker **computational security** says that a scheme's security cannot be broken by an adversary whose computational power is bounded in some way. Such schemes usually rely on a computational problem which is known to be hard, and prove their own security by showing that breaking the scheme's security implies being able to solve the problem. The publicly verifiable secret sharing schemes that we will study later on are all of this kind, since they are proposed to be possible to implement in the real world. The unconditional level of security is most often too strong to be considered for such applications since the latter assume that any adversary will be computationally bounded anyway.

4.4 Extended interpolation idea

The main idea in the verifiable secret sharing scheme we will consider in this section is an extension of the Lagrange interpolation method of Shamir, extended to a bivariate polynomial in $\mathbb{F}_q^2$. For this scheme to work we will need to show that given two polynomials $f_i(x)$ and $g_j(y)$ intersecting in the form $f_i(\alpha_j) = g_j(\alpha_i)$ for $\alpha_0, \dots, \alpha_{t+1} \in \mathbb{F}_q$, then they describe a unique bivariate polynomial $F(x, y)$ of degrees (t, t). This will allow the correctness of the scheme to say that what is reconstructed is in fact the same secret for any authorised set of parties. We will also show that given insufficient shares, ie. strictly less than t shares, then we gain no information about this secret. In other words, the shares of a forbidden set of parties have the same distribution for any initial polynomial $p(y)$.

Let us first show that a simple generalisation of the fact that $t + 1$ polynomials of degree t define a unique bivariate polynomial of degrees (t, t). The claim is similar to that of the Lagrange interpolation theorem and hence the proof is similar too.

Lemma 4.5. *For polynomials $f_1(x), \dots, f_{t+1}(x)$ of degree t and values $\alpha_1, \dots, \alpha_{t+1} \in \mathbb{F}_q$, then there is a unique bivariate polynomial $F(x, y)$ of degree (t, t) such that $F(x, \alpha_k) = f_k(x)$ for $k \in [t + 1]$.*

Proof. By Lagrange interpolation let

$$F(x,y) = \sum_{i=1}^{t+1} f_i(x) \prod_{j \neq i} \frac{y-\alpha_j}{\alpha_i - \alpha_j}.$$

For $k \in [t+1]$, every product in the sum of $F(x, \alpha_k)$ has the term $\alpha_k - \alpha_k = 0$, except at $i = k$, hence as required we have that

$$F(x, \alpha_k) = f_k(x) \prod_{j \neq k} \frac{\alpha_k - \alpha_j}{\alpha_k - \alpha_j} = f_k(x).$$

Clearly $F(x,y)$ has degrees (t,t) since $f_i(x)$ has degree t in x and the products have t elements in y.

Now to show it is unique, suppose there are two such bivariate polynomials, $F(x,y)$ and $S(x,y)$ and consider their difference $D(x,y) = F(x,y) - S(x,y)$. Letting $p(y) = \sum_{j=0}^{t} d_{i,j} y^j$ for some $d_{i,j} \in \mathbb{F}_q$ we can write $D(x,y) = \sum_{i=0}^{t} (x^i p(y))$. Given that

$$\begin{aligned} D(x, \alpha_k) &= F(x, \alpha_k) - S(x, \alpha_k) \\ &= f_k(x) - f_k(x) = 0, \end{aligned}$$

then we have that $p(\alpha_k) = 0$ for all $k \in [t+1]$. So $p(y)$ has at least $t+1$ zeros, but is a polynomial of degree at most t by definition, hence it must be the constant zero and its coefficients $d_{i,j}$ are all zero. So $D(x,y)$ is the zero bivariate polynomial and $F(x,y) = S(x,y)$, as required for uniqueness. □

We can now use this lemma to show that the polynomials $f_i(x), g_j(y)$ will define a unique bivariate $F(x,y)$, as required for this extended interpolation idea to work.

Lemma 4.6. *For sets of polynomials $\{f_1(x), \dots, f_k(x)\}$ and $\{g_1(y), \dots, g_k(y)\}$ all of degree t, with $k \geq t+1$, such that $f_i(\alpha_j) = g_j(\alpha_i)$ for $\alpha_i, \alpha_j \in \mathbb{F}_q$ and $i, j \in [k]$, then there is a unique bivariate polynomial $F(x,y)$ of degree (t,t) such that all polynomials $f_i(x)$ and $g_j(y)$ lie on it.*

Proof. For any subset $K \subset [k]$ of size $|K| = t+1$, by Lemma 4.5 we have that there is a unique $F(x,y)$ of degree (t,t) such that

$$F(x, \alpha_i) = f_i(x) \text{ for } i \in K. \tag{1}$$

We want to show that $F(x, \alpha_i) = f_i(x)$ and $F(\alpha_j, y) = g_j(y)$ for all $i, j \in [k]$.

For $i \in K, j \in [k]$ we are given that $f_i(\alpha_j) = g_j(\alpha_i)$ and from equation (1) we have $f_i(\alpha_j) = F(\alpha_j, \alpha_i)$, so $g_j(\alpha_i) = F(\alpha_j, \alpha_i)$. Both $g_j(y)$ and $F(\alpha_j, y)$ are degree t polynomials, so intersecting on the $t+1$ points α_i for $i \in K$ means that they are equal for all $j \in [k]$, as required.

Now to extend $F(x, \alpha_i) = f_i(x)$ from $i \in K$ to $i \in [k]$. Let $i, j \in [k]$, we have $f_i(\alpha_j) = g_j(\alpha_i) = F(\alpha_j, \alpha_i)$. Again both $f_i(x)$ and $F(x, \alpha_i)$ are degree t polynomials so intersecting on the $r \geq t+1$ points α_j means that they are equal for all $i \in [k]$. □

Lastly we consider a lemma which will serve for the privacy of the scheme. Let $\mathcal{C} \subset \mathcal{P}$ be a set of corrupt parties with $|\mathcal{C}| \leq t$. We will abuse notation and write $i \in \mathcal{C}$ to refer to the indexes of these parties. We want to show that regardless of the initial polynomial $p(y)$, the polynomials $f_i(x), g_j(y)$ of the bivariate $F(x,y)$ constructed from it with $F(0,y) = p(y)$ are distributed uniformly. This implies that the shares of the corrupt parties $f_i(0) = F(0, \alpha_i)$ are also uniformly distributed and hence give no information on the original polynomial $p(y)$ or the secret $s = p(0)$.

Lemma 4.7. *For $\mathcal{C} \subset \mathcal{P}$ with $|\mathcal{C}| \leq t$ and polynomials $p(y), q(x)$ of degree t with $p(\alpha_i) = q(\alpha_i)$ for all $i \in \mathcal{C}$, the probability distributions of $\mathcal{F} = (F(x, \alpha_i), F(\alpha_i, y))_{i \in \mathcal{C}}$ and $\mathcal{S} = (S(x, \alpha_i), S(\alpha_i, y))_{i \in \mathcal{C}}$ are identical, where F and S are random degree (t, t) bivariate polynomials with $F(0, y) = p(y)$ and $S(0, y) = q(y)$.*

Proof. First we want to show that for any pair of degree t polynomials $(p(y), q(x))$ there are as many possible pairs $\mathcal{F}$ as pairs $\mathcal{S}$. This shows that for some pair $(f_i(x), g_i(y))_{i \in \mathcal{C}}$ the probability that it is obtained from $\mathcal{F}$ or $\mathcal{S}$ is the same.

Fix some $Z = (f_i(x), g_i(y))_{i \in \mathcal{C}}$. We count the number of possible $\mathcal{F}$'s this Z is valid for. Z has $|\mathcal{C}| \leq t$ degree t polynomials $f_i(x)$ and as we know any set of $t+1$ such polynomials define an F of degrees (t, t). So there are $t + 1 - |\mathcal{C}|$ possible polynomials $f_j(x)$ that are valid for $\mathcal{F}$, that is $f_j(0) = p(\alpha_j)$ and $f_j(\alpha_i) = g_i(\alpha_j)$ for $i \in \mathcal{C}$. By these restrictions, $f_j(x)$ is already defined on $|\mathcal{C}| + 1$ points (α_i with $i \in \mathcal{C}$ and 0). Since it is fully defined by $t + 1$ points, it has $t - |\mathcal{C}|$ points left free, each of which has $|\mathbb{F}_q| = q$ possibilities. Hence there are $(q^{t-|\mathcal{C}|})^{t+1-|\mathcal{C}|}$ possible $f_j(x)$ that are valid for $\mathcal{F}$.

There are exactly the same for $\mathcal{S}$, the counting method is identical. So the probability that Z is obtained from $\mathcal{F}$ or $\mathcal{S}$ is the same, as required. □

Now that we have the required lemmas for the correctness and privacy, we may define the verifiable secret sharing scheme in question.

4.5 A verifiable secret sharing scheme

For the rest of this section we will consider a verifiable secret sharing scheme which extends the ideas of Shamir's scheme. Our example comes from the survey [13] and is a simplified take on the original [3]. The GMW verifiably secret sharing scheme has the requirement that $n > 3t$ where t is the number of corrupt parties. This is necessary for the correctness of the scheme in the case that the dealer is corrupt.

The general idea of the scheme is as follows. For a secret $s \in \mathbb{F}_q$ the dealer randomly picks a degree t polynomial $p(y)$ with $p(0) = s$. It then randomly picks a bivariate polynomial $F(x, y)$ of degrees (t, t) such that $F(0, y) = p(y)$. For a random set of distinct elements $\{\alpha_i : \alpha_i \in \mathbb{F}_q, \alpha_i \neq \alpha_j \text{ for } j \neq i\}_{i \in [n]}$ it gives to each party P_i the polynomials $f_i(x) = F(x, \alpha_i)$ and $g_i(y) = F(\alpha_i, y)$. Here the $f_i(x)$ stands as the share of the secret, whereas the $g_i(y)$ will be used for verifiability. As we will see, a set of at least t parties can then exchange their shares $f_i(0) = F(0, \alpha_i) = p(\alpha_i)$ and hence reconstruct the secret $p(0) = s$.

In order to achieve verifiability in the presence of corrupt parties, the honest parties verify the shares they receive from both the dealer and other parties and can choose to publicly complain about either. For example if party P_i receives $f_j(\alpha_i)$ from P_j and $f_j(\alpha_i) \neq g_i(\alpha_j)$, then at least one of P_i,P_j or the dealer is corrupt, so if P_i is honest then it should broadcast a complaint about P_j. The dealer attempts to resolve these by broadcasting the disputed value, $F(\alpha_i, \alpha_j)$. This is followed by complaints about the dealer in the case that the latter broadcast of the disputed values does not hold with the dealer's original values. These complaints can then be resolved by the dealer who broadcasts the original polynomials of any party who complains about the dealer. Note that, perhaps surprisingly, this does not leak information to the adversary. Since an honest party would not accuse an honest dealer, if the dealer is accused by a party, then one of the two must be corrupt and hence the adversary does not gain any new information.

Once the verifications and accusations are done, and if the protocol is still going with sufficient parties, then they can broadcasts their shares $f_i(0)$ and use a decoding algorithm to retrieve the secret. The algorithm in question is the Berlekamp-Welch decoding algorithm which we will define below. The protocol makes use of the error-correction property of the decoding method such that even if an authorised set has some corrupt parties, the honest ones in the set can still recover the secret.

The idea of the **Berlekamp-Welch algorithm** is to find a monic polynomial $E(x)$ such that

$$p(\alpha_i)E(\alpha_i) = y_i E(\alpha_i) \tag{2}$$

where $\alpha_i \in \mathbb{F}_q$ and $E(\alpha_i) = 0$ only for some r indices i, or 'errors'. This polynomial is often called the error-locator and can be written as $E(x) = \prod_{i:p(\alpha_i)\neq y_i}(x - \alpha_i)$, but it is unknown since the error indices are unknown. To find this polynomial we first define $Q(x) = E(x)p(x)$ and hence we can solve equation (2) as $y_i E(\alpha_i) - Q(\alpha_i) = 0$. Having solved the system of linear equations, we can set $p(x) = Q(x)/E(x)$ and compute the values $p(\alpha_i)$ for each i where $E(\alpha_i) = 0$.

Definition 4.8. For a dealer D with a secret $s \in \mathbb{F}$ and parties $\mathcal{P} = \{P_1, \dots, P_n\}$, the GMW VSS scheme runs as follows.

Distribution:

1. D selects a random degree t polynomial $p(y)$ with $p(0) = s$,
2. D selects a random degree (t, t) polynomial $F(x, y)$ with $F(0, y) = p(y)$,
3. D selects distinct random $\alpha_i \in \mathbb{F}_q$,
4. D sets $f_i(x) = F(x, \alpha_i)$ and $g_i(y) = F(\alpha_i, y)$ and sends both to P_i.

Player complaints:

1. P_i sends $f_i(\alpha_j)$ to P_j for $j \in [n]$,
2. for any P_j for which the received $f_j(\alpha_i) \neq g_i(\alpha_j)$ then P_i broadcasts a complaint about P_j.

First resolutions:

1. for any complaint from P_i about P_j, D broadcasts $F(\alpha_i, \alpha_j)$.

Dealer complaints:

1. P_i broadcasts a complaint of D if:
   - any party has complained about more than t parties or has complained about themself,
   - or if P_j complained about P_i and $F(\alpha_j, \alpha_i) \neq f_i(\alpha_j)$,
   - or if P_i complained about P_j and $F(\alpha_i, \alpha_j) \neq g_i(\alpha_j)$.

Second resolutions:

1. D broadcasts $f_i(x)$ and $g_i(y)$ for any P_i who has complained about D.

Final complaints:

1. P_i broadcasts a complaint of D if for any $j \in [n]$, $f_j(\alpha_i) \neq g_i(\alpha_j)$ or $g_j(\alpha_i) \neq f_i(\alpha_j)$,
2. if more than t parties complain about D then P_i exits the protocol, D is considered corrupt.

Reconstruction:

1. P_i broadcasts $s_i = f_i(0)$,
2. P_i recovers $p(0) = s$ by Berlekamp-Welch decoding with $y_i = s_i$ as the received encrypted messages.

Let us now consider the properties of correctness and privacy of the scheme. For the first we want to show that in the cases of an honest or corrupt dealer, then honest parties in authorised sets do recover the appropriate secret. This is done by showing that the information they have, and which they know holds due to the verifications, is enough to overcome even the maximum number of corrupt parties in their authorised set.

Theorem 4.9. *The GMW VSS scheme is correct. Specifically, if the dealer is honest then an honest party in an authorised set can successfully recover the secret. And given a corrupt dealer, all honest parties in any authorised set recover a common secret.*

Proof. If D is honest then we know that $f_i(\alpha_j) = g_j(\alpha_i)$ for all $i, j \in [n]$. So no honest party complains about the dealer or another honest party. In an authorised set of parties $\mathcal{A}$ there are at most t corrupt parties, and hence the decoding method has at most $r \leq t$ incorrect values. So an honest party in $\mathcal{A}$ does recover the original secret.

Now assume D is corrupt but too few parties complain about it, hence the reconstruction protocol goes ahead. So at least $n - t \geq 2t + 1$ parties do not complain about the dealer. We want to show that the values that the honest parties know all come from a common $F'(x, y)$ of degree (t, t) with $F'(0, 0) = s'$. Let $\mathcal{H}$ be the set of honest parties who do not complain about D. In the worse case, all corrupt parties are in $\mathcal{H}$ so $|\mathcal{H}| \geq n - 2t \geq t + 1$. So by lemma 4.5 there is a unique $F'(x, y)$ of degree (t, t). For an honest party $P_i \notin \mathcal{H}$, then P_i complained about D and hence D broadcast $f_i(x), g_i(y)$. We are done if we can show that these polynomials agree with the common $F'(x, y)$. Firstly we know that $f_i(\alpha_j) = g_j(\alpha_i)$ for $j \in \mathcal{H}$, otherwise P_j would have complained about D. Also $g_j(y) = F'(\alpha_j, y)$ for the same j. Since $|\mathcal{H}| \geq t + 1$ then $\{(\alpha_i, g_j(\alpha_i))\}_{i \in \mathcal{H}}$ uniquely defines $F'(x, \alpha_j)$. So $f_i(x) = F'(x, \alpha_i)$ since they share the set of points $\{(\alpha_i, g_j(\alpha_i))\}_{i \in \mathcal{H}}$. The same argument can be made for $g_i(y)$. $\square$

Now for the security recall for some adversary Adv we denote $View_{Adv}$ the collection of the information that Adv knows. We want to show that the $View_{Adv}$ cannot give any information about the secret s. This will show that the scheme is information-theoretically secure, that is, even a computationally unbounded adversary cannot find any information on the secret with insufficient shares.

Theorem 4.10. *The GMW VSS scheme satisfies privacy. Specifically, $View_{Adv}$ and the secret s are independent if the dealer is honest.*

Proof. Given D is honest then $View_{Adv} = \{f_i(x), g_i(y)\}_{i \in Adv}$, where $i \in Adv$ denotes the indexes of the parties that are controlled by the adversary. Since no honest party complains about the honest dealer, the latter does not broadcast more information than what is known in $View_{Adv}$. This information is derived from the random $F(x, y)$ which itself comes from the random polynomial $p(y)$. Therefore by Lemma 4.6 the distribution of this information is undistinguishable and gives no information on the secret s. $\square$

For completeness we note that this scheme can be improved in terms of efficiency in the following way. As presented above it has 7 protocols, or rounds, which are required to run one after the other, but this can actually be shortened 5 rounds. This adaptation comes from [18] and is discussed in the survey [13]. The idea is to have the parties who complain in the *Player complaints* round to also broadcast their version of the relevant value. So P_i complaining about P_j would also broadcast $F(\alpha_i, \alpha_j)$. Following that, in the

First resolutions round both the dealer and the complained about party, P_j in this case, broadcast their own version of the same value. By comparing the dealer's version to each parties, then it is possible to know which of the two parties would complain about the dealer in the later *Dealer complaints* round. So the dealer and the complaining parties, can share their versions of the required polynomials $f_i(x), g_i(y)$ at the same time. This essentially allows the resolutions of the complaints to be made in the same rounds as the complaints themselves, hence reducing the scheme to 5 rounds.

5. Publicly verifiable secret sharing

5.1 Basics

5.1.1 Publicly verifiable secret sharing model

Extending from the idea that the parties should be able to verify that the shares they receive are compatible with some shared secret, we now consider publicly verifiable secret sharing schemes which is a very popular model of secret sharing to work on. This introduces two important changes to allow the schemes to be implemented in a public setting where the parties do not necessarily trust each other. The first is to extend the verifiability of the shares such that any party can verify any of the shares, even if they are assigned to another party. This is most often formalised by introducing a verifier party which does not take part in the distribution or the reconstruction but is only there to verify shares, and does so only with the public information shared. This highlights the fact that the verification process should be carried out without any private information from the dealer or the parties, since a verifier who has no such information should be capable of checking the shares.

The other important change that the publicly verifiable model introduces is to use some form of encryption for the communication of information. That is, when the dealer sends to each party their respective share, instead of assuming that there is a private communication channel between the dealer and every party, it instead uses a public broadcast which all parties and verifiers can read. Of course in this case the shares must be encrypted such that only the required party recovers the share, otherwise all the parties would immediately have all shares and secret sharing would be futile. We note that this means the verifier must be able to check that the shares are consistent but does not even have access to the shares, only to their encrypted forms. A method which is commonly used to help this is to make the dealer broadcast some sort of proof which helps the verifier do just that. If such a proof of consistency is written into the protocol of the scheme, then an honest dealer will therefore help the verifier, and if a dealer does not produce such a proof then it can be assumed that it is corrupt. The same thing may be done for the parties which share with each other their decrypted shares during reconstruction. That is, all parties in the authorised set must trust that the share sent by each party is valid and it is not a corrupt party sending some other value which will not lead to the secret. Hence the parties can also share a proof of correct decryption and the verifier can validate these.

We can now define a model for publicly verifiable secret sharing schemes, making use of the public key encryption for the parties to communicate privately. However we will not define these by building up from our previous definition of verifiable secret sharing schemes, due to the use of PKE for communication we now need to reconsider the model for the scheme as a whole.

Definition 5.1. A **publicly verifiable secret sharing** scheme for a dealer $\mathcal{D}$ with a secret s and a set of players $\mathcal{P} = \{P_1, \dots, P_n\}$ is made of four protocols and a PKE scheme denoted ε.

- *Setup*: $\mathcal{D}$ and P_i generate private and public key pairs for ε and broadcast any public parameters.
- *Distribution*: $\mathcal{D}$ computes a share for each party from the secret. It encrypts each share using ε and the public key of the respective party. Finally it broadcasts the encrypted shares.
- *Verification*: $\mathcal{D}$ broadcasts a proof that all shares are consistent. Any verifier V can check this.
- *Reconstruction*: P_i recovers its share from the encrypted one using ε. For some authorised set $\mathcal{A}$ and party $P_i \in \mathcal{A}$, P_i sends its share to the parties in $\mathcal{A}$. P_i also sends a proof that the share is the

correct decryption of the encrypted share. A verifier V can check this. Finally P_i can recover the secret using the shares it has received from the other parties in $\mathcal{A}$.

Now as with the previous sharing schemes we require that the PVSS scheme satisfies at least the properties of correctness and privacy. More formally, for correctness we require that for an authorised set of parties where each encrypted share is verified to be consistent and every decryted share is verified to be a valid decryption, then the reconstructed secret is the original secret. Note that this is regardless of the honestly of the parties in the set. Even if corrupt parties are taking part in the reconstruction, if the verifications hold then the reconstruction must hold.

Definition 5.2. The PVSS scheme satisfies **correctness** if for any authorised set of parties $\mathcal{A}$ where both the verification of consistency and valid shares hold for all shares, then the recovered secret is the original secret.

For an adversary Adv controlling a set of parties, let $View_{Adv}$ be all the information that Adv has access to, hence the secret keys and shares of all the parties it controls and those that are public. We could define the privacy requirement just as we did for the secret sharing model earlier, requiring that $View_{Adv}$ be independent of the secret. However it will be more interesting for the coming schemes to instead consider the approach of conditional security. In this case we want that no computationally bounded adversary can correctly recover the secret with probability significantly better than zero. Recall that the union of all the parties under Adv must be a forbidden subset.

Definition 5.3. The PVSS scheme satisfies **privacy** if for a secret s and any computationally bounded Adv controlling a set of parties $\mathcal{A}$, then the probability that the original secret is recovered is negligible.

Of course for publicly verifiable schemes we expect a new property, **public verifiability**. This says that if a corrupt party comes up with a value without following the protocol and sends it to an honest party, then this honest party will known that the value they have received is not valid. Of course this is public since all values are broadcasted publicly and hence anyone can run the verification algorithm. We denote this by saying that a verifier V can run the algorithm, where the verifier can be anyone, not necessarily a party involved in the scheme. The verification requires checking that the encrypted shares can indeed be the encryption of some valid shares for a common secret.

Definition 5.4. The PVSS scheme satisfies public verifiability of distribution if for any subset of parties $\mathcal{B}$ for which the verification holds then the shares are consistent.

Similarly for the verification of the decrypted shares.

Definition 5.5. The PVSS scheme satisfies public verifiability of decryption if for any party P_i for which the verification of the decrypted shares hold, then they are the correct decryption of the shares.

Finally we also consider non-interactive PVSS schemes. We follow the original non-interactive PVSS definition of [17] in requiring that the verifications can be done by a single party on their own, without communicating with other parties. This will be a useful property to consider in the public schemes where it is assumed the parties do not trust each other and the amount of communication between them ought to be minimised. Of course the scheme must still allow for the keys, shares and proofs to be communicated between parties, this is essential.

In this chapter we will study several different approaches to publicly verifiable schemes, coming from the different encryption systems which can be applied. The paper [19] presents a helpful categorisation of schemes depending on their underlying encryption. Before we consider a very well known example of a publicly verifiable secret sharing scheme by Schoenmaker we first introduce the encryption it will use, ElGamal.

5.1.2 ElGamal encryption

The ElGamal public key encryption scheme uses the idea of exponentiation in the exponent to encrypt the message. The security is based on the Diffie-Hellman assumptions, which we present now. For the rest of this section we consider a cyclic group $\mathbb{G}$ of prime order q with a generator $g \in \mathbb{G}$. The computational version of the Diffie-Hellman problem essentially says that it is difficult to compute $g^{\alpha\beta}$ from g^{α}, g^{β}. The stronger decisional version says that it is difficult to differentiate between $g^{\alpha\beta}$ and some random value g^{γ} even given g^{α}, g^{β}. We define these by writing them out in the below diagrams, these will be used throughout the section to illustrate games and protocols between different parties.

Definition 5.6. The **computation Diffie-Hellman** assumption, CDH, says that the following game is infeasible for the adversary, where the adversary wins if $\hat{w} = w$.

$$\begin{array}{ccc}
\mathit{Chal} & & \mathit{Adv} \\
\hline
\alpha, \beta \in_R \mathbb{Z}_q & & \\
w \leftarrow g^{\alpha\beta} & & \\
 & \xrightarrow{(g^{\alpha}, g^{\beta})} & \\
 & & \hat{w} \in \mathbb{G}
\end{array}$$

Definition 5.7. The **decisional Diffie-Hellman** assumption, DDH, says that the following game is infeasible for the adversary, where the adversary wins if $\hat{b} = b$.

$$\begin{array}{ccc}
\mathit{Chal} & & \mathit{Adv} \\
\hline
\alpha, \beta, \gamma \in_R \mathbb{Z}_q & & \\
w_0 \leftarrow g^{\alpha\beta}, w_1 \leftarrow g^{\gamma} & & \\
b \in_R \{0,1\} & & \\
 & \xrightarrow{(g^{\alpha}, g^{\beta}, w_b)} & \\
 & & \hat{b} \in \{0,1\}
\end{array}$$

Definition 5.8. The ElGamal public key encryption scheme has the following three functions.

- *Gen*(): Randomly selects the private key $sk \in \mathbb{Z}_q$ and sets the public key as $pk = g^{sk}$.
- $Enc(m, pk)$: Randomly chooses an ephermeral key $k \in \mathbb{Z}_q$ and sets the shared secret $s = pk^k$. Sets $c_1 = g^k$ and $c_2 = ms$, then the encryption is (c_1, c_2).
- $Dec((c_1, c_2), sk)$: Recovers the shared secret by computing $c_1^{sk} = (g^k)^{sk} = pk^k = s$. Then recovers the message by computing $c_2 s^{-1} = mss^{-1} = m$.

The correctness of the scheme is clear from the decryption step. We do not present a proof of security since the PVSS scheme which will use this encryption scheme will actually use a slightly modified version for efficiency. The proof of security will be given there.

5.1.3 Schoenmaker PVSS scheme

We now present an example of a non-interactive PVSS scheme by Schoenmakers [27]. This scheme has been used as a foundation to build others, of these we will study two [7, 20]. The setup is very similar as we have had previously, a cyclic group $\mathbb{G}$ of prime order q with two generators g and G. Before describing the scheme we will need a small protocol which will be useful for the proofs of knowledge between the parties. Introduced by Chaum and Pederson in [14] the idea is to show an equality between two discrete logarithms.

Definition 5.9. The protocol DLEQ(g_1, h_1, g_2, h_2) shows that $\log_{g_1} h_1 = \log_{g_2} h_2$ for generators $g_1, h_1, g_2, h_2 \in \mathbb{G}$ as follows. Let the prover know some α such that $h_1 = g_1^\alpha$ and $h_2 = g_2^\alpha$. The prover sends a commitment (g_1^w, g_2^w) to the verifier. The verifier sends a challenge c, to which the prover replies with $r = w - \alpha c \mod q$. The verifier can check that $g_1^w = g_1^r h_1^c$ and $g_2^w = g_2^r h_2^c$.

$$\begin{array}{ccc}
\textit{Prover} & & \textit{Verifier} \\
\hline
w \in_R \mathbb{Z}_q & & \\
g_1^w, g_2^w \in_R \mathbb{G} & & \\
 & \xrightarrow{(g_1^w, g_2^w)} & \\
 & & c \in \mathbb{Z}_q \\
 & \xleftarrow{c} & \\
r = w - \alpha c \mod q & & \\
 & \xrightarrow{\text{r}} & \\
 & & \text{check} \begin{cases} g_1^w = g_1^r h_1^c, \\ g_2^w = g_2^r h_2^c. \end{cases}
\end{array}$$

If the checks hold, the verifier is convinced that the prover knows a suitable α since $h_1^c = g_1^{w-r} = g_1^{\alpha c}$, and similarly for g_2. Hence the verifier is convinced that $\log_{g_1} h_1 = \log_{g_2} h_2$. As discussed previously this can be transformed to a non-interactive proof by the Fiat-Shamir method. This is simply done by having the prover hash together h_1, g_1, h_2, g_2 and using the result as the challenge, as we will see in the scheme.

Note that this PVSS scheme does not share a secret chosen by the dealer but instead shares a random secret. Many PVSS schemes are presented like this. We will see afterwards how to transform such a scheme to instead send a specifically chosen secret.

Definition 5.10. The Schoenmaker PVSS scheme runs as follows for a dealer D, parties P_i for $i \in [n]$ and any verifier V.

Setup:

1. P_i generates a private key $sk_i \in \mathbb{Z}_q^*$ and a public key $pk_i = G^{sk_i}$ and broadcasts the latter.

Distribution:

1. D picks random polynomial $p(x) = \sum_{j=0}^{t-1} \alpha_j x^j$ with $s = \alpha_0$.
2. D broadcasts commitments $C_j = g^{\alpha_j}$ for $j \in [t-1]$ and encrypted shares $Y_i = pk_i^{p(i)}$ for $i \in [n]$.
3. Let $X_i = \prod_{j=0}^{t-1} C_j^{i^j}$, then D broadcasts a non-interactive proof for DLEQ(g, X_i, pk_i, Y_i) using the Fiat-Shamir method as follows:
   (a) commitments $a_{1i} = g^{w_i}, a_{2i} = pk_i^{w_i}$ with $w_i \in_R \mathbb{Z}_q$ for $i \in [n]$,
   (b) challenge $c = H(\{X_i, Y_i, a_{1i}, a_{2i}\}_{i\in[n]})$,
   (c) response from the prover $r_i = w_i - p(i)c \mod q$,
   (d) and hence non-interactive proof $(c, \{r_i\}_{i\in[n]})$.

This proof shows that D knows of the unique $p(i)$ such that $X_i = g^{p(i)}$ and $Y_i = pk_i^{p(i)}$.

Verification:

1. V computes $X_i = \prod_{j=0}^{t-1}(C_j)^{i^j}$ using the C_j's.
2. V computes $a_i = g^{r_i} X_i^c$ and $b_i = pk^{r_i} Y_i^c$.
3. V checks that $H(\{X_i, Y_i, a_i, b_i\}_{i\in[n]}) = c$.

Reconstruction:

1. P_i computes the share $S_i = Y_i^{\frac{1}{sk_i}} = G^{p(i)}$ and broadcasts S_i.
2. P_i broadcasts a proof of correct decryption (of Y_i to S_i) using DLEQ(G, pk_i, S_i, Y_i), which proves knowledge of some α such that $pk_i = G^\alpha$ and $Y_i = S_i^\alpha$.
3. By Lagrange interpolation, given S_i for $i \in [t]$ and writing $\lambda_i = \prod_{j\neq i} \frac{j}{j-i}$, then P_i can compute

$$\prod_{i=1}^{t} S_i^{\lambda_i} = \prod_{i=1}^{t} G^{p(i)\lambda_i} = G^{\sum_{i=1}^{t} p(i)\lambda_i} = G^{p(0)} = G^s.$$

Let us now consider the security of the scheme, which we claim holds conditionally under the assumptions of DDH and ROM. In the proof we will need to show that any forbidden set cannot recover any partial information on the secret. This is done by constructing an adversary Adv_{DDH} to the DDH attack game which itself uses an adversary Adv_{PVSS} to the Schoenmaker scheme. By construction, if Adv_{PVSS} can break a decisional version of the scheme then Adv_{DDH} can solve the DDH problem, which is thought to be computationally hard. Note that the decisional version of the scheme is simply getting the adversary to guess between two options which one the secret share is.

Theorem 5.11. *Under DDH and ROM assumptions, no set of $t-1$ parties can gain any information on the secret.*

Proof. We construct an adversary Adv_{DDH} using an adversary Adv_{PVSS} which controls $t-1$ parties. We briefly describe the game as it is given in Figure 5.1.3. Note that in the latter we fix the subscript $i \in [n]$ and $j \in [t-1]$, where the Adv_{PVSS} controls the parties P_j without loss of generality.

Initially Adv_{DDH} receives the DDH problem to distinguish between $g^{\alpha\beta}$ and g^γ. Adv_{DDH} constructs a Schoenmaker PVSS scheme with g^β as the secret and g^α as the generator G, and hence $g^{\alpha\beta}$ as the secret. Adv_{DDH} randomly choosing $p(j)$ fixes the polynomial $p(x)$ since $p(0)$ is also already fixed. Then it forms the X_j, Y_j given the public keys pk_j and by Lagrange interpolation can compute the rest of the X_i and hence the C_j since $X_i = \prod_{j=0}^{t-1}(C_j)^{i^j}$. Finally it sets Y_k such that $Y_k = X_k^{\delta_k} = g^{p(k)\delta_k} = pk_k^{p(k)}$, as required. It sends (C_j, Y_i) to Adv_{PVSS} as in the distribution step of the scheme, although ignoring the proof part. It is known that Adv_{PVSS} would gain no information from the proof since it is a zero-knowledge proof and we are assuming the random oracle model. Adv_{DDH} also sends g^x to Adv_{PVSS} to see if it can distinguish it from the secret share $G^\beta = g^{\alpha\beta}$. The latter guesses $\hat{w} = 1$ if it believes they are the same, and hence $g^x = g^{\alpha\beta}$, and $\hat{w} = 0$ otherwise. So all Adv_{DDH} has to do is guess the same $\hat{b} = \hat{w}$, and it will be correct if Adv_{PVSS}'s original guess was correct. □

As we have noted earlier, the secret share G^s with $s \in \mathbb{Z}_q$ is random. It is easy to instead make this scheme share a specific secret $\sigma \in \Sigma$ with $2 \leq |\Sigma| \leq q$. The distribution step does not change, with some random

$$
\begin{array}{ccccc}
\mathit{DDH} & & \mathit{Adv}_{\mathit{DDH}} & & \mathit{Adv}_{\mathit{PVSS}} \\
\hline
\alpha, \beta, \gamma \in_R \mathbb{Z}_q & & & & \\
b \in_R \{0,1\} & & & & \\
x = \begin{cases} \alpha\beta, & b = 1, \\ \gamma, & b = 0. \end{cases} & & & & \\
 & \xrightarrow{(g^{\alpha}, g^{\beta}, g^{x})} & & & \\
 & & G = g^{\alpha}, C_0 = g^{\beta} & & \\
 & & p(1), \ldots, p(t-1) \in_R \mathbb{Z}_q & & \\
 & & & & sk_j \in \mathbb{Z}_q^*, pk_j = g^{sk_j} \\
 & & & \xleftarrow{(pk_j)} & \\
 & & X_j = g^{p(j)}, Y_j = pk_j^{p(j)} & & \\
 & & \text{(Lagrange) } X_i = g^{p(i)} & & \\
 & & \text{compute } C_j \text{ from } X_i & & \\
 & & Y_k = X_k^{\delta_k} \text{ with } \delta_k \in_R \mathbb{Z}_q & & \\
 & & & \xrightarrow{(C_j, Y_i)} & \\
 & & & \xrightarrow{g^{x}} & \\
 & & & & \hat{w} \in \{0,1\} \\
 & & & \xleftarrow{\hat{w}} & \\
 & & \hat{b} = \hat{w}. & &
\end{array}
$$

Figure 3: The DDH attack using a Schoenmaker PVSS adversary. Recall that we fix the subscripts $i \in [n]$, $j \in [t-1]$ and $k \in [n] \backslash [t-1]$.

$s \in \mathbb{Z}_q$, but the dealer should also broadcast $U = \sigma + H(G^s)$. Each party having recovered G^s can simply compute $U + H(G^s) = \sigma$. Finally note that since we are under the random oracle assumption anyway, the hash of G^s is uniformly random, and hence U does not give any information to an adversary.

We also note that this scheme is an example of a homomorphic secret sharing scheme. That is, given the encrypted shares $Y_i^{(0)}, Y_i^{(1)}$ for secrets $s^{(0)}, s^{(1)}$ respectively, then the product of the encrypted shares $Y_i^{(0)} Y_i^{(1)}$ will recover the sum of the secrets $s^{(0)} + s^{(1)}$. This can be seen from the encryption of the shares which gives

$$Y_i^{(0)} Y_i^{(1)} = pk_i^{p^{(0)}(i)} pk_i^{p^{(1)}(i)} = pk^{p^{(0)}(i)+p^{(1)}(i)},$$

and hence when the Lagrange interpolation finds $G^{p^{(0)}(0)+p^{(1)}(0)} = G^{s^{(0)}+s^{(1)}}$, the sum of the secrets. This will be used in section 8.3 to implement a voting scheme from the Schoenmaker scheme. Briefly, the idea is that given mutliple shares of different votes, the product of these shares is a share for the total of the votes. Hence using the homomorphic property, the talliers are able to recover the total number of votes without having to decrypt each individual vote.

5.2 PVSS with Paillier encryption

5.2.1 Paillier cryptosystem

In this section we will consider the Paillier cryptosystem, introduced in [24], which is used to construct an encryption scheme and a publicly verifiable secret sharing scheme. The system is built around the assumption that computing N-th residue classes is computationally difficult. We will first introduce this problem and then consider how it can be used in a cryptosystem.

For the rest of this section we fix $N = pq$ for p, q large primes. Let us consider the **Euler totient function** $\phi(N)$, defined as the number of integer smaller than N that are coprime with N. In our special case of $N = pq$ we have $\phi(N) = (p-1)(q-1)$. This simply due to the fact that the function is multiplicative and that a prime is coprime with all numbers less than itself, so $\phi(p) = p - 1$. We will also need **Carmichael's function** $\lambda(N)$, which is defined as the smallest positive integer m such that $\alpha^m = 1 \mod N$ for all $\alpha \in [N]$. In our special case we have $\lambda(N) = \text{lcm}(p-1, q-1)$, as shown in the following lemma.

Lemma 5.12. *For p, q primes, then $\lambda(pq) = \text{lcm}(p-1, q-1)$.*

Proof. Clearly $pq = \text{lcm}(p, q)$ so $\lambda(pq) = \lambda(\text{lcm}(p, q))$ and we are done if we can show that $\lambda(\text{lcm}(p, q)) = \text{lcm}(\lambda(p), \lambda(q))$. Both being prime we have $\lambda(p) = p - 1$ and $\lambda(q) = q - 1$ and hence we would have $\lambda(pq) = \text{lcm}(p-1, q-1)$. Let $\lambda' = \lambda(\text{lcm}(p, q))$ and $\ell = \text{lcm}(\lambda(p), \lambda(q))$. By definition $\alpha^{\lambda'} = 1 \mod N$ for all $\alpha \in [\text{lcm}(p, q)]$. Now by the chinese remainder theorem this says that

$$\begin{cases} \alpha^{\lambda'} = 1 \mod p, \\ \alpha^{\lambda'} = 1 \mod q. \end{cases}$$

So $\lambda(p), \lambda(q) | \lambda'$ and hence $\ell | \lambda'$. Similarly the other way, given that $\lambda(p), \lambda(q) | \ell$, then

$$\begin{cases} \alpha^{\ell} = 1 \mod p \text{ for all } \alpha \in [p], \\ \beta^{\ell} = 1 \mod q \text{ for all } \beta \in [q]. \end{cases}$$

Again since $pq = \text{lcm}(p, q)$ then $\gamma^{\ell} = 1 \mod N$ for all $\gamma \in [N]$, and hence $\lambda' | \ell$. So λ' and ℓ divide each other and hence are equal, as required. ☐

From now on we denote $\lambda = \lambda(N)$. Let us now consider a building block of the Paillier system, the N-th residuosity mod N^2, and its associated problem. The originality of the Paillier encryption scheme to be considered later is in the use of the square moduli in the ring $\mathbb{Z}_{N^2}$.

Definition 5.13. A number z is an N-th **residue** mod N^2 if there is a $y \in \mathbb{Z}^*_{N^2}$ such that $z = y^N \mod N^2$.

It is easy to see that the set of N-th residues is a multiplicative subgroup of $\mathbb{Z}^*_{N^2}$. More interestingly it has order $\phi(N)$, where the larger group $\mathbb{Z}^*_{N^2}$ has order $\phi(N^2)$. We denote **CR[N]** the problem of distinguishing an N-th residue from a non N-th residue, that is, is there such a y. And we form the following assumption about the intractability of this problem, which will be useful to prove the semantic security of the encryption scheme.

Conjecture 5.14. *The Decisional Composite Residuosity Assumption,* **DCRA***, says that there is no polynomial time distinguisher for N-th residue mod N^2.*

We consider a function which will later be used as the encryption function, and we note some of its interesting properties. Fixing $g \in \mathbb{Z}^*_{N^2}$ we denote the function $\mathcal{E}_g : \mathbb{Z}_N \times \mathbb{Z}^*_N \to \mathbb{Z}^*_{N^2}$ which maps $(x, y) \to g^x y^N \mod N^2$. To verify this function is well-defined we consider a different representation of y given by $y' = y + kN$ for $k \in [N-1]$. This y' with the same x is mapped to

$$\begin{aligned}\mathcal{E}_g(x, y') &= g^x (y + kN)^N \\ &= g^x y^N + \sum_{i=1}^{N} \binom{N}{i} (kN)^i y^{N-i} \\ &= g^x y^N \mod N^2,\end{aligned}$$

since each element in the sum has an N^2 term. It is easy to see that $\mathcal{E}_g$ is homomorphic since

$$\mathcal{E}_g(x_1, y_1)\mathcal{E}_g(x_2, y_2) = g^{x_1+x_2}(y_1 y_2)^N = \mathcal{E}_g(x_1 + x_2, y_1 y_2) \mod N^2.$$

The property of being homomorphic will be important later on when we use the $\mathcal{E}_g$ to encrypt messages. In order to define the residuosity classes we require another property of the function. That is, $\mathcal{E}_g$ is bijective for an appropriate g. First denote $\mathcal{B}_\alpha \subset \mathbb{Z}^*_{N^2}$ the set of elements of order αN, and denote $\mathcal{B}$ their union for $\alpha = 1, \dots, \lambda$.

Lemma 5.15. *If the order of g is a non-zero multiple of N, then $\mathcal{E}_g$ is an isomorphism.*

Proof. We already know that $\mathcal{E}_g$ is homomorphic so we only need to show it is also bijective. Since $\mathbb{Z}_N \times \mathbb{Z}^*_N$ and $\mathbb{Z}^*_{N^2}$ have the same cardinality of $N\phi(N)$, all that is required is to show that the kernel is the identity. So consider some $x \in \mathbb{Z}_N, y \in \mathbb{Z}^*_N$ for which

$$\mathcal{E}_g(x, y) = g^x y^N = 1 \mod N^2. \tag{3}$$

By definition of λ as the smallest positive m such that $\alpha^m = 1 \mod N^2$ for all $\alpha \in [N]$, then $(y^N)^\lambda = (Y^\lambda)^N = 1 \mod N^2$. So $\mathcal{E}_g(x, y)^\lambda = g^{x\lambda} = 1 \mod N^2$ and λx must be a multiple of g's order and hence a multiple of N. Since $gcd(\lambda, N) = gcd(\text{lcm}(p-1, q-1), pq) = 1$ then x is a multiple of N. So $x = 0 \mod N$ and equation (3) gives $y^N = 1 \mod N^2$, so $y = 1$ over $\mathbb{Z}^*_N$. So we find the kernel is the identity element $(0, 1) \in \mathbb{Z}_N \times \mathbb{Z}^*_N$. □

Now that we know that $\mathcal{E}_g$ is bijective for $g \in \mathcal{B}$, we can define the residuosity class for an element in the image of $\mathcal{E}_g$.

Definition 5.16. For $g \in \mathcal{B}$ and $w \in \mathbb{Z}_{N^2}^*$, the N-th **residuosity class** of w with respect to g, denoted $[\![w]\!]_g$, is the unique $x \in \mathbb{Z}_N$ such that there is a $y \in \mathbb{Z}_N^*$ with $\mathcal{E}_g(x, y) = w$.

Naturally we can denote the inverse of this function as the class function $C_g : (\mathbb{Z}_{N^2}^*, \times) \to (\mathbb{Z}_N, +)$ mapping $C_g(w) = [\![w]\!]_g$. An important property of this class function is that it is homomorphic for any $g \in \mathcal{B}$. This follows from the fact that the function $\mathcal{E}_g$ is an isomorphism and $C_g(w)$ is equivalent to taking the element x from $\mathcal{E}_g^{-1}(w) = (x, y)$. The class function being homomorphic implies that the Paillier encryption scheme is also homomorphic, since the encryption function is $\mathcal{E}_g$. We will later see a PVSS scheme which takes advantage of this encryption property. Finally we define a simple function which will be used for the decryption of the messages in the scheme. Let $S_N = \{u < N^2 : u = 1 \mod N\}$ be the multiplicative subgroup of integers mod N^2 for which the function $L(u) = \frac{u-1}{N}$ is well defined.

5.2.2 Paillier encryption

We may now describe the Paillier encryption scheme which simply makes use of the $\mathcal{E}_g$ function as the encryption function. Note that a message being sent is assumed to be in the form of an integer of a predetermined small size.

Definition 5.17. The Paillier encryption scheme for a sender A and receiver B runs as follows.

Setup:

- B fixes $N = pq$ and a random base $g \in \mathcal{B} \subset \mathbb{Z}_{N^2}^*$.
- B broadcasts the public parameters N and g.
- B computes the private parameter $\lambda = \text{lcm}(p-1, q-1)$.

Encryption:

- For a message $m < N$, A randomly chooses an $r < N$.
- A computes the ciphertext $c = \mathcal{E}_g(m, r) = g^m r^N \mod N^2$.
- A broadcasts c.

Decryption:

- B having received the ciphertext $c < N^2$, computes the message $m' = \frac{L(c^\lambda \mod N^2)}{L(g^\lambda \mod N^2)} \mod N$.

As stated earlier it is straight-forward to see that the encryption is homomorphic. For messages $m_1, m_2 < N$ and random $r_1, r_2 < N$ then

$$\begin{aligned}\mathcal{E}_g(m_1, r_1)\mathcal{E}_g(m_2, t_1) &= g^{m_1} r_1^N g^{m_2} r_1^N \\ &= g^{m_1+m_2}(r_1 r_2)^N \\ &= \mathcal{E}_g(m_1 + m_2, r_1 r_2).\end{aligned}$$

Now to prove the correctness of the scheme we want to show that the decrypted message m' is in fact the original message m.

Theorem 5.18. *The paillier encryption scheme is correct, hence $m' = m$.*

Proof. First we want to show that for any $w \in \mathbb{Z}_{N^2}^*$ then

$$L(w^\lambda \mod N^2) = \lambda [\![w]\!]_{n+1} \mod N.$$

Since $(N+1)^N = \sum_{k=0}^{N} \binom{N}{k} N^k = 1 \mod N^2$ then $N+1$ has order N and is in $\mathcal{B}$. This means that $\mathcal{E}_{N+1}$ is surjective and hence there is some $(a,b) \in \mathbb{Z}_N \times \mathbb{Z}_N^*$ such that $\mathcal{E}_{N+1}(a,b) = (N+1)^a b^N = w \mod N^2$. By definition this says that $a = [\![w]\!]_{N+1}$. So we can write

$$\begin{aligned} w^\lambda &= (N+1)^{a\lambda} b^{N\lambda} \\ &= (N+1)^{a\lambda} \\ &= 1 + aN\lambda \mod N^2. \end{aligned}$$

Where the second line comes from the definition of λ as the smallest m such that $\alpha^m = 1 \mod N^2$ for all $\alpha < N$, and the last equality comes from the fact that all the terms with $N^{a\lambda}$ are 0 since, $\lambda = \text{lcm}(p-1, q-1)$ is even and hence $N^{a\lambda} = (N^2)^{a\lambda/2} = 0 \mod N^2$. So we have the required,

$$\begin{aligned} L(w^\lambda \mod N^2) &= L(1 + aN\lambda \mod N^2) \\ &= a\lambda \\ &= \lambda [\![w]\!]_{N+1} \mod N. \end{aligned}$$

With this equation we can write the received message m' as follows

$$m' = \frac{L(c^\lambda \mod N^2)}{L(g^\lambda \mod N^2)} = \frac{[\![c]\!]_{N+1}}{[\![g]\!]_{N+1}} \mod M.$$

Finally we use the equation $[\![w]\!]_{g_1} = [\![w]\!]_{g_2} [\![g_2]\!]_{g_1} \mod N$. This can be seen to hold if we let $\mathcal{E}_{g_2}(x_2, y_2) = w$ and $\mathcal{E}_{g_1}(x_1, y_1) = g_2$ and check that $\mathcal{E}_{g_1}(x_2 x_3, y_2 y_3^{x_2}) = w$. So we have $\frac{[\![c]\!]_{N+1}}{[\![g]\!]_{N+1}} = [\![c]\!]_g \mod N$. This means that the decrypted message is the unique $m' \in \mathbb{Z}_N$ such that for some $r \in \mathbb{Z}_N^*$ then $\mathcal{E}_g(m', r) = c$. □

The semantic security is quite straight-forward given how the scheme is built around the residuosity class.

Theorem 5.19. *The Paillier PKE scheme is semantically secure under the decisional composite residuosity assumption.*

Proof. Given an adversary Adv_{PKE} which can break the semantic security of the encryption scheme, we construct an adversary Adv_{DCR} which can solve the CR[N] problem. We briefly explain how the attack game in Figure 4 works. Adv_{DCR} receives a number z and needs to decide if it is an N-th residue or not. It also receives two messages m_0, m_1 from Adv_{PKE} from which it chooses one at random m_w and sets $c = g^{m_w} z$. Adv_{PKE} replies to the encryption with $\hat{w}$ and finally Adv_{DCR} guesses that z is an N-th residue if and only if $\hat{w} = w$.

We consider the cases $b = 0, 1$ to compute the probability that Adv_{DCR} succeeds. First if $b = 1$ then z is an N-th residue and there is some $y < N^2$ such that $z = y^N$. So the encryption is $c = g^{m_w} y^N$ which is of the form of $\mathcal{E}_g(m_b, y)$ for which we assume Adv_{PKE} can guess $\hat{w} = w$ correctly. Now if $b = 0$ then z cannot be written as some y^N and hence the encryption c does not have the form of some encryption $\mathcal{E}_g(m_b, y)$ and hence Adv_{PKE}'s guess is no better than random. So $\mathbb{P}(\hat{b} = b) = \mathbb{P}(\hat{w} = w) = 1/2$. Therefore for $b \in_R \{0, 1\}$ we have $\mathbb{P}(\hat{b} = b) = 3/4$, which is a non-negligable probability of solving the DCR problem, as required. □

DCR		Adv_{DCR}		Adv_{PKE}
parameters N, g				
$b \in_R \{0,1\}$				
$z \in_R \mathbb{N}$ — z-reducible iff $b = 1$				
	$\xrightarrow{(N,g,z)}$			
			$\xrightarrow{(N,g)}$	
				$m_0, m_1 \in \mathbb{Z}_N$
			$\xleftarrow{(m_0,m_1)}$	
		$w \in_R \{0,1\}$		
		$c = g^{m_w} z$		
			$\xrightarrow{c}$	
				$\hat{w} \in \{0,1\}$
			$\xleftarrow{\hat{w}}$	
		$\hat{b} = \begin{cases} 1 & \hat{w} = w, \\ 0 & \text{otherwise.} \end{cases}$		

Figure 4: The DCR attack using a Paillier PKE adversary.

5.2.3 PVSS scheme with Paillier encryption

We can now introduce the Ruiz and Villar [26] scheme, which we will denote RV, as an example of a PVSS scheme which makes use of the Paillier encryption scheme. Its privacy is based on the Decisional Composite Residuosity Assumption, and as we will see, the verification process is unconditionally secure. An interesting idea presented in this scheme reverses the usual method of each party having a secret key for which the dealer has a respective public key. Instead the dealer has a single private key and all parties have the respective public key. Of course the parties also send some other value to the dealer at setup so that the dealer can encrypt a message which only a specific party can later decipher. Lastly we also note that this PVSS scheme is non-interactive but without the use of Fiat-Shamir heuristics or other ad-hoc methods. So the verification process can naturally be carried out by any party without the need for communication.

In their paper Ruiz and Villar first present a scheme and then consider its security, where the adversaries are assumed to be passive. They say that with some modifications the proposed scheme is also secure against active adversaries. The modifications this involves are minor differences in the way that parties broadcast and verify values. Here we will directly consider this modified scheme. Let us now define the scheme, where the general setup is very similar to that of Paillier encryption.

Definition 5.20. The RV PVSS scheme works as follows for a dealer D with a secret s and parties P_i for $i \in [n]$.

Setup:

1. D selects large primes p, q and sets $N = pq$,
2. D selects $g \in_R \mathbb{Z}^*_{N^2}$ of order N,
3. D broadcasts (N, g),
4. P_i selects $(m_i, r_i) \in_R \mathbb{Z}_N \times \mathbb{Z}^*_N$,

5. P_i sets $c_i = g^{m_i} r_i^N \mod N^2$ and broadcasts it,
6. D checks that $c_i \in \mathbb{Z}_{N^2}^*$ and kicks out any party for which it is not, or which does not send a c_i.

Distribution:

1. D uses Paillier decryption to recover $(m_j, r_j) \leftarrow Dec(c_j)$ for $i \in [n]$,
2. D sets $a_0 = s$ and selects $a_j \in_R \mathbb{Z}_N$ for $j \in [t-1]$,
3. D sets the polynomial $p(x) = \sum_{j=0}^{t-1} a_j x^j$,
4. D sets the shares $s_i = p(i) \mod N$ for $i \in [n]$,
5. D sets $d_i = s_i + m_i \mod N$ and broadcasts d_i.

Verification:

1. D selects $r'_j \in_R \mathbb{Z}_N^*$ for $j \in [t-1]$,
2. D sets $A_j = g^{a_j} r_j'^N \mod N^2$,
3. D sets $t_i = r_i \prod_{k=0}^{t-1} r_k'^{i^k} \mod N$ for $i \in [n]$,
4. D broadcasts (A_j, t_i) for $j \in [t-1], i \in [n]$,
5. V can verify that for $\ell \in [n]$,

$$\prod_{j=0}^{t-1} A_j^{\ell^j} = \frac{g^{d_\ell} t_\ell^N}{c_\ell} \mod N^2, \tag{4}$$

Reconstruction: for parties $A = \{P_0, \ldots, P_{t-1}\}$,

1. P_i broadcasts (m_i, r_i),
2. V can check $c_\ell = g^{m_\ell} r_\ell^N \mod N^2$ for $\ell \in [n]$,
3. P_i recovers the share $s_i = d_i - m_i$,
4. P_i recovers the secret by Lagrange interpolation $s = a_0 = \sum_{i \in A} \prod_{h \neq i} \frac{h}{h-i} s_i$.

Let us first consider how this recovery works and show that the scheme is correct. The important verification step is equation (4), carried out by V in the verification protocol. First we check that this equation should in fact hold.

Lemma 5.21. *With respect to the constructions in definition 5.20, the verification holds if the commits are valid, ie. if $A_j = g^{a_j} r_j'^N \mod N^2$, then $\prod_{\ell=0}^{t-1} A_\ell^{j^\ell} = \frac{g^{d_j} t_j^N}{c_j} \mod N^2$.*

Proof. Given $A_j = g^{a_j} r_j'^N \mod N^2$, then we have

$$\begin{aligned}\prod_{\ell=0}^{t-1} A_\ell^{i^\ell} &= \prod_{\ell=0}^{t-1} (g^{a_\ell} r_\ell'^N)^{i^\ell} \\ &= g^{\sum_{\ell=0}^{t-1} a_\ell i^\ell} (\prod_{\ell=0}^{t-1} r_\ell'^{i^\ell})^N \mod N^2.\end{aligned}$$

Using $s_i = \sum_{\ell=0}^{t-1} a_\ell i^\ell$ and $t_i = r_i \prod_{\ell=0}^{t-1} r_\ell'^{i^\ell} \mod N$ then

$$\prod_{\ell=0}^{t-1} A_\ell^{i^\ell} = g^{s_i}(\frac{t_i}{r_i})^N \mod N.$$

Finally multiplying by $\frac{g^{m_i} r_i^N}{c_i} = 1$ and using $d_i = m_i + s_i$ then as required

$$\prod_{\ell=0}^{t-1} A_\ell^{i^\ell} = \frac{g^{d_i} t_i^N}{c_i} \mod N^2.$$

□

Theorem 5.22. *The RV PVSS scheme is correct. Equivalently, if all verifications hold and there is a sufficient set of parties A for recovery, then each party in A recovers the original secret s.*

Proof. Assuming that all verifications hold, most importantly equation (4), then we consider the decryption of both sides of the equation. By the homomorphism of the Paillier encryption scheme the decryption of the left-hand side is

$$\begin{aligned} Dec\left(\prod_{\ell=0}^{t-1} A_\ell^{i^\ell}\right) &= \sum_{\ell=0}^{t-1} Dec(A_\ell)^{j^\ell} \\ &= \sum_{\ell=0}^{t-1} a_\ell^{j^\ell} \mod N. \end{aligned}$$

For the right-hand side, using $c_i = g^{m_i} r_i^N$ and noticing the correct form of the message to decrypt then

$$\begin{aligned} Dec\left(\frac{g^{d_i}}{c_i} t_i^N\right) &= Dec\left(g^{d_i - m_i}\left(\frac{t_i}{r_i}\right)^N\right) \\ &= d_i - m_i \\ &= s_i \mod N. \end{aligned}$$

So $s_i = \sum_{\ell=0}^{t-1} a_\ell^{i^\ell} \mod N$. Hence all the recovering parties have the same polynomial $p(x)$ and recover the same $a_0 = s$. □

As mentioned earlier, here we consider a version of the scheme which allows for active adversaries. The proof of security given in the paper [26] considers passive adversaries. The following proof is an adaptation of the latter for the active case. Both proofs rely on the Decisional Composite Residuosity Assumption, so the scheme does not require a stronger assumption for the stronger security. Both have a similar layout with an adversary Adv_{DCR} to the DCR problem which uses as a black box an adversary Adv_{PVSS} to the security of the PVSS scheme. The main difference is that with a passive adversary to the PVSS scheme, all parties follow the protocol honestly and hence Adv_{PVSS} does not provide any values. That is, all values are constructed by Adv_{DCR} with the only requirement that the verifications hold. In the active adversary case, Adv_{DCR} must allow for the corrupt parties to input their own values, since they may not follow the protocol honestly. So Adv_{DCR} must choose the values for the honest parties in such a way that they can be verified alongside the corrupt ones.

Definition 5.23. A PVSS scheme is semantically secure against an active adversary if for any active polynomial time adversary Adv controlling $t-1$ parties the distributions $(View_{Adv}, s)$ and $(View_{Adv}, s')$ are indistinguishable, where s is a fixed secret and s' a distinct random secret.

Theorem 5.24. *The RV PVSS scheme is semantically secure against an active adversary under the DCR assumption.*

Proof. Given an adversary Adv_{PVSS} to the scheme that can guess whether or not a given secret is the shared secret, we build an adversary Adv_{DCR} to the DCR problem. Hence if the scheme can be broken then so can the DCR problem, a contradiction to our assumption. For the rest of the proof, unless stated otherwise, we assume the following notation $i \in [n], j \in [t-1]$ and $k \in [n]\backslash[t-1]$.

We first describe how Adv_{DCR} transforms the DCR challenge into a valid PVSS challenge for Adv_{PVSS}. Adv_{DCR} starts by receiving the public parameters which it forwards to Adv_{PVSS}. It also receives z_b, where the *DCR* problem is to guess whether z_b is of the form ρ^N or $g^\mu \rho^N$. In return it receives $(c_j)_{j\in[t-1]}$ from Adv_{PVSS}. Note that these may or may not be in the correct form of $c_j = g^{m_j} r_j^N \mod N^2$, since the adversary is free to send whatever they want. Adv_{DCR} does not know if they are valid but continues the protocol honestly and deciphers the c_j's to recover (m_j, r_j). Next it randomly selects share $(s_j)_{j\in[t-1]}$ and encrypts them as $d_j = s_j + m_j$. The other shares $(s_k)_{k\in[n]\backslash[t-1]}$ are also random but do not need to be chosen specifically, it suffices to randomly choose d_k. These encrypted shares are broadcasted $(d_i)_{i\in[n]}$.

Next Adv_{DCR} must create the values $(A_j)_{j\in[t-1]}$ and $(t_i)_{i\in[n]}$, which it does working backwards from the s_i, r_j and z_b. First it randomly selects a secret s_0 and sets $A_0 = z_b g^{s_0}$, where we note that

$$A_0 = z_b g^{s_0} = \begin{cases} g^{s_0}\rho^N & \text{if } b = 1, \\ g^{s_0+\mu}\rho^N & \text{if } b = 0. \end{cases} \tag{5}$$

So it is of the correct form for a secret s_0 or $s_0 + \mu$ respectively. Next Adv_{DCR} sets $(a_j)_{j\in[t-1]}$ as the unique solutions to the equation $s_k = \sum_{\ell=0}^{t-1} a_\ell k^\ell$, and hence each one can be written as $a_j = \sum_{\ell=0}^{t-1} \nu_{\ell j} s_\ell$ for some ν_{kj}. Finally Adv_{DCR} can set A_j and t_i as shown in Figure 5 for $j \in [t-1], i \in [n]$. It sends both sets to Adv_{PVSS} along with s_0, to which it receives the reply b'. Finally Adv_{DCR} sets its guess as b'.

We now consider the probability with which Adv_{PVSS} will be able to correctly guess b'. If it correctly guesses b' then by equation (5) this is also a correct guess for the *DCR* problem. That is, if $b' = 1$ then $A_0 = g^{s_0}\rho^N$, so $z_b = \rho^N$ and hence $b = 1$, similarly for $b' = 0$. This correct guess relies on whether or not the values Adv_{PVSS} receives are in the correct form for a fixed secret s_0. It is clear that the d_i it receives are, since for $i \in [t-1]$ Adv_{DCR} follows the protocol honestly. More interesting is the verification in equation (4), which we can see holds as follows. By construction of A_j for $j \in [t-1]$,

$$A_j = A_0^{\nu_{0j}} \prod_{k=1}^{t-1} (g^{s_k} r_k^N)^{\nu_{kj}}.$$

Using $A_0 = z_b g^{s_0} = g^s \rho^N$ for some s, then

$$A_j = g^{s\nu_{0j} + \sum_{k=1}^{t-1} s_k \nu_{kj}} \left(\rho \prod_{k=1}^{t-1} r_k^{\nu_{kj}} \right)^N . \tag{6}$$

The critical point here is that if $b = 1$ then $z_b = g^{s_0}\rho^N$ and hence $r'_j = z_b(\prod_{k=0}^{t-1} r_k^{\nu_{kj}})^N = (\rho \prod_{k=0}^{t-1} r_k^{\nu_{kj}})^N$. In which case we can use this and $a_j = \sum_{k=0}^{t-1} \nu_{kj} s_k$ in equation (6) to find $A_j = g^{a_j} r'_j$. From which Lemma

5.21 says that equation (4) does indeed hold. So the values sent to Adv_{PVSS} are in the correct form and hence $\mathbb{P}(b' = 1|b = 1) > 1/2 + \varepsilon$. In the case that $b = 0$, then the above argument does not hold since $z_b = g^{s_0+\mu}\rho^N$ and hence A_j has an additional term of g^{μ}. In this case Adv_{DCR} keeps the guess $b' = 0$ with $\mathbb{P}(b' = 0|b = 0) = 1/2$. In total, $\mathbb{P}(b' = b) = \frac{1}{2}\mathbb{P}(b' = 0|b = 0) + \frac{1}{2}\mathbb{P}(b' = 1|b = 1) > 1/2 + \varepsilon$. So as required, a successful adversary to the DCR problem can be constructed from a successful adversary to the PVSS semantic security. $\square$

DCR		Adv_{DCR}		Adv_{PVSS}
parameters N, g				
$b \in_R \{0,1\}, \rho \in_R \mathbb{Z}_N^*$				
$\mu = 0$ if $b = 1$ and $\mu \in_R \mathbb{Z}_N$ otherwise				
$z_b = g^{\mu}\rho^N \mod N$				
	$\xrightarrow{(g, N, z_b)}$			
			$\xrightarrow{(N, g)}$	
				$c_j \in \mathbb{Z}_N^2$
			$\xleftarrow{(c_j)}$	
		$(m_j, r_j) \leftarrow Dec(c_j)$		
		$s_j \in_R \mathbb{Z}_N$		
		$d_j = s_j + m_j \mod N$		
		$d_k \in_R \mathbb{Z}_N$		
			$\xrightarrow{(d_i)}$	
		$s_0 \in_R \mathbb{Z}_N$		
		$A_0 = z_b g^{s_0}$		
		a_j s.t. $s_k = \sum_{j=0}^{t-1} a_j k^j$		
		ν_{kj} s.t. $a_j = \sum_{k=0}^{t-1} \nu_{kj} s_k$		
		$A_j = A_0^{\nu_{0j}} \prod_{k=1}^{t-1} (g^{s_k} r_k^N)^{\nu_{kj}}$		
		$r'_j = z_b(\prod_{k=0}^{t-1} r_k^{\nu_{kj}})^N$		
		$t_i = r_i \prod_{j=0}^{t-1} r_j'^{ij}$		
			$\xrightarrow{(A_j, t_i, s_0)}$	
				$b' \in \{0,1\}$
			$\xleftarrow{b'}$	
	$\xleftarrow{b'}$			

Figure 5: The DCR attack using a PVSS adversary. Recall that $i \in [n], j \in [t-1]$ and $k \in [n]\backslash[t-1]$.

5.3 PVSS with bilinear maps

The scheme by Schoenmakers that we saw in section 5.1.3 can be adapted to use bilinear maps as encryption instead of ElGamal. This was proposed by [20] who noted that one of the main advantages of this changes was to remove the need for zero-knowledge proofs for verification. As we will see, verifications can be carried out by sending commitments and checking equations which use bilinear maps. This method boasts unconditional public verifiability. This means that Villar's adaptation of the PVSS scheme does not need the commitment-challenge-response communication of zero-knowledge proofs, or the assumption of the random oracle model to use the Fiat-Shamir heuristic as was shown above.

In the paper [20] a few small adaptations are proposed which are of interest and we will consider here. The first is to have secret reconstruction over a public channel. That is, once the parties of an authorised set agree to recover the secret together, they communicate their shares and verifications as encrypted messages over the channel to which everyone has access. This is instead of assuming secure private channels between any set of parties, or allowing the reconstruction to be followed publicly by anyone. There are a few things to note here. Firstly, this is something that can be done for any PVSS scheme, since it simply requires encrypting the messages that the parties send to each other. Since it is done using bilinear maps, it is in a way more efficient to do that using for example ElGamal and zero-knowledge proofs for validity. Here we consider this small adaptation out of general interest and because it may be a useful consideration when using such a scheme in the real world.

The other small adaptation is with regards to the adversary model. Here we will consider an **adaptive** adversary which can corrupt parties at any time during the scheme, until it reaches at most $t-1$ parties. Note that once a party is corrupted it cannot then become honest. This adversary model is stronger than what we have so far seen, referred to as a **static** adversary.

5.3.1 Bilinear maps

For security the scheme relies on the Decisional Bilinear Diffie-Hellman, **DBDH**, assumption which we consider now. The main idea behind bilinear maps is as follows. For the rest of this section we fix G and G_1 as groups of order q a large prime, and g a generator of G. For written simplicity we use the additive notation for G and the multiplicative one for G_1.

Definition 5.25. The map $e : G \times G \to G_1$ is bilinear if it satisfies:

- bilinearity: $e(aP, bQ) = e(P,Q)^{ab}$ for $P, Q \in G$ and $a, b \in \mathbb{F}_q$,
- non-degeneracy: not all pairs P, Q are mapped to the identity in G_1,
- computability: there is an efficient algorithm to compute $e(P,Q)$ for $P, Q \in G$.

As the name suggests, the DBDH assumption is the Diffie-Hellman assumption adapted to a bilinear map.

Definition 5.26. The Decisional Bilinear Diffie-Hellman assumption says that there is no polynomial time algorithm to distinguish between $e(g,g)^{abc}$ and $e(g,g)^{d}$, where g, g^a, g^b, g^c are publicly known and a, b, c, d are randomly selected from $\mathbb{F}_q^*$.

The PVSS scheme we will consider in this section will use a special case of the DBDH problem where $a = b$.

Definition 5.27. The Decisional Bilinear Square, **DBS**, assumption says that there is no polynomial time algorithm to distinguish between $e(h,h)^c$ and $e(h,h)^d$, where $g, g^c, h = g^a$ are publicly known and a, c, d

are randomly selected from $\mathbb{F}_q^*$.

5.3.2 PVSS scheme with bilinear maps

We stick with the same groups G, G_1 and the bilinear map $e : G \times G \to G_1$, and consider two dinstinct generators g, h of group G. Let the dealer D want to share a secret $S = e(h,h)^{z_0}$ with z_0 randomly selected in $\mathbb{F}_q^*$, as before this can be transformed into a scheme sharing a chosen secret.

Definition 5.28. The Villar PVSS scheme runs as follows for D to share $S = e(h,h)^{z_0}$ to parties P_i for $i \in [n]$, with any verifier V.

Setup:

1. P_i randomly chooses a secret key $d_i \in_R \mathbb{F}_q^*$,
2. P_i sets its public key $h_i = h^{d_i}$ and broadcasts it.

Distribution:

1. D randomly selects a polynomial $p(x) = \sum_{j=0}^{t-1} \alpha_j x^j$ with $\alpha_j \in \mathbb{F}_q$ and $\alpha_0 = z_0$,
2. D sets the commitments $C_j = g^{\alpha_j}$ and broadcasts them,
3. D encrypts the shares $Y_i = h_i^{p(i)}$ and broadcasts them.

Verification:

1. V sets $X_i = \prod_{j=0}^{t-1} C_j^{i^j}$,
2. V checks that $e(X_i, h_i) = e(g, Y_i)$ holds for all $i \in [n]$.

Reconstruction: for an authorised set A with $P_i \in A$,

1. P_i recovers the share $S_i = Y_i^{1/d_i}$,
2. P_i randomly selects $\rho \in \mathbb{F}_q$,
3. P_i sets $r = h_i^{\rho}, z = Y_i^{\rho}$ and $w = h_j^{\frac{1}{d_i \rho}}$ for $j \in A \backslash \{i\}$,
4. P_i broadcasts (r, z, w).
5. V checks that $e(r, Y_i) = e(z, h_i)$ and $e(r, w) = e(h_j, h)$,
6. P_i, having received (r', z', w') from some $P_j \in A$, computes the share $e(h,h)^{p(j)} = e(z', w')^{1/d_i}$,
7. P_i can recover the secret with the shares of all $P_j \in A$ as

$$\prod_{i \in A} e(z, w)^{\lambda_i / d_i} = \prod_{i \in A} e(h,h)^{p(i)\lambda_i} = e(h,h)^{\sum_{i \in A} p(i)\lambda_i} = e(h,h)^{p(0)},$$

where $\lambda_i = \prod_{j \neq i} \frac{i}{j-i}$ and $p(0) = z_0$.

The correctness of this scheme is clear from the reconstruction step, of course assuming that the verifications hold. We now consider the verifiability of the information broadcast by the dealer during the distribution and by the players during the reconstruction.

Theorem 5.29. *If the verification of the dealer's share pass, then there is a unique polynomial $p(x)$ such that $Y_i = h_i^{p(i)}$ for all $i \in [n]$.*

Proof. If the verification passes then $e(X_i, h_i) = e(g, Y_i)$ for all $i \in [n]$. By construction of X_i and C_j and the bilinearity of e, the left-hand side is $e(X_i, h_i) = e(\prod_{j=0}^{t-1} C_j^{i^j}, h_i) = e(g^{\sum_{j=0}^{t-1} \alpha_j i^j}, h_i) = e(g, h_i)^{p(i)}$. Similarly by the construction of Y_i the right-hand side is $e(g, Y_i) = e(g, h_i^{s_i}) = e(g, h_i)^{s_i}$. So together these give $s_i = p(i)$. The uniqueness of $p(x)$ is by the usual degree argument. □

Theorem 5.30. *If the verification of P_i's share pass, then $S_i = Y_i^{1/d_i}$.*

Proof. If the verification passes then $e(S_i, h_i) = e(Y_i, h)$. By the construction of S_i and the bilinearity of e the left-hand side becomes $e(S_i, h_i) = e(Y_i^{s_j}, h^{d_i}) = e(Y_i, h)^{s_i h_i}$. So by the verification we have $s_i = 1/d_i$ and hence $S_i = Y_i^{s_i} = Y_i^{1/d_i}$, as required. □

Now for the proof of security of the scheme. We show that an adversary who can find the secret of the PVSS scheme with an forbidden set of corrupted parties can be used as a blackbox for the DBS problem. Of interest in this proof is the way that the adaptive adversary is handled by the dealer. Initially the adversary chooses which parties to corrupt, if any, and tells the dealer. The dealer then guesses the remaining parties that the adversary will corrupt while the scheme takes places. if the guess is incorrect the the dealer's use of the adversary as a blackbox is of no use and it simply guesses for the DBS problem. So we work with the worst case probability in terms of the dealer guess, that is if the adversary initially chooses to corrupt no parties and later corrupts all $t-1$. Note that although these later corruptions could take place at any time, without loss of generality we can model them to all take place just before the reconstruction phase. This is because the adversary can then take advantage of seeing the encrypted shares broadcast by the dealer. Again we assume the worst case and send to the adversary all the possible encrypted messages between any pair of honest parties.

Theorem 5.31. *The Villar PVSS scheme is semantically secure against an adaptive adversary under the DBS assumption.*

Proof. Given an adversary Adv_{PVSS} to the scheme that can guess whether or not a given secret is the shared secret, we build an adversary Adv_{DBS} to the DBS problem. We assume the same parameters as previously, that is, G, G_1, e, q, h, g. First the DBS problem is set up by randomly selecting a, c, d in $\mathbb{Z}_q^*$ and setting $h = g^a$ and $m = g^c$. Then $b \in \{0, 1\}$ is randomly chosen and the challenge is set as $T_b = e(h, h)^{bc+(1-b)d}$. The Adv_{DBS} receives (m, h, T_b) from the DBS problem and must guess the value of b.

On the other side the Adv_{PVSS} selects the parties it initially corrupts $B_0 \subset \mathcal{P}$, with $|B_0| \leq t-1$. For each corrupt party $P_j \in B_0$ it sets a public key h_j and broadcasts it.

The Adv_{DBS} now guesses the final set of corrupt parties B such that $B_0 \subset B$ and $|B| \leq t-1$. For each $P_i \in B \backslash B_0$, the honest parties which it guesses will be corrupted later, it randomly selects private and public keys $d_i \in_R \mathbb{F}_q^*$ and $h_i = h^{d_i}$. Then for each $P_i \in \mathcal{P} \backslash B$, the honest parties which it guesses will stay honest, it selects $r_i \in_R \mathbb{F}_q^*$ and sets $h_i = g^{r_i}$. Finally it broadcasts $(h_i)_{i \in \mathcal{P} \backslash B_0}$.

Now Adv_{DBS} constructs shares for the parties it thinks will be corrupted by the end and, working backwards, finds the appropriate commitments for these shares. So for $j \in B$, it randomly chooses $s_j \in \mathbb{F}_q^*$ and sets $Y_j = h_j^{s_j}$. It finds $p(x) = \sum_{j=0}^{t-1} \alpha_j x^j$, the unique polynomial such that $p(j) = s_j$ and $g^{p(0)} = g^c$. So each α_j can be written as $\alpha_j = \sum_{\ell=0}^{t-1} (u_{\ell j} s_j) + u_{0j} c$ for some $u_{\ell j}$. Then it sets the commitments $C_j = g^{\sum_{\ell \in B} (u_{\ell j} s_i)} m^{u_{0j}}$ for $j \in [t-1]$ and $C_0 = m$. Lastly for $i \in \mathcal{P} \backslash B$ it sets $Y_i = h_i^{p(i)}$. It broadcasts all this information as $((Y_i)_{i \in [n]}, (C_j)_{j \in B}, T_b)$.

Having received these values, Adv_{PVSS} chooses the honest parties it wants to corrupt as $B_1 \subset \mathcal{P} \backslash B_0$ such that $|B_0 \cup B_1| \leq t-1$. It sends this new set B_1 to Adv_{DBS} in order to receive their secret keys. Before Adv_{DBS} sends these, it first checks if $B_1 \subset B \backslash B_0$, since if it guessed incorrectly there is no need to continue. So if $B_1 \not\subset B \backslash B_0$ then Adv_{DBS} guesses a random b'. Otherwise it broadcasts $(d_i)_{i \in B_1}$.

Now for the reconstruction, for any distinct pair $j, \ell \in \mathcal{P} \backslash (B_0 \cup B_1)$, then Adv_{DBS} randomly selects $\rho \in \mathbb{F}_q^*$ and sets $r = h_i^\rho, z = Y_i^\rho$ and $w = h^{r_\ell/(r_i \rho)}$ and broadcasts (r, z, w). Finally Adv_{PVSS} guess b' and Adv_{DBS} guess the same.

The probability that Adv_{DBS} correctly guesses the parties B_1 is

$$\gamma := \mathbb{P}(B = B_0 \cup B_1) = \frac{\binom{t-1-|B_0|}{|B_1|}}{\binom{n-|B_0|}{|B_1|}}.$$

The worst case of $|B_0| = 0$ and $|B_1| = t-1$ gives $\gamma = \frac{1}{\binom{n}{t-1}}$. Now if the guess is correct then $\mathbb{P}(b' = b) > 1/2 + \varepsilon$ by assuming that Adv_{PVSS} is non-negligibly successful. If the guess of B is incorrect then clearly b' is a random guess. So overall this gives

$$\begin{aligned} \mathbb{P}(b' = b) &= \gamma(1/2 + \varepsilon) + (1-\gamma)1/2 \\ &> 1/2 + \gamma\varepsilon, \end{aligned}$$

which as required, is a non-negligible probability of success against the DBS problem. □

5.3.3 Homomorphic property

The homomorphic property for secret sharing schemes is rather simple since it only requires operations on the shares of a party. For publicly verifiable secret sharing schemes this is more difficult, since we need the verifications to hold for the result of the operations too. This is especially true when using such methods as zero-knowledge proofs for the verifications, which have the inherent property of serving for nothing else than proving the knowledge of a single value. In this way, the Villar scheme's homomorphic property is of special interest since the simple verification method allows for an easy way to combine secrets. That is, any linear combination of the shares of some secrets is valid to recover the same linear combination of said secrets.

Theorem 5.32. *Given the information sets (C_j, Y_i) and (C'_j, Y'_i) broadcast by the dealer to share secrets s and s' respectively. Then for the secret $S^\alpha S'^\beta$, with $\alpha, \beta \in \mathbb{F}_q^*$, the information is $(C_j^\alpha C'^\beta_j, Y_i^\alpha Y'^\beta_i)$.*

Proof. By checking the validity of the shares as in the scheme we see that sending $Y_i^\alpha Y'^\beta_i$ with commitments $C_j^\alpha C'^\beta_j$ is valid for a secret. That is, we check that $e(X_i, h_i) = e(g, Y_i)$ where $X_i = \prod_{j=0}^{t-1} C_j^{i^j}$. Firstly,

$$X_i^\alpha X'^\beta_i = \prod_{j=0}^{t-1} C_j^{\alpha i^j} C'^{\beta i^j}_j = g^{\sum_{j=0}^{t-1} (\alpha_j \alpha + \beta_j \beta) i^j}.$$

And therefore we can directly write $e(X_i^\alpha X'^\beta_i, h_i) = e(g, Y_i^\alpha Y'^\beta_i)$, so the verification passes and the information sent represents consistent shares. Now we consider the reconstruction step to see that the shared secret is in fact $S^\alpha S'^\beta$. The reconstruction step is the same as in the scheme except we have $S_i = (Y_i^\alpha Y'^\beta_i)^{1/d_i}$ and $z = (Y_i^\alpha Y'^\beta_i)^\rho$. This gives

$$\begin{aligned} e(z, w)^{\lambda_i} &= e(Y_i^\alpha Y'^\beta_i, h_i^{1/d_i})^{\lambda_i} \\ &= e(h_i^{p(i)\alpha + p(i)'\beta}, h_i^{1/d_i})^{\lambda_i} \\ &= e(h, h)^{(p(i)\lambda_i + p(i)'\beta)\lambda_i}, \end{aligned}$$

and so for the reconstruction we find

$$\begin{aligned} \prod_{i \in A} e(z, w)_i^\lambda &= e(h, h)^{\alpha \sum_{i \in A} p(i)\lambda_i + \beta \sum_{i \in A} p(i)'\lambda_i} \\ &= e(h, h)^{\alpha p(0)} e(h, h)^{\beta p(0)'} \\ &= S^\alpha S'^\beta. \end{aligned}$$

□

6. Recent improvements in PVSS schemes

In this section we consider and contribute in a small way to the DHPVSS (Diffie-Hellman PVSS) scheme proposed in the YOLO YOSO paper [9], which itself comes as an improvement of previous works in SCRAPE [7] and ALBATROSS [8]. Originally built around the the Schoenmaker PVSS scheme, discussed in section 5.1.3, the improvements are two-fold. Firstly, it uses Reed-Solomon codes to allow for more efficient verification of distributed shares, as introduced in SCRAPE and explained in the next subsection. In this sense, the dealer commits and proves the validity of the shares it distributes, instead of the polynomial from which they are computed. Note that this requires a DLEQ proof for each share. The second improvement is to combine the code verification and the DLEQ proofs to instead require only one DLEQ proof, sufficient for all shares. This was introduced in the DHPVSS scheme and requires strong homomorphic properties from the encryption function. It is stated in the YOLO YOSO paper that ElGamal is the only known encryption for which such properties hold. Our contribution is to show that another, similar encryption scheme has the required properties and hence we present the DHPVSS with this new scheme. This encryption scheme was introduced by Castagnos and Laguillaumie in 2015 in [11] and we will refer to it as CL15. The advantage of this scheme over ElGamal's is that it can handle additive homomorphic encryption over a set of integers $\mathbb{Z}_p$ for any large p, hence it would be fit for a large voting operation.

6.1 SCRAPE adaptation of Schoenmaker

Let us consider a recent adaptation of the Schoenmaker PVSS scheme, as discussed in section 5.1.3. The main idea behind SCRAPE, as presented in [7], is to use the similarity between Shamir's secret sharing and Reed-Solomon codes. As discussed in section 3.4, shares constructed in Shamir's scheme form codewords of some Reed-Solomon code C. Due to this, we can check if a set of shares is valid by checking if as a sequence it is a valid codeword in C. This can be done by considering the inner product of the codeword and some codeword in the dual code $C^{\perp}$. If the shares are valid this will give zero, and if the shares are not valid then with high probability this will give a non-zero value. As we see in the next lemma, for a code based on $\mathbb{Z}_q$, checking invalid shares against a random codeword in the dual has probability $1/q$ of incorrectly returning that these are valid. And as expected, for higher accuracy it is possible to repeat the check multiple times to improve the probability as much as necessary. Recall that an $[n, k, d]$ code is a linear error-correcting code over $\mathbb{Z}_q$ of length n, dimension k and minimum distance d, and whose dual code $C^{\perp}$ is the vector space of $v \in \mathbb{Z}_q^n$ such that $\langle v, c \rangle = 0$ for all $c \in C$.

Lemma 6.1. *Let C be a $[n, k, d]$ code and $C^{\perp}$ its dual. For $v \in \mathbb{F}_q^n \backslash C$ and a randomly choosen $c^{\perp} \in C^{\perp}$ then $\mathbb{P}(\langle v, c^{\perp} \rangle = 0) = 1/q$.*

Proof. Let D be the vector subspace spanned by C and v together. Since $c^{\perp}$ is orthogonal to all vectors in C by definition, then by linearity it is orthogonal to all vectors in D if and only if is orthogonal to v. Since $v \notin C$, then the dimension of D is $\dim(D) = 1 + \dim(C^{\perp}) = k + 1$, and the dimension of $D^{\perp}$ is $n - k - 1$. So choosing $c^{\perp}$ uniformly at random, there are $|C^{\perp}| = q^{n-k}$ possible codewords, of which $|D^{\perp}| = q^{n-k-1}$ are orthogonal to v. So $\mathbb{P}(\langle v, c^{\perp} \rangle = 0) = \frac{|D^{\perp}|}{|C^{\perp}|} = \frac{1}{q}$ as required. □

With this in mind we consider the changes made to the Schoenmaker scheme expressed in SCRAPE. The principal difference is that the dealer commits to the shares instead of commiting to the polynomial from which they are derived. This is done for a share s_i by broadcasting $Y_i = pk_i^{s_i}$ with a commitment $v_i = g^{s_i}$, where pk_i is the public key of party P_i. To verify the validity of the shares, a verifier can take a random codeword $c^{\perp} = (c_1^{\perp}, \dots, c_n^{\perp})$ of the dual code and check that $\prod_{i\in[n]} v_i^{c_i^{\perp}} = 1$. This is equivalent to the inner product idea above since $\prod_{i\in[n]} v_i^{c_i^{\perp}} = g^{\sum_{i\in A} s_i c_i^{\perp}} = g^{\langle v, c^{\perp}\rangle}$. So the product will give 1 if and only if the inner product gives 0. An additional step is required here to prove that the encrypted shares are the same as the ones in the commitments, this can be done by a zero-knowledge proof. So once the shares and commitments are broadcast by the dealer, a verifier can check $\text{DLEQ}(g, v_i, pk_i, Y_i)$ for each $i \in [n]$.

Let us reconsider the verifiability of the scheme with this adaptation. We want that the verification fails for an invalid set of shares, where invalidity here can come from two possibilities. Either not all encrypted shares are the same as in the their respective commitments, then $\log_g(v_i) \neq log_{pk_i}(Y_i)$ for some $i \in [n]$, and the $\text{DLEQ}(g, v_i, pk_i, Y_i)$ check will fail. Or the shares are not consistent, then $v = (v_1, \dots, v_n) \notin C$ and the inner product check will fail with overwhelming probability. So if the verifications hold, then it is safe to assume that the shares are valid and consistent.

6.2 Adapting ElGamal encryption

As previously mentioned, the DHPVSS scheme uses ElGamal encryption due to its linearly homomorphic properties which allow short proofs of knowledge. That is, it is easy to prove that one knows the original message of an encryption in zero-knowledge. Similarly one needs to be able to show that is has correctly decrypted an encrypted message, again in zero-knowledge. The ElGamal scheme was the only encryption scheme for which it was known that these properties hold. Although we have so far been considering the multiplicatively homomorphic ElGamal scheme, it is definitively interesting to look for an additively homomorphic scheme. In the common application of PVSS schemes for voting systems where the votes are tallied up while encrypted and hence no individual vote is decrypted, the operation needed is naturally addition. For this we can look at ElGamal in the exponent, where instead of encrypting a message m we instead encrypt g^m, in which case the product of the encryptions of two messages m_1, m_2 would be $g^{m_1+m_2}$ and hence give the sum of the messages. However this requires being able to compute the discrete log to recover the sum of the messages. This is commonly assumed to be easy if the exponent is small, but this may not be the case in a voting system with many voters. To solve this problem we consider the CL15 encryption scheme presented in [11]. The principal idea of the CL15 scheme is to use a group in which the *DDH* problem is hard and which has a subgroup in which the *DL* problem is easy. In this way, encrypting to the subgroup makes decryption as easy as taking the discrete log of the encrypted message.

In this section we introduce the CL15 encryption scheme and show that it has the required properties of homomorphic encryption to work with the DHPVSS scheme. Note that this contribution is somewhat small, since although ElGamal was the only scheme for which it was thought to hold, CL15 is very close to ElGamal, and this is why it was considered in the first place. Then we present an adaptation of the DHPVSS scheme with CL15 encryption. We also note the advantage of using this encryption is that the message space is $\mathbb{Z}_p$ for some p which can be made as large as necessary and in which any number of addition operations can be carried out.

Definition 6.2. A CL15 tuple is defined as (B,n,p,g,f,G,F) for which Bp, with p prime, is an upper bound for n, $G = \langle g\rangle$ is a cyclic group of *unknown* order n in which the *DDH* problem is assumed to be hard, and $F = \langle f\rangle$ is a subgroup of G of order p in which the *DL* problem is assumed to be easy.

This definition and the following discussion of the encryption scheme are simplifications of the full idea of *DDH* groups with easy *DL* subgroups as presented in [11]. See also further improvements in terms of simplicity and efficiency in [12] and [10]. These consider in full detail implementations of such a group using imaginary quadratic fields and their class groups. We will not go into these details and simply assume that such groups exist.

Definition 6.3. The CL15 encryption scheme for a CL15 tuple (B, p, g, f, G, F) is as follows.

Key generation:

1. B randomly selects a secret key $sk \in_R [Bp - 1]$,
2. B sets its public key g^{sk} and sends it to A.

Encryption:

1. A randomly picks $r \in_R [Bp - 1]$,
2. A encrypts message m as $(c_1, c_2) = (g^r, f^m pk^r)$ and sends it to B.

Decryption:

1. B having received (c_1, c_2) can compute $M = c_2 c_1^{-sk}$,
2. B recovers the message as $m = \log_f(M)$.

Clearly the CL15 encryption scheme is closely related to the ElGamal scheme. Note that $Bp - 1$ is used instead of n since the latter is the order of the group G and must remain unknown. This is required to maintain the hardness of the partial discrete log problem in G, a detail we will not go into but which is presented in [11] and was first introduced in [24]. Let us firstly consider the straight-forward proof of security of the scheme.

Theorem 6.4. *The CL15 encryption scheme is semantically secure against IND-CPA attacks under the DDH assumption.*

Proof. We construct an adversary $\mathcal{B}$ which uses a CL15 adversary $\mathcal{A}$ to break a DDH challenge with the same advantage as $\mathcal{A}$, denoted ε. Let the *DDH* challenger send to $\mathcal{B}$ the CL15 tuple (B, p, g, f, G, F) and the challenge $(X = g^x, Y = g^y, Z)$ where $Z = g^{xy}$ if $b_1 = 0$ and $Z = g^c$ if $b_1 = 1$, given $x, y, c \in_R [Bp - 1]$. Then $\mathcal{B}$ sets its public key $pk = X$ and sends it to $\mathcal{A}$. In return it receives two messages $m_0, m_1 \in [Bp - 1]$. $\mathcal{B}$ randomly selects $b_2 \in_R \{0, 1\}$ and sends the encryption $(Y, f^{m_{b_2}} Z)$ to $\mathcal{A}$. Receiving $\tilde{b}_2$ in return, $\mathcal{B}$ guesses $\tilde{b}_1 = 0$ if $\tilde{b}_2 = b_2$ and $\tilde{b}_1 = 1$ otherwise.

In the case that the challenge was a valid DDH tuple then $Z = g^{xy}$, so the encryption was valid and with probability $1/2 + \varepsilon$, $\mathcal{A}$ guesses $\tilde{b}_2$ correctly, so $\mathcal{B}$ should guess $\hat{b}_1 = 0$. If the DDH challenge was not a valid DDH then Z was random and hence the encryption was random and so is $\mathcal{A}$'s guess. So if $\hat{b}_2 \neq b_2$ then $\mathcal{B}$ should guess $\hat{b}_1 = 1$. Altogether this gives $\mathcal{B}$ a non-negligible advantage of $\varepsilon/2$. $\square$

6.3 Proofs through homomorphisms

As we have claimed and as will be needed later for the DHPVSS adaptation, we now consider the linearly homomorphic properties of the CL15 encryption scheme. Consider messages m, m' encrypted with a public key pk as (c_1, c_2) and (c_1', c_2') respectively. Then it is straight-forward to see that an encryption of the sum $m + m'$ can be computed as $(c_1^*, c_2^*) = (c_1 c_1' g^r, c_2 c_2' pk^r)$ for some random $r \in_R [Bp - 1]$. Then the decryption follows as

$$M = c_2^*(c_1^*)^{-sk} = \frac{c_2 c_2' pk^r}{(c_1 c_1' g^r)^{sk}} = f^{m+m'},$$

where $\log_f(M) = m + m'$ as required. The same can be done for multiplication by a scalar. Again for a message m encrypted as (c_1, c_2), and for an integer $\alpha \in \mathbb{Z}_p$, the encryption of αm is $(c_1^*, c_2^*) = ({c'}_1^{\alpha} g^r, {c'}_2^{\alpha}, pk^r)$ for some random $r \in [Bp - 1]$. Then the decryption finds, as required,

$$\log_f((c_2/c_1^{sk})^{\alpha}) = \log_f(f^{m\alpha}) = m\alpha.$$

The other property we are interested in is linear decryption, as it is referred to in YOLO YOSO. This says that the decryption function which we denote $D_c(sk)$ is linear in sk, that is, $D_c(sk_1 + sk_2) = D_c(sk_1) + D_c(sk_2)$ for any secret keys sk_1, sk_2 and encrypted message c. This will allow for a party to prove in zero-knowledge that it has correctly decrypted a share. Let us introduce these non-interactive zero-knowledge proofs in a general setting and then consider the two examples that will be used in the scheme. We define these following Schnorr's protocol for proofs of knowledge. Consider a finite field $\mathbb{F}$, vector spaces $\mathcal{W}, \mathcal{X}$ over $\mathbb{F}$ and a vector space homomorphism $f : \mathcal{W} \to \mathcal{X}$. Let $R = \{(w, x) \in \mathcal{W} \times \mathcal{X} | f(w) = x\}$ be the set of relations between a claim $x \in \mathcal{X}$ and a witness $w \in \mathcal{W}$ for the claim. Then the proof protocol between a prover P and a verifier V is defined as follows.

Definition 6.5. The protocol $\Pi(w; f, x)$ for an $\mathbb{F}$-vector space homomorphism $f : \mathcal{W} \to \mathcal{X}$ with respect to the relation R is:

1. P samples $r \in_R \mathcal{W}$, computes a commitment $a = f(r)$ and sends it to V,
2. V samples a challenge $e \in_R \mathbb{F}$ and sends it to P,
3. P computes the response $z = r + ew$ and sends it to V,
4. V verifies that $f(z) = a + ex$.

Since f is a homomorphism then when V checks the proof it finds the required

$$f(z) = f(a + ew) = f(r) + f(e)f(w) = a + ex. \tag{7}$$

This can be transformed into a non-interactive zero-knowledge proof by using the Fiat-Shamir heuristic. Hence instead of sending the commitment to the verifier and receiving a challenge in return, the prover computes a pseudo-random challenge $e = H(x, a)$ from the claim and the commitment. The verification for this is simply to have V check if $e = H(x, f(z) - ex)$ which holds if P is honest due to equation (7).

The two non-interactive zero-knowledge, or NI-ZK, proofs that will be used are as follows. The first will prove knowledge of a discrete log of the form $g^{sk} = pk$, for a secret key and a respective public key. We denote this by

$$DL(sk; g, pk) = \Pi(sk; f(x) = g^x, pk). \tag{8}$$

The second proof is for knowledge of equality between two discrete logs and the corresponding value and is of the form

$$\begin{cases} g^{sk} = pk, \\ \beta^{sk} = \alpha, \end{cases}$$

which we denote as $DLEQ(sk, g, pk, \beta, \alpha) = \Pi(sk; f(x) = (g^x, \beta^x), (pk, \alpha))$. This is used in two ways. The dealer will use it show to a valid distribution of the shares and the parties will use it to show valid decryptions of the encrypted shares.

A final comment on the importance of the validation of the key pairs as seen in equation (8). Since an adversary can wait for all honest parties to submit their public keys, it could then compute its own public keys which allow it to recover the secret from the encrypted shares. Consider an adversary $\mathcal{A}$ that controls a single corrupt party P_n, and all other parties $P_1, \dots, P_{n-1}$ are honest. Let $\mathcal{A}$ wait for all honest parties to have shared their public keys pk_i. Then, if the adversary does not need to prove that they know a secret key equal to the discrete log of the public keys for the parties it controls, it sets

$$pk_n = \prod_{i<n} pk_i^{-\lambda_i} \tag{9}$$

where $\lambda_i = \prod_{j \neq i} \frac{j}{j-i}$. Once the dealer sends the shares $\sigma_i = m(i)$ for a secret s as the encryptions $(a_i, A_i) = (g^{sk_D}, f^{sigma_i} pk_i^{sk_D})$, then $\mathcal{A}$ can recover the secret by computing

$$\begin{aligned}
A_i f^{\sigma_n} \prod_{i<n} A_i^{\lambda_i} &= pk_n^{sk_D} \prod_{i<n} \left(pk_i^{sk_D \lambda_i} f^{\sigma_i \lambda_i} \right) \\
&= \prod_{i<n} \left(pk_i^{-\lambda_i sk_D} \right) \prod_{i<n} \left(pk_i^{sk_D \lambda_i} f^{\sigma_i \lambda_i} \right) \quad \text{by equation (9)} \\
&= f^{\sum_{i \in [n]} m(i) \lambda_i} \\
&= f^{m(0)}.
\end{aligned}$$

6.4 Adapted DHPVSS scheme

Combining the previous sections' work on verification using Reed-Solomon dual-codes and homomorphic properties of the CL15 encryption scheme, we can now present the adapted DHPVSS scheme. Note that the scheme itself works in almost the same way as with ElGamal, the only difference being that with CL15 encryption, we may freely use the additive homomorphism without the requirement that the encrypted message must be small.

Definition 6.6. The adapted DHPVSS scheme runs as follows for dealer D to share secret $s \in \mathbb{Z}_p$ to parties P_i for $i \in [n]$, with some verifier V.

Setup:

1. D generates a CL15 tuple (B, p, g, f, G, F) as defined in definition 6.2 and broadcast it as the public parameters pp,
2. D randomly selects a secret key $sk_D \in_R \mathbb{Z}_p^*$ and sets a public key $pk_D = g^{sk_D}$ which it broadcasts,
3. Party P_i randomly selects a secret key $sk_i \in_R \mathbb{Z}_p^*$ and sets a public key $pk_i = g^{sk_i}$ which it broadcasts,
4. P_i also broadcasts a NI-ZK proof of knowledge of its secret key as $\Omega_i = DL(sk_i; g, pk_i)$.

Key verification:

1. V can check P_i broadcasted a valid public key, that is, Ω_i proves knowledge of sk_i.

Distribution:

1. D randomly selects a polynomial $m(x) \in \mathbb{Z}_p[x]$ of degree at most t and such that $m(0) = s$,
2. D sets the shares $\sigma_i = m(i)$,
3. D encrypts the shares as $(a_i, A_i) = (g^{sk_D}, f^{\sigma_i} pk_i^{sk_D})$ and broadcasts them,
4. D sets a code word $m^* = H(pk_D, \{pk_i, (a_i, A_i)\}_{i \in [n]})$ and coefficients $d_i = m^*(\alpha_i) \prod_{j \in [n] \setminus \{i\}} \frac{1}{i-j}$ for $i \in [n]$,

5. D sets $A = \prod_{i\in[n]} A_i^{d_i}$ and $B = \prod_{i\in[n]} pk_i^{d_i}$,
6. D broadcasts a NI-ZK proof of valid distribution $\Omega_D = DLEQ(sk_D; g, pk_D, B, A)$.

Distribution verification:

1. V recovers the code word m^* and the coefficients d_i,
2. V computes $A = \prod_{i\in[n]} A_i^{d_i}$ and $B = \prod_{i\in[n]} pk_i^{d_i}$,
3. V checks that the proof Ω_D holds with respect to g, pk_D, B, A.

Decryption:

1. P_i computes $A_i' = A_i pk_D^{-sk_i}$,
2. P_i computes and broadcasts the decrypted share $\sigma_i' = \log_f(A_i')$,
3. P_i broadcasts a NI-ZK proof of correct decryption $\Omega_i' = DLEQ(sk_i; g, pk_i, pk_D, A_i/A_i')$.

Decryption verification:

1. V can check that Ω_i' is a valid proof with respect to $g, pk_i, pk_D, A_i/A_i'$.

Reconstruction: for an authorised set of parties, w.l.o.g. P_i for $i \in [t]$.

1. P_i computes the coefficients $\lambda_j = \prod_{\ell\in[t]\setminus\{j\}} \frac{\ell}{\ell-j}$ and recovers the secret $s' = \sum_{j\in[t]} \sigma_j \lambda_j$.

Theorem 6.7. *The DHPVSS scheme with CL15 encryption is verifiable.*

Proof. The verifiability of the key generation is straight forward. If Ω_i is a valid proof then $DL(sk; g, pk_i)$ holds, so the prover, P_i, knows some α such that $g^\alpha = pk_i$, i.e. $\alpha = sk_i$, as required.

Now for the verifiability of the distribution. If Ω_D is valid then $DLEQ(sk_D; g, pk_D, B, A)$ holds and hence the dealer knows some γ such that $DL_g(pk_D) = DL_B(A) = \gamma$. This can be expressed as the equations

$$\begin{cases} g^\gamma = pk_D, & (10) \\ B^\gamma = A. & (11) \end{cases}$$

By the verifiability of the key generation and equation (10) then $\gamma = sk_i$. Since V can compute A and B, equation (11) gives

$$\prod_{i\in[n]} (pk_i^{d_i})^{sk_D} = \prod_{i\in[n]} A_i^{d_i}. \tag{12}$$

Suppose that the shares are not valid as shares from any polynomial of degree at most t. That is, for all $m(x) \in \mathbb{Z}_p[x]_{\leq t}$, then there is some $i \in [n]$ for which $A_i pk_i^{-sk_D} \neq f^{m(i)}$. Let r be the number of queries made to the oracle for a codeword m^* in the dual. Then by the dual code lemma 6.1 we claim that the probability that $\prod_{i\in[n]}(A_i pk_i^{-sk_i})^{d_i} = 1$, i.e. equation (12), is at most r/p. This is because for share σ_i then $A_i pk_i^{-sk_D} = f^{\sigma_i}$ and hence denoting $\sigma = (\sigma_1, \ldots, \sigma_n)$ and $d = (d_1, \ldots, d_n)$ we have

$$\prod_{i\in[n]} (A_i pk_i^{-sk_D})^{d_i} = f^{\langle \sigma, d\rangle}.$$

So equation (12) holds for these invalid shares only when $\langle \sigma, d\rangle = 0$, that is, with negligible probability no more than r/p.

Finally for the verifiability of the decryption. If Ω'_i is valid then $DLEQ(sk; g, pk_i, pk_D, A_i/A'_i)$ holds, hence $\log_g(pk_i) = \log_{pk_D}(A_i/A'_i) = \phi$ for some ϕ. Again due to the verifiability of the key generation giving $\phi = sk_i$ this can be written

$$\begin{cases} g^{sk_i} = pk_i, \\ pk_D^{sk_i} = A_i/A'_i. \end{cases} \tag{13}$$

By the verifiability of the disribution, A_i is an encrypted share and so of the form $A_i = f^{\sigma_i} pk_D^{sk_i}$ for some σ_i. Then equation (13) gives $A'_i = f^{\sigma_i}$, so the decrypted share is $\log_f(A'_i) = \sigma_i$. □

Now we consider the security of the scheme, which is proved under the *DDH* assumption. The general idea behind the proof is to consider the *DDH* tuple (g, g^a, g^b, w_b) with $w_b \in_R \{g^{ab}, g^c\}$ as public keys for the dealer and some randomly chosen honest party. That is, $pk_D = g^a$ and $pk_k = g^b$ for some honest P_k. The adversary $\mathcal{B}$ we construct for the *DDH* game does not have the secret keys for either, but it does not need them to simulate a DHPVSS sharing for some blackbox adversary $\mathcal{A}$. More specifically, to encrypt some share σ_i for party P_i, then $\mathcal{B}$ uses the fact that $pk_i = g^{sk_i}$ for some sk_i and hence $f^{\sigma_i} pk_i^{sk_D} = f^{\sigma_i} pk_D^{sk_i}$. This requires that all parties public keys are of the correct form of $pk = g^{sk}$, which is known to be true since the public keys are verified to show that the party knows such an sk.

To make use of $\mathcal{A}$'s advantage, $\mathcal{B}$ distributes shares of two different secrets using randomly chosen $b_2 \in_R \{0, 1\}$ and some $j \in [n]$. The distribution is such that when $j = 1$ and $b_2 = 1$ then $\mathcal{A}$ sees valid shares for one secret and, when $j = n - t$ and $b_2 = 0$ then $\mathcal{A}$ sees valid shares for the other. For any other combination of j and b_2, it receives an invalid combination of shares for both. In this way, $\mathcal{B}$ can guess if the *DDH* tuple was valid by seeing if $\mathcal{A}$'s guess is correct.

Theorem 6.8. *The DHPVSS scheme with CL15 encryption is semantically secure under the DDH assumption.*

Proof. Consider an adversary $\mathcal{A}$ which, given a forbidden set of shares can guess with non-negligible advantage whether or not this is a given secret. Then we construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ as a blackbox to dinstinguish *DDH* tuples with non-negligible advantage. We now explain how the $\mathcal{B}$ adversary is constructed. Without loss of generality we assume that of the n parties, the ones corrupted by $\mathcal{A}$ are $P_{n-t+1}, \dots, P_n$, and the rest, $P_1, \dots, P_{n-t}$ are honest.

First the *DDH* challenger randomly selects $sk_D, sk, c \in_R \mathbb{Z}_p$ and sets $w_0 = g^{sk_D sk}$ and $w_1 = g^c$. It randomly selects $b_1 \in_R \{0, 1\}$ and sends $(g, g^{sk_D}, g^{sk}, w_{b_1})$ to $\mathcal{B}$.

Now $\mathcal{B}$ randomly chooses an honest party $k \in_R [n - t]$ and sets its private and public keys using the sk received from the *DDH* challenger, so $sk_k = sk$ and $pk_k = g^{sk_k}$. $\mathcal{B}$ can simulate the NI-ZK proof Ω_k without knowing sk_k as follows. It randomly selects $z, e \in_R \mathbb{Z}_p$ and sets $a = g^z pk_k^{-e}$. The transcript of this simulated proof reads as (a, e, z), for which $g^z = a \cdot pk_k^e$ as required.

For $i \in [n - t] \backslash \{j\}$, the other honest parties, $\mathcal{B}$ randomly picks private keys $sk_i \in_R \mathbb{Z}_p$ and sets public keys $pk_i = g^{sk_i}$ and proofs $\Omega_i = DL(sk_i; g, pk_i)$. Sending the public keys and proofs of all honest parties $(pk_i, \Omega_i)_{i \in [n-t]}$ to $\mathcal{A}$, it receives in return those of the corrupt parties $(pk_j, \Omega_j)_{j \in [n-t+1,n]}$. Since $\mathcal{A}$ is run as a blackbox, $\mathcal{B}$ can extract from each proof Ω_j the corresponding secret key sk_j for $j \in [n - t + 1, n]$.

Now $\mathcal{B}$ randomly picks two secrets $s_0, s_1 \in_R \mathbb{Z}_p$ and randomly selects two polynomials $m(x), m'(x) \in_R \mathbb{Z}_p[x]$ such that $m(0) = s_0$, $m'(0) = s_1$ and $m(i) = m'(i)$ for $i \in [n - t + 1, n]$. Note that these polynomials give the same evaluation for all corrupt parties. $\mathcal{B}$ sets the shares as $\sigma_i = m(i)$ and $\sigma'_i = m'(i)$.

To encrypt the shares $\mathcal{B}$ does the following. For share σ_k, $\mathcal{B}$ randomly selects its own $b_2 \in_R \{0,1\}$, and then sets encryption as

$$\begin{cases} A_k = f^{\sigma_k} w_b, & \text{if } b_2 = 0 \\ A_k = f^{\sigma'_k} w_b, & \text{otherwise.} \end{cases}$$

The shares of the corrupt parties are encrypted as $A_j = f^{\sigma_j} pk_D^{sk_j}$ for $j \in [n-t+1, n]$. Finally the shares of the remaining honest parties are encrypted as

$$\begin{cases} A_i = f^{\sigma_i} pk_D^{sk_i}, & \text{for } i \in [j-1] \\ A_i = f^{\sigma'_i} pk_D^{sk_i} & \text{for } i \in [j+1, n-t]. \end{cases}$$

For all encryptions, $a_i = g^{sk_D}$ as $\mathcal{B}$ receives from the DDH challenge.

The last thing for $\mathcal{B}$ to do is to construct the NI-ZK proof of valid distribution, Ω_D. It sets a random word $m^*(x) = H(pk_D, \{pk_i, (a_i, A_i)\}_{i\in[n]})$ in the dual code and the coefficients $d_i = m^*(i) \prod_{j\in[n]\setminus\{i\}} \frac{1}{i-j}$. It computes $A = \prod_{i\in[n]} A_i^{d_i}$ and $B = \prod_{i\in[n]} pk_i^{d_i}$ with which it can simulate the proof of distribution $\Omega_D = DLEQ(sk_D; g, pk_D, B, A)$ as follows. It randomly selects $z, e \in_R \mathbb{Z}_p$ and sets $(a,b) = (g^z pk_D^{-e}, B^z A^{-e})$. The proof transcript is $((a,b), e, z)$ such that $g^z = apk^e$ and $B^z = bA^e$. Finally $\mathcal{B}$ sends to $\mathcal{A}$ the encrypted shares and the proof $(\{a_i, A_i\}_{i\in[n]}, \Omega_D)$. It receives in return some guess $\tilde{b}_2 \in \{0,1\}$ and guess itself $\tilde{b}_1 = 1$ if $\tilde{b}_2 = b_2$ and $\tilde{b}_1 = 0$ otherwise.

Now we consider how $\mathcal{B}$ breaks the *DDH* game if $\mathcal{A}$ breaks the DHPVSS game. Recall that the shares of the corrupt parties are valid for both secrets s_0 and s_1 by construction of the polynomials $m(x), m'(x)$. When $j = 1$ and $b_2 = 1$, the shares of the honest parties are for the secret s_1. So $\mathcal{A}$ is sent s_0 and shares for s_1, this is equivalent to the case $b = 1$ in the DPVSS game. On the other hand when $j = n-t$ and $b_2 = 0$, the shares of the honest parties are for the secret s_0 and hence the adversary sees case $b = 0$. If $\mathcal{A}$ can break the DHPVSS game then the probability that it correctly guesses $\tilde{b}_2 = 1$ is non-negligibly larger than the probability that it incorrectly guesses $\tilde{b}_2 = 1$. That is, for non-negligible ε the advantage of $\mathcal{A}$ is

$$\Delta_{\mathcal{A}} := \mathbb{P}(\tilde{b}_2 = 1 | b_2 = 1, j = 1) - \mathbb{P}(\tilde{b}_2 = 1 | b_2 = 0, j = n-t) > \varepsilon. \tag{14}$$

By the assignment of shares, one can check that the shares are equivalent for $b_2 = 1, j = i+1$ and $b_2 = 0, j = i$ for $i \in [n-t-1]$. Then equation (14) can be written

$$\Delta_{\mathcal{A}} = \sum_{i\in[n-t]} \Big(\mathbb{P}(\tilde{b}_2 = 1 | b_2 = 1, j = i) - \mathbb{P}(\tilde{b}_2 = 1 | b_2 = 0, j = i) \Big). \tag{15}$$

Now considering the probability that $\mathcal{B}$ guesses correctly when given an invalid *DDH* tuple, this happens if $\tilde{b}_2 = b_2$, hence

$$\begin{aligned} \mathbb{P}(\tilde{b}_1 = 1 | b_1 = 1) &= \frac{1}{2(n-t)} \sum_{i\in[n]} \Big(\mathbb{P}(\tilde{b}_2 = 1 | b_2 = 1, j = i) + (1 - \mathbb{P}(\tilde{b}_2 = 1 | b_2 = 0, j = i)) \Big) \\ &= 1/2 + \frac{1}{2(n-t)} \sum_{i\in[n]} \Big(\mathbb{P}(\tilde{b}_2 = 1 | b_2 = 1, j = i) - \mathbb{P}(\tilde{b}_2 = 1 | b_2 = 0, j = i) \Big) \\ &> 1/2 + \frac{\varepsilon}{2(n-t)}, \end{aligned}$$

where the inequality comes from equations (14) and (15). Clearly when $b_1 = 0$, $\mathcal{B}$ is given a random g^c and hence the shares sent to $\mathcal{A}$ are not valid, so $\tilde{b}_2$ is a random guess and so is $\tilde{b}_1$, giving

$$\mathbb{P}(\tilde{b}_1 = 1 | b_1 = 0) = 1/2.$$

Overall $\mathcal{B}$'s advantage is therefore

$$\frac{\mathbb{P}(\tilde{b}_1 = 1 | b_1 = 1) + \mathbb{P}(\tilde{b}_1 = 1 | b_1 = 0)}{2} = \frac{\varepsilon}{4(n-t)},$$

which is non-negligible. $\square$

7. Generalisations

We have so far considered schemes based almost exclusively on the (t, n) threshold access structure and which broadly follow Shamir's methods of distribution and reconstruction using polynomials and interpolation. However all these schemes considered in chapters 5 and 6 can be generalised to any linear secret sharing scheme and hence to any access function. For simplicity we continue our explanation for an ideal scheme, ie. where all parties are either authorised or forbidden, since this can be generalised easily enough. As discussed in section 3.4, we already know that these can be generalised to linear schemes using codes. In [27], Schoenmaker explains an extension of its scheme to any linear access structure, that is, using only linear operations for computing the shares and recovering the secret. The main idea is as follows.

Consider a monotone access structure Γ on a set of parties $[n]$ with the shares and secret in $\mathbb{Z}_q$. A matrix M of size $(n+1) \times k$ is required such that for any authorised set $B \subset \Gamma$, and only for such an authorised set, there is a $c \in \mathbb{Z}_q^{|B|}$ for which

$$cM_B = e_1 := (1, 0, \dots, 0), \tag{16}$$

where M_B is the matrix of the rows of B. The matrix M will be used to generate the shares and the requirement can be seen as guaranteeing that any authorised set will be able to recover the chosen element, in this case e_1, and hence the secret. The dealer randomly selects a column vector $a \in \mathbb{Z}_q^d$, setting the random secret as $s = a_1$ and computing the shares as $s_i = M_i \cdot a$. The rest of the scheme can be carried out as in the original scheme presented in section 5.10 until the reconstruction. At this point each participant has decrypted its share as g^{s_i} for $i \in B$. By the above requirement (16) there is a c such that $cM_B = e_1$ in which case the secret is recovered as

$$\prod_{i \in B} (g^{s_i})^{c_i} = g^{(\sum_{i \in B} c_i M_i)a} = g^{e_1 a} = g^s.$$

So the Schoenmaker scheme can be generalised to any linear access structure in this way. As we saw in section 6.1, as similar generalisation to codes is used in SCRAPE to allow for verification of the shares. That is, a set of shares is valid if they are valid as codewords, which can be checked using dual code.

7.1 Abelian secret sharing

We can see that the schemes so far studied as transforming a secret and its shares $(s_0; s_1, \dots, s_n)$ into some form of encrypted shares $(g^{s_0}; g^{s_1}, \dots, g^{s_n})$, from which the parties can recover the secret. That is, the schemes take a elements in a linear space and encrypt them to Abelian groups, similarly with commitments. An interesting idea would be to generalise this original linear space to an Abelian group. Let us now consider the generalisation to Abelian secret sharing schemes, basing ourselves on the corresponding sections of [21] which introduced this idea. Firstly we can define an Abelian secret sharing scheme as one based on Abelian groups.

Definition 7.1. For Abelian groups G_i, $i \in [n]$, an Abelian secret sharing scheme is a code C which is a subgroup of $G_0 \times \cdots \times G_n$.

Again we denote a codeword in C as $(s_0, \dots, s_n)$, and note that since C is a subgroup, for two codewords $(s_0, \dots, s_n), (s'_0, \dots, s'_n)$ then their sum $(s_0 + s'_0, \dots, s_n + s'_n)$ is also in C. Notice that in the particular case where the Abelian groups are $G_i = \mathbb{F}_q^{r_i}$, then the code C is a vector subspace and hence it generates a linear secret sharing scheme.

For the authorised and forbidden sets, since we are no longer assuming that the access function is perfect, we define these individually. So the set of authorised parties are denoted as Γ_1 and the set of forbidden parties are Γ_0, where a party may be in neither set. Again for ease of notation we consider an authorised party A, without loss of generality we assume that $A = \{1, \dots, k\}$ and we denote its complement $P \setminus A =: B$. Hence when writing a codeword representing a secret and a set of shares we simply write s_0, S_A, s_B to denote $(s_0; s_1, \dots, s_k, s_{k+1}, \dots, s_n)$. We will also always speak of an authorised set A and a forbidden set B. Now we can rewrite the original definition of authorised and forbidden sets from 3.2 in terms of the codewords in C. Recall this said a set A was authorised if it could recover the secret from its shares, and a set B forbidden if its shares give it no information about the secret, that is, there are as many codewords with s_0 for any given s_B.

Lemma 7.2. *Authorised and forbidden sets can be expressed in the following ways:*

- *A is authorised if and only if $s_A = 0$ implies $s_0 = 0$,*
- *B is forbidden if and only if for all $s_0 \in G_0$ there is a codeword $s_0, s_A, 0 \in C$.*

Proof. To show that $s_A = 0 \implies s_0 = 0$, consider a codeword $s_0, 0, s_B$, since $0, 0, 0$ is also a codeword then both s_0 and 0 are recoverable secrets from the set of shares 0, so they must be equal, $s_0 = 0$, as required. In the other direction, to show that A is authorised, consider two codewords with the same A shares, s_0, s_A, s_B and s_0', s_A, s_B', then their difference is also in C, ie. $s_0 - s_0', 0, s_B - s_B'$. Since the A shares of this new codeword are 0 then its secret must also be 0, so $s_0 = s_0'$. Hence a given set of A shares always recovers the same secret.

Now to show that there is always a codeword $s_0, s_A, 0$ given some $s_0 \in G_0$. Consider some $s_0 \in G_0$, then $s_B = 0$ has as many valid codewords s_0, s_A, s_B as for any other s_B. If this number is zero then s_0 is not a secret in the scheme, otherwise there is some $s_0, s_A, 0 \in C$, as required. Finally to show that B is forbidden. Given s_0 we want to show that all s_B have the same number of codewords s_0, s_A, s_B. Consider some s_B, then for any $s_0, s_A, s_B \in C$, we know there is a codeword $0, s_A', 0$, so taking their sum $s_0, s_A - s_A', s_B \in C$. That is, each s_B has as many codewords as there are possible s_A'. □

We now consider three constructions of such schemes. In all three we consider the groups to be subgroups of some G, which we call the main group. For the first construction, consider mapping to the quotient groups of the subgroups as follows. For subgroups $H_0, \dots, H_n$ of a main group G, consider the map from G to the quotient groups $G_i = G/H_i$, mapping g to its respective class $s_i = gH_i$. Note that when sharing some $g \in G$, if it is in some subgroup H_i then the i-th share is $gH_i = H_i$, or the identity in G/H_i. Now referring back to Lemma 7.2 to express authorised and forbidden sets in terms of the groups H_i gives the following. The requirement for some set of parties A being authorised should say that the intersection of all H_i for $i \in A$, ie. the subset of G from which every element maps to the identity for all parties in A, should be a subset of H_0, so that the secret is zero when the A share is zero. For a set of parties B to be forbidden we should require that all of G is covered by H_0 and $\bigcap_{i \in B} H_i$, ie. for any non-zero secret there is some codeword with all B shares zero. More formally we can write the requirements as follows.

Lemma 7.3. *The following equivalences for authorised and forbidden sets of parties hold for the first construction.*

1. $A \in \Gamma_1 \iff \bigcap_{i \in A} H_i \subset H_0$,
2. $B \in \Gamma_0 \iff H_0 + \bigcap_{i \in B} H_i = G$.

Proof. To show that $\bigcap_{i\in A} H_i \subset H_0$, consider some $g \in \bigcap_{i\in A} H_i$, then the codeword generated by g is $s_0, 0, s_B$. Since the A shares are zero then $s_0 = 0$ and hence $g \in H_0$. Now to show A is authorised, consider a codeword $s_0, 0, s_B$ then $g \in \bigcap_{i\in A} H_i$. By the assumption then $g \in H_0$ and hence $s_0 = 0$.

To show $H_0 + \bigcap_{i\in B} H_i = G$, consider some $g \in G$ which generates the codeword s_0, s_A, s_B. By the assumption that B is forbidden there is some $g' \in \bigcap_{i\in B} H_i$ which generates $s_0, s'_A, 0$. Letting g'' be the element generating the difference of the two, ie. $0, s_A - s'_A, s_B$, then $g'' \in H_0$ and hence $g = g' + g'' \in H_0 + \bigcap_{i\in B} H_i$. Finally to show that B is forbidden. Given some s_0, consider $g \in G$ which generates s_0, s_A, s_B. By the assumption then $g = g' + g''$ for some $g' \in H_0$ and $g'' \in \bigcap_{i\in B} H_i$ where g' generates $0, s'_A, s'_B$ and g'' generates $s''_0, s''_A, 0$. So taking the sum we find $s''_0 = s_0$ and hence g'' generates $s_0, s''_A, 0$, as required. $\square$

For the second construction we use subgroups $G_0, \dots, G_n$ of a main group G and consider their duals. With Abelian groups we will also want an equivalent idea to that of the dual of a vector space. Hence we consider the Pontryagin dual for groups.

Definition 7.4. The Pontryagin dual of an Abelian group G, denoted $\widehat{G}$, is the group of homomorphisms $G \xrightarrow{\gamma} \mathbb{C}^*$, mapping to the multiplicative group of non-zero complex numbers. Otherwise written $\widehat{G} = Hom(G, \mathbb{C}^*) = \{\gamma : G \to \mathbb{C}^* | \gamma(0) = 1, \gamma(a+b) = \gamma(a)\gamma(b)\}$.

More specifically the mappings $\Pi_i : \widehat{G} \to \widehat{G}_i$, (to the restriction of $\widehat{G}$ to the i-th coordinate), which maps $\Pi(\alpha) = \alpha|_{G_i}$. In this case the code C is a subgroup of the duals $\widehat{G}_0 \times \cdots \times \widehat{G}_n$. The scheme we construct now is actually equivalent to the first and hence its authorised and forbidden sets have the equivalent requirements but on the groups G_i.

Lemma 7.5. *The following equivalences for authorised and forbidden sets hold for the second construction.*

1. $A \in \Gamma_1 \iff G_0 \subset \sum_{i\in A} G_i$,
2. $B \in \Gamma_0 \iff G_0 \cap \sum_{i\in B} G_i = (0)$[1].

Short of giving a proof we give a sketch of the important ideas and leave some ends to be checked. Considering the map $\Pi_i : \widehat{G} \to \widehat{G}_i$ sending α to $\alpha|_{G_i}$, the kernel, $\ker(\Pi_i)$, is the set of all $\alpha : G \to \mathbb{C}$ for which $\alpha|_{G_i} = 1$, the identity element in $\mathbb{C}$. This is exactly the set H_i from the first construction, since taking $g \in G_i = G/H_i$ would give $gH_i = H_i$, ie. the identity in G/H_i. So we have that $H_i = \ker(\Pi_i)$ and hence the first equivalence in the Lemma is proven if we show that

$$\bigcap_{i\in A} H_i \subset H_0 \iff G_0 \subset \sum_{i\in A} G_i,$$

ie. that the requirements are in fact equivalent between the subgroups H_i and their quotients with G. This also holds for the second equivalence which requires that

$$G_0 \cap \sum_{i\in B} G_i = (0) \iff H_0 + \bigcap_{i\in B} H_i = G.$$

We consider a final construction which describes the dual scheme of the second. Again for some subgroups $G_0, \dots, G_n$ of G, but in this case we construct the code as containing any codeword $(s_0, \dots, s_n) \in G_0 \times \cdots \times G_n$ such that $\sum_{i=0}^{n} s_i = 0$. It can be show that the access structure here is exactly the dual of the second

[1]The trivial group with only element 0.

construction, that is, some set of parties A is authorised if and only if its complement is forbidden in the second construction, and vice versa. This construction can be seen as the generalisation of the dual in a linear scheme, and hence duality can be used for verifying the validity of shares.

8. Applications

8.1 Overview

Secret sharing schemes and their adaptations as we have studied in this thesis can be used in a varied range of applications. A protocol to share a hidden secret among multiple parties which can be recovered only by a set of predefined parties can often be useful in real world implementations where it is most often assumed that no one can trust all parties in a system. There are straight-forward use cases for this, like the strengthened password system discussed in the introduction. But more interestingly secret sharing can be used as a sub-protocol to allow a larger operation to take place. In this overview we will consider two examples which show the utility of secret sharing. Both take advantage of the fact that secret sharing does not require a trusted third party and does not assume there is trust between parties.

The first considers the problems of distributed computation and introduces a solution to the average consensus problem with the use of additive secret sharing. The average consensus problem considers a set of parties, or nodes, distributed in a graph system where two nodes can communicate with each other if and only if they are neighbours. Each node has a secret value a_i and the average consensus problem is to find the average of these values $\frac{1}{n}\sum_{i=1}^{n} a_i$, whilst keeping each value secret. The protocol proposed in [22] uses the idea of additive secret sharing. This is a scheme where the threshold is $t = n$ and the shares s_i are uniformly random elements in the domain which altogether sum to the secret, $\sum_{i=1}^{n} s_i = s$. It is easy to see that even with $n-1$ shares an adversary has no way of guessing the final share and hence the secret is uniformly distributed. More interestingly this scheme is also homomorphic, in this case additively since for some shares $s_1, \dots, s_n$ of some secret $\sum_{i=1}^{n} s_i = s$ and another set of shares $s'_1, \dots, s'_n$ for secret $\sum_{i=1}^{n} s'_i = s'$, then if each respective share is summed, when the share sums are put together it finds $\sum_{i=1}^{n}(s_i + s'_i) = s + s'$. Now the protocol runs by first having each node distribute encrypted shares of its secret value to its neighbours. Each node takes note of the encrypted shares and the process repeats until some tracking variable converges. The required average of the values is then computed as an average of the tracking variable, making use of the additively homomorphic property of the secret sharing scheme. Note that this protocol does not require a trusted third party at any time and that the secret value of a node remains secret as long as it has at least one honest neighbour. This sets a strong requirement for an adversary to find the secret value of any node.

The second example we consider is the I Told You Tomorrow protocol from [1] which looks at time-locking secrets over a blockchain. A time-locked secret is one which is fixed and then kept secret for a predetermined amount of time. Commonly this delay is set as the time it takes parties to solve a puzzle or problem which is assumed to take a certain amount of time. In this protocol the goal is to remove the need for this puzzle and use economic incentives to control the release of the secret. Of special interest in this example is the application to the blockchain environment and its use of smart contracts. For a minimal introduction we will only say that a blockchain can be seen as a public ordered list of information blocks which, once linked, cannot be changed, removed or hidden. Blockchains are intrinsically linked to some sort of economic system since they are run by distributed systems which must be incentivised to act in a certain way. A smart contract is the equivalent of a program set in a blockchain, with the notable property that since it cannot be edited once it is submitted, then it can be trusted to run exactly as instructed. In a way this acts as a third party trusted by all parties, since anyone can examine for them-self the program of the smart contract and therefore know exactly how it will run. So the proposed protocol first distributes shares of the secret to the parties and then use a smart contract to apply economic incentives and fines to control the recovery of the secret. Of course it is assumed that the participants are rational, meaning that they choose to act in whichever way will give them the most economic value. So the secret is kept time-locked with the assumption that recovering the secret has a certain value, and the implementation of rewards and fines, for example a fine when a party is caught sending its share ahead of time.

8.2 Schoenmaker implementation

The goal of this section is to consider how the implementation of a PVSS scheme would fair. For this we will consider the Schoenmaker scheme since it in a way stands as a basic scheme from which many of the later PVSS schemes were built. It also relies on the standard DDH assumption in groups which can be implemented without use of any additional libraries. We do note that the Schoenmaker schemes assumes a Random Oracle Model for security, which is certainly not achieved with a program as we will construct. This we overlook since our aim is to demo an implementation of the scheme and not to build a usable version. The program is built in Python and is available on GitHub[2] to be read fully. Our presentation of the program will go as follows, first we consider the basic implementation of the security aspect, then the objects involved in the scheme, that is, the parties, the dealer and the verifier, we consider the functions they each have and finally we see how the demo puts these together for a working implementation of the scheme.

One of the main reasons we considered the Schoenmaker as the ideal scheme to implement is the fact that its security relies on the easy to implement DDH assumption. Other encryption methods also have implemented libraries which we could have used, but the drawback would have been to lose some control on the exact way in which the values are managed. Working with DDH in groups is instead possible to implement directly and in the time-frame allowed was ideal in order to focus on other interesting parts of the implementation. To this end the setup of the encryption is kept very simple. Working in $\mathbb{Z}_p$ for some large prime p is done by simply taking the modulus of any operation on the elements. Recalling the security setup as presented in section 5.1.3, we wish to consider a cyclic group $\mathbb{G}$ of prime order q with two generators g and h. To achieve this we make use of a pair of Sophie Germain and safe primes. In short, a prime q is a **Sophie Germain prime** if $2q + 1$ is also prime, in which case we call the later a **safe prime** and denote it p. The utility of this pair of primes comes from the fact that the multiplicative group $\mathbb{Z}_p$ has order $2q$, and hence there is a large subgroup of prime order q. So to find the required cyclic group all that must be done is to find two elements of $\mathbb{Z}_p$ of order q. Then considering the additive group in $\mathbb{Z}_q$ is equivalent to the multiplicative group constructed by taking the powers of these generators. For example in the demo we have used the following parameters,

Listing 1: parameters for the group

```
    #group parameters
    params = {'mod_p': 1907,
              'order_q': 953,
              'generator_h': 348,
              'generator_g': 483}
```

where 953 is a Sophie Germain prime, $1907 = 953 \times 2 + 1$ a safe prime and 348 and 483 are generators of order 953 in $\mathbb{Z}_p$. Of course this is sufficient for a demo so that the values are comprehensible when printed on the screen, for a serious implementation far larger primes would be used. To conclude this section on the security we note that these parameters are chosen ad-hoc and very much written into the program. That is, they are hard-coded into the program and stored as parameters for every new object at initialisation.

[2]https://github.com/JeremyLvl/TFM

This is a simpler approach than what would be required in a true implementation of this scheme where we assume that the objects involved do not necessarily trust each other.

The objects involved in this implementation are straight-forward, there is a dealer, a party and a verifier. The dealer is initialised with the group parameters and uniformly chooses a non-trivial random secret in $\mathbb{Z}_q$, ie. not 0 or 1. From now on we will refer to such an element of $\alpha \in \mathbb{Z}_q$ as a safe number, since it is for which the Diffie-Hellman assumption says it is infeasible to compute given g^α or h^α. Recall that in the Schoenmaker scheme the secret is random and for the dealer to share a chosen secret it must broadcast an additional value, which the parties can use along with the shared secret to recover the chosen secret. We implement this feature into the voting program, see Section 8.3, where a vote is a chosen secret and another random secret shared, so we do not implement it into the PVSS program.

After the dealer, the party objects are created. In their instantiations they each generate a key pair made up of a safe secret key $sk \in \mathbb{Z}_q$ and its corresponding public key $pk = g^{sk} \in \mathbb{Z}_p$ such that pk is not the trivial element 1. Finally a verifier object is created and simply given the group parameters. Note that any number of verifiers can be used but the representation is the same given only a single one. Also the verifier do not necessarily need to be constructed this early on, ie. a new verifier can be constructed at any time and asked to verify some broadcast information, as we will see.

Once the dealer has the public keys of the parties it may start the share generation and distribution process. Given t the number of parties required for a reconstruction of the secret, the dealer constructs a polynomial *poly* of degree $t-1$ with random coefficients such *poly*(0) is the secret. Note that random values are computed using numpy library's randomness generation.

Listing 2: construction of secret polynomial

```
    #generate random polynomial with p(0)=secret
        coefs = np.random.randint(self.q, size=t-1)
        coefs = np.append(coefs, self.secret)
        poly = np.poly1d(coefs)
```

From this the dealer can compute the share for party i as $pk^{poly(i)} \mod p$.

Listing 3: user share from secret polynomial

```
    #compute each user's share
    idx_enc = {}
    for idx in idx_pk:
        #compute share as pk^{p(i)}
        pk = idx_pk[idx]
        idx_enc[idx] = pow(pk, int(poly(idx)), self.p)
```

Note that each party is indexed by the dealer by the order in which they are constructed, so each has a unique index denoted *idx*. This indexing is used through-out by all objects to uniquely determine the parties. An interesting question that arises when implementing such a scheme into a program is whether or not the public keys can themselves be used as indexes. I believe this is the case since in a full implementation the group $\mathbb{Z}_p$ where the public keys are is assumed to be large enough to have unique public keys. In any case this demo kept the use of indexes for parties and hence the dealer can store values for the parties in a simple dictionary mapping an index to its value, for example *idx_enc* for the encrypted share.

Along with the encrypted shares, the dealer computes commitments for the coefficients *coef* of the polynomial as g^{coef} and a non-interactive zero-knowledge proof that it knows of the unique polynomial *poly* from which the encrypted shares and commitments are computed. All these it stores in a dictionary *broadcast* which will be the information publicly sent out to the parties and for any verifier to check.

Listing 4: proof of knowledge of secret polynomial

```
#elements for proof of knowledge of p(i)
W = np.random.randint(self.q, size=n+1)
idx_A, idx_B = {}, {}
for idx in idx_pk:
    idx_A[idx] = pow(self.g, int(W[idx]), self.p)
    idx_B[idx] = pow(idx_pk[idx], int(W[idx]), self.p)

#gather elements and hash for challenge
chal = self.hash([idx_X, idx_enc, idx_A, idx_B])

#responses to challenge
idx_resp = {}
for idx in idx_pk:
    idx_resp[idx] = (W[idx] - int(poly(idx)) * chal) % self.q

#broadcast proof as challenge and response
broadcast['dealer_proof'] = {'chal': chal,
                             'idx_resp': idx_resp}
```

The proof of knowledge of the polynomial works as follows. Given the set of commitments to the coefficients *coef_commits* indexed by *idx*, the dealer computes the product of these each to the power of idx^j for $j \in \{0, 1, \ldots, t-1\}$. Randomly selecting $w_1, \ldots, w_n \in \mathbb{Z}_q$ it then computes the required commitments as *idx_A* and *idx_B* and finds the challenge by hashing the set of all encrypted shares, the product of *coef_commits* and *idx_A* and *idx_B*. This hashing is simply done by converting the above set of values to bits, using the sha256 function from the hashlib library, and finally recovering an integer from the result of the hash.

Listing 5: hashing function

```
#hash list to integer
def hash(self, to_hash_list):
to_hash = repr(to_hash_list).encode('utf-8')
return int.from_bytes(sha256(to_hash).digest(), 'big')
```

Finally the responses are computed using the polynomial, the challenge and the randomly selected values such that

$$idx\_resp_{idx} = w_{idx} - poly(idx) * challenge \mod q, \tag{17}$$

forming the dealer's proof as the challenge and the responses. Once the shares, commitments and proof are publicly sent out in a broadcast, the verifier is tasked with checking the dealer's proof of distribution. This is done in a similar way to the construction of the proof above. The verifier computes the same product of commitments to the coefficients. For *idx_A* and *idx_B*, since it does not know the set of *n* randomly selected integers in $\mathbb{Z}_q$ it must instead use the given responses and the fact that equation (17) holds if the dealer is honest. It can then hash the recovered set of values and check that the result is equal to the challenge given by the dealer.

With the verification of the dealer's distribution done, the final step is for an authorised set of parties to reconstruct the secret. In this demo we simply hard-code a single authorised set of sufficiently many parties but the generalisation to any other threshold method is straight-forward. The reconstruction method would not change and it would be a matter of starting the process for any authorised set of parties which wants to.

The first step in the reconstruction is for each party in the reconstruction set to decrypt the share they receive as their own from the dealer. As explained in the scheme this is done by computing the encrypted share to the power of the inverse of the secret key in $\mathbb{Z}_q$, as shown below.

Listing 6: Share decryption

```
    #decrypt share using inverse of sk in Z/qZ
    sk_inv = pow(self.sk, -1, self.q)
    self.share = pow(self.enc_share, sk_inv, self.p)
```

Then the party must broadcast its share and a non-interactive zero-knowledge proof of correct decryption. The latter follows a similar approach to the dealer's proof except that only one value needs to be proved accurate in this case. The proof and verification follow the standard DLEQ idea and we do not go further into either. We however do note that the decrypted shares are shared publicly between the reconstructing parties without re-encryption and hence anyone following the transcript of public information could recover the secret. Although it is possible to re-encrypt the shares for the reconstruction step, this adds many additional requirements since the shares must be encrypted for each reconstruction party and must each be accompanied by a proof of correct encryption. For this demo we choose to follow the Schoenmaker scheme directly and have parties publicly send their shares. We will notice that in the later demo of a voting system this is no longer a problem since the shares themselves are not shared directly but instead compounded together. That is, what is sent by the parties, which play the role of the talliers, is a share for the total number of votes, hence publicly sharing poses no problem.

Assuming that each reconstructing party has now received the valid shares from the others, then the final recovery of the secret can be carried out by each of these.

Listing 7: Secret recovery

```
    def secret_prod(self, idx_shares):
        prod = 1
        for idx in idx_shares:
            #compute lambda from indices
            indices = list(idx_shares.keys())
            indices.remove(idx)
            lambdas = [Fraction(j,j-idx) for j in indices]
            L = m.prod(lambdas)

            #get share as base of power
            share = idx_shares[idx]
            base = share

            #find root if lambda is fraction
            if L.denominator != 1:
                denom_inv = pow(L.denominator, -1, self.q)
                base = pow(share, denom_inv, self.p)

            #multiply product
            prod = (prod * pow(base, L.numerator, self.p)) % self.p

        #compute final secret
        return int(prod % self.p)
```

This follows from the standard Shamir reconstruction, but some care must be taken when manipulating values in the group $\mathbb{Z}_p$. For each party's *idx*, share pair the lambda coefficient $\lambda = \prod_{j \neq idx} \frac{j}{j - idx}$ is computed. If this is a fraction then the power $share^{\lambda}$ must be carried out in two operations. That is, the inverse of the denominator is found and the base of the calculation is updated to be the share to this power. Finally the value in the product is the base to the numerator of λ, and the final product is in fact the secret.

Having seen how the PVSS scheme works we can consider a small demo which puts it all together. First the user selects the number of parties and gives the indices for an authorised set of parties which will carry out the reconstruction. The dealer is initialised with some random secret to be shared and the parties are initialised with a secret key, public key pair. Of course in the demo we print the secret key for completeness but the secret key is kept private and is not shared with anyone. The encrypted shares for each party are broadcast along with a proof of correct distribution from the dealer. The verifier then validates the correctness of this proof before the parties decrypt their own share. For the reconstruction, the authorised parties broadcast their decrypted share and a proof of having correctly decrypted it. This is checked by the verifier and once all the verified shares are sent, each authorised party can then reconstruct the original secret.

Listing 8: Demo of PVSS scheme

```
number of parties: 4
authorised parties (subset of [1,2,...,4]): 2,4

dealer initialised with shared secret 1405

party initialised with sk: 499, pk: 487
party initialised with sk: 456, pk: 280
party initialised with sk: 291, pk: 258
party initialised with sk: 571, pk: 1189

share for party 1 encrypted as 1540
share for party 2 encrypted as 196
share for party 3 encrypted as 282
share for party 4 encrypted as 746

dealer broadcasts proof of distribution
```

```
verifier has validated  the dealer's distribution

party 1 decrypted own share to 1515
party 2 decrypted own share to 636
party 3 decrypted own share to 1887
party 4 decrypted own share to 1200

party 2 broadcast decrypted share as 636 and proof of decryption
verifier has validated party 2's share

party 4 received encrypted share from 2 as 636
party 4 recovered secret 1405

party 4 broadcast decrypted share as 1200 and proof of decryption
verifier has validated party 4's share

party 2 received encrypted share from 4 as 1200
party 2 recovered secret 1405
```

8.3 A voting scheme

This implementation of Schoenmaker's publicly verifiable secret sharing scheme can be adapted to make a voting scheme. This voting scheme will allow for any number of voters to privately cast a 0 or 1 vote such that the total number of each votes is found without having to decrypt any particular vote. This is possible due to the homomorphic property of the Schoenmaker scheme as described in section 5.1.3. The objects in this scheme are as follows. A voter is a dealer from the PVSS scheme to which we give an index, since in the PVSS scheme there was exactly one dealer, whereas in this voting scheme we will require many voters. Naturally the voter will share its vote like a dealer would share a secret in the PVSS scheme. A tallier is a party in the PVSS scheme, although unlike a party which immediately recovers the secret given the appropriate shares, it will instead combine the shares to construct a share for the sum of the secrets. Finally there are two verifier objects, the PVSS verifier which checks the validity of the shares sent by the voters, and a new voting verifier which validates that the vote is either 0 or 1. Of course it would be tempting for a voter to vote a larger number than 1 or a negative number and bias the total, which would otherwise go unnoticed since the vote is broken into shares.

Once all the objects are created, the voters can cast their votes as follows. The vote is either 0 or 1, for the purpose of the demo it is randomly selected. The secret that is shared is actually not the vote itself, but a safe secret in $\mathbb{Z}_q$ and they are put together such that the tallier decrypts the sum of votes. The threshold for the number of parties required to reconstruct the secret is set to be the number of talliers, although it could certainly be envisaged that for a larger election we would want to only use a few talliers for each vote. That is, each voter would either have its assigned subset of talliers or would set some threshold number of talliers required to recover their vote. The voter therefore commits to its vote U and some secret C and must prove that they are valid, ie. $C = g^s$ for some secret $s \in \mathbb{Z}_q$ and $U = g^{s+v}$ for some vote $v \in \{0, 1\}$. The proof works as follows, where the challenge is a hashing of the values to imitate the verifier sending a random challenge.

Listing 9: Proof of valid vote

```
#secret and vote
C = pow(self.g, self.secret, self.p)
U = pow(self.h, (self.secret+v)%self.q, self.p)

broadcast['secret_commits'] = {'secret': C,
                               'vote': U}

#proof of valid vote
w = randint(3, self.q)
r = [0,0]
d = [0,0]
a = [0,0]
b = [0,0]
r[1-v] = randint(3, self.q)
d[1-v] = randint(3, self.q)

a[v] = pow(self.g, w, self.p)
b[v] = pow(self.h, w, self.p)
a[1-v] = (pow(self.g, r[1-v], self.p)
          * pow(C, d[1-v], self.p)) % self.p
frac_1 = pow(self.h, v-1, self.p)
frac_2 = (U * frac_1) % self.p
b[1-v] = (pow(self.h, r[1-v], self.p)
          * pow(frac_2, d[1-v], self.p)) % self.p

chal = self.hash([a,b])%self.q

d[v] = (chal - d[1-v]) % self.q
r[v] = (w - self.secret * d[v]) % self.q

broadcast['voter_proof'] = {'commit': [a,b],
                            'chal': chal,
                            'resp': [d,r]}
```

In short, the values $w, r_{1-v}, d_{1-v} \in_R \mathbb{Z}_q$ are chosen at random by the voter who sets as commitments

$$\begin{cases} a_v = g^w, \quad a_{1-v} = g^{r_{1-v}} C^{d_{1-v}}, \\ b_v = h^w, \quad b_{1-v} = h^{r_{1-v}} \left(\frac{U}{h^{1-v}}\right)^{d_{1-v}}. \end{cases}$$

Hashing the values of a_0, a_1, b_0, b_1 to form a challenge $c \in \mathbb{Z}_q$ the voter can then set the responses as

$$\begin{cases} d_v = c - d_{1-v} \mod q, \\ r_v = w - sd_v \mod q. \end{cases}$$

Finally the verifier can check

$$\begin{cases} c = d_0 + d_1 \mod q, \\ a_0 = g^{r_0} C^{d_0}, \quad a_1 = g^{r_1} C^{d_1}, \\ b_0 = h^{r_0} U^{d_0}, \quad b_1 = h^{r_1} \left(\frac{U}{h}\right)^{d_1}. \end{cases}$$

It can easily be checked that whenever $v \in \{0, 1\}$ then the verifier's check will hold. For example with $v = 0$, then the voter's response sets $d_0 = c - d_1 \mod q$ and $r_0 = w - sc + sd_1 \mod q$ and hence checking the verifier's second equation becomes

$$a_0 = g^w = g^W g^{-sc+sd_1} (g^s)^{c-d_1} = g^{r_0} C^{d_0}.$$

The voting verification is carried out, along with the same verification of the dealer's distribution, this for the shares sent out by the voter. Next the talliers sum the shares they have received from each voter and compute their product. Once decrypted, this forms for each tallier a share for the total number of 1 votes.

Listing 10: Share of sum of votes from shares of votes

```
    #sum received shares
    prod = 1
    for share in shares:
        prod = (prod * share) % self.p

    #decrypt share sum
    sk_inv = pow(self.sk, -1, self.q)
    share_sum = pow(prod, sk_inv, self.p)
```

Finally the talliers count the votes. This first involves reconstructing the secret as in the PVSS scheme, which recovers $C = g^s$. Then the voter's values of $U = g^{s+v}$ are combined in a product to find the vote and secret sum. Finally the sum of votes can be found by dividing the latter by the former.

Listing 11: Computing the sum of votes

```
    #gather share sums to recover sum of secrets
    secret_sum = self.secret_prod(jdx_votes)

    #compute product of U's to find sum of secrets and votes
    U_prod = 1
    for U in list(idx_U.values()):
        U_prod = (U_prod * U) % self.p

    #compute sum of votes
    secret_sum_inv = pow(secret_sum, -1, self.p)
    vote_sum = (U_prod * secret_sum_inv) % self.p
```

Note that when we speak here of a sum of votes S, what we really have is h^S, hence it remains to find the discrete log to find the true number of votes.

Lastly we consider a full run-through of the voting demo with 3 voters and two talliers. In this demo we let the vote be random between 0 and 1, of course the real vote should be entered by the party through a machine they trust. As explained earlier, each voter splits their vote into shares, encrypts them and broadcasts a proof of having done so correctly. Once both the voting verifier and the PVSS verifier validate each vote and its correct distribution, the tallier can put together their assigned shares to find a share for the sum of the votes, and hence together they find the sum of votes.

Listing 12: Demo of voting scheme

```
Number of voters: 3
voter initialised with shared secret 1000
voter initialised with shared secret 1827
voter initialised with shared secret 1385

number of talliers: 2
tallier initialised with sk: 568, pk: 1633
tallier initialised with sk: 950, pk: 1342

voter 1 votes 0
share for tallier 1 encrypted as 1902
share for tallier 2 encrypted as 610
voter 1 broadcasts proof of distribution

voter 2 votes 1
share for tallier 1 encrypted as 880
share for tallier 2 encrypted as 159
voter 2 broadcasts proof of distribution

voter 3 votes 1
share for tallier 1 encrypted as 529
share for tallier 2 encrypted as 1184
voter 3 broadcasts proof of distribution

voting verifier has validated voter 1's vote
PVSS verifier has validated voter 1's distribution

voting verifier has validated voter 2's vote
PVSS verifier has validated voter 2's distribution

voting verifier has validated voter 3's vote
PVSS verifier has validated voter 3's distribution

tallier 1 counted 2 votes in favour, out of 3 total
tallier 2 counted 2 votes in favour, out of 3 total
```

We note that this implementation does not take into account votes and commitments entered by the parties which do not validate. In the simple case of the implementation we set all parties to send correct shares and their respective commitments. Although it is an interesting question as to what should be done about a vote that does not validate, and what is considered not being valid. Of course a party could blame a verifier for cheating, in which case multiple verifiers should be taken to the vote, since the probability that so many have been corrupt in the same direction is decreasing. In the case of a publicly visible code for the voting platform, it may be best to only allow the party to enter their vote and let the program take care of construction of the commitment and the rest. Anyone who would like to check the program is honest may check the public code, but overall the commitments should always be valid. Of course this should be a problem for only a very small proportion of the votes and as many paper voting systems currently do, they could simply be counted as ‘unrecognised’.

References

[1] Enrico Bacis, Dario Facchinetti, Marco Guarnieri, Marco Rosa, Matthew Rossi, and Stefano Paraboschi. I told you tomorrow: Practical time-locked secrets using smart contracts. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*, ARES 21, New York, NY, USA, 2021. Association for Computing Machinery.

[2] Amos Beimel. Secret-sharing schemes: A survey. pages 11–46, 05 2011.

[3] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 1–10, New York, NY, USA, 1988. Association for Computing Machinery.

[4] Josh Benaloh. Secret sharing homomorphisms: Keeping shares of a secret sharing. In *CRYPTO*, 1986.

[5] G. R. Blakley. Safeguarding cryptographic keys. In *1979 International Workshop on Managing Requirements Knowledge (MARK)*, pages 313–318, 1979.

[6] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. 2015.

[7] Ignacio Cascudo and Bernardo David. Scrape: Scalable randomness attested by public entities. Cryptology ePrint Archive, Paper 2017/216, 2017. `https://eprint.iacr.org/2017/216`.

[8] Ignacio Cascudo and Bernardo David. Albatross: publicly attestable batched randomness based on secret sharing. Cryptology ePrint Archive, Paper 2020/644, 2020. `https://eprint.iacr.org/2020/644`.

[9] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. Yolo yoso: Fast and simple encryption and secret sharing in the yoso model. Cryptology ePrint Archive, Paper 2022/242, 2022. `https://eprint.iacr.org/2022/242`.

[10] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ecdsa from hash proof systems and efficient instantiations. In *Advances in Cryptology – CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III*, page 191–221, Berlin, Heidelberg, 2019. Springer-Verlag.

[11] Guilhem Castagnos and Fabien Laguillaumie. Linearly Homomorphic Encryption from DDH. In *The Cryptographer's Track at the RSA Conference 2015*, number 9048 in Topics in Cryptology — CT-RSA 2015, San Francisco, United States, April 2015.

[12] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Practical Fully Secure Unrestricted Inner Product Functional Encryption modulo p. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security*, volume LNCS of *Advances in Cryptology – ASIACRYPT 2018*, pages 733–764, Brisbane, Australia, December 2018.

[13] Anirudh Chandramouli, Ashish Choudhury, and Arpita Patra. A survey on perfectly-secure verifiable secret-sharing. Cryptology ePrint Archive, Paper 2021/445, 2021. `https://eprint.iacr.org/2021/445`.

[14] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.

[15] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395, 1985.

[16] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438, 1987.

[17] Eiichiro Fujisaki and Tatsuaki Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1998.

[18] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, STOC '01, page 580–589, New York, NY, USA, 2001. Association for Computing Machinery.

[19] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. Cryptology ePrint Archive, Paper 2021/1397, 2021. `https://eprint.iacr.org/2021/1397`.

[20] Somayeh Heidarvand and Jorge L. Villar. Public verifiability from pairings in secret sharing schemes. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, pages 294–308, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[21] Amir Jafari and Shahram Khazaei. On abelian and homomorphic secret sharing schemes. Cryptology ePrint Archive, Paper 2019/575, 2019. `https://eprint.iacr.org/2019/575`.

[22] Qiongxiu Li, Ignacio Cascudo, and Mads Graesbøll Christensen. Privacy-preserving distributed average consensus based on additive secret sharing. In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5, 2019.

[23] R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Commun. ACM*, 24(9):583–584, sep 1981.

[24] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Application of Cryptographic Techniques*, 1999.

[25] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of The Society for Industrial and Applied Mathematics*, 8:300–304, 1960.

[26] Alexandre Ruiz and Jorge L. Villar. Publicly verifiable secret sharing from paillier's cryptosystem. In Christopher Wulf, Stefan Lucks, and Po-Wah Yau, editors, *WEWoRC 2005 – Western European Workshop on Research in Cryptology*, pages 98–108, Bonn, 2005. Gesellschaft für Informatik e.V.

[27] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, page 148–164, Berlin, Heidelberg, 1999. Springer-Verlag.

[28] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.