CHAPTER 19

Secret Sharing Schemes

Chapter Goals

- To introduce the notion of secret sharing schemes.
- To give some simple examples of general access structures.
- To present Shamir's scheme, including how to recover the secret in the presence of active adversaries.
- To show the link between Shamir's secret sharing and Reed–Solomon codes.
- As an application we show how secret sharing can provide more security for a certificate authority, via distributed RSA signature generation.

19.1. Access Structures

Suppose you have a secret s which you wish to share amongst a set $\mathcal{P}$ of n parties. You would like certain subsets of the n parties to recover the secret but not others. The classic scenario might be that s is a nuclear launch code and you have four people, the president, the vice-president, the secretary of state and a general in a missile silo. You do not want the general to be able to launch the missile without the president agreeing, but to maintain deterrence you would like, in the case that the president has been eliminated, that the vice-president, the secretary of state and the general can agree to launch the missile. If we label the four parties as P, V, S and G, for president, vice-president, secretary of state and general, then we would like the following sets of people to be able to launch the missile

$$\{P, G\} \text{ and } \{V, S, G\},$$

but no smaller sets. It is this problem which secret sharing is designed to deal with, however the applications are more widespread than might at first appear.

To each party we distribute some information called a *share*. For a party A we will let s_A denote the secret share which they hold. In the example above there are four such shares: s_P, s_V, s_S and s_G. Then if the required parties come together we would like an algorithm which combines their relevant shares into the secret s. But if the wrong set of parties come together they should learn no information about s.

Before introducing schemes to perform secret sharing we first need to introduce the notion of an access structure. Any subset of parties who can recover the secret will be called a *qualifying set*, whilst the set of all qualifying sets will be called an *access structure*. So in the example above we have that the two sets

$$\{P, G\} \text{ and } \{V, S, G\}$$

are qualifying sets. However, clearly any set containing such a qualifying set is also a qualifying set. Thus

$$\{P, G, V\},\ \{P, G, S\} \text{ and } \{P, V, G, S\}$$

are also qualifying sets. Hence, there are five sets in the access structure. For any set in the access structure, if we have the set of shares for that set we would like to be able to reconstruct the secret.

N.P. Smart, *Cryptography Made Simple*, Information Security and Cryptography, DOI 10.1007/978-3-319-21936-3_19

Definition 19.1. *A monotone structure on a set $\mathcal{P}$ is a collection Γ of subsets of $\mathcal{P}$ such that*

- $\mathcal{P} \in \Gamma$.
- *If there is a set $A \in \Gamma$ and a set B such that $A \subset B \subset \mathcal{P}$ then $B \in \Gamma$.*

Thus in the above example the access structure is monotone. This is a property which will hold for all access structures of all secret sharing schemes. For a monotone structure we note that the sets in Γ come in chains, $A \subset B \subset C \subset \mathcal{P}$. We shall call the sets which form the start of a chain the *minimal qualifying sets*. The set of all such minimal qualifying sets for an access structure Γ we shall denote by $m(\Gamma)$. We can now give a very informal definition of what we mean by a secret sharing scheme:

Definition 19.2. *A secret sharing scheme for a monotone access structure Γ over a set of parties $\mathcal{P}$ with respect to a space of secrets S is a pair of algorithms called* `Share` *and* `Recombine` *with the following properties:*

- `Share`(s, Γ) *takes a secret $s \in S$ and a monotone access structure and determines a value s_A for every $A \in \mathcal{P}$. The value s_A is called A's share of the secret.*
- `Recombine`(H) *takes a set $H_{\mathcal{O}}$ of shares for some subset $\mathcal{O}$ of $\mathcal{P}$, i.e.*

$$H_{\mathcal{O}} = \{s_O : O \in \mathcal{O}\}.$$

If $\mathcal{O} \in \Gamma$ then this should return the secret s, otherwise it should return nothing.

A secret sharing scheme is considered to be secure if no infinitely powerful adversary can learn anything about the underlying secret without having access to the shares of a qualifying set. Actually such schemes are said to be information-theoretically secure, but since most secret sharing schemes in the literature are information-theoretically secure we shall just call such schemes *secure*.

In this chapter we will consider two running examples of monotone access structures, so as to illustrate the schemes. Both will be on sets of four elements: The first is from the example above where we have $\mathcal{P} = \{P, V, S, G\}$ and

$$\Gamma = \{\{P, G\}, \{V, S, G\}, \{P, G, V\}, \{P, G, S\}, \{P, V, G, S\}\}.$$

The set of minimal qualifying sets is given by

$$m(\Gamma) = \{\{P, G\}, \{V, S, G\}\}.$$

The second example we shall define over the set of parties $\mathcal{P} = \{A, B, C, D\}$, with access structure

$$\begin{aligned}\Gamma = \{\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\},\\ \{A, B, C\}, \{A, B, D\}, \{B, C, D\}, \{A, B, C, D\}\}.\end{aligned}$$

The set of minimal qualifying sets is given by

$$m(\Gamma) = \{\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\}\}.$$

This last access structure is interesting because it represents a common form of threshold access structure. Notice that, in this access structure, we require that any two out of the four parties should be able to recover the secret. We call such a scheme a 2-out-of-4 threshold access structure.

One way of looking at such access structures is via a Boolean formulae. Consider the set $m(\Gamma)$ of minimal qualifying sets and define the formula:

$$\bigvee_{\mathcal{O} \in m(\Gamma)} \left(\bigwedge_{O \in \mathcal{O}} O \right).$$

So in our first example above the formula becomes

$$(P \wedge G) \vee (V \wedge S \wedge G).$$

Reading this formula out, with $\wedge$ being "and", and $\vee$ being "or", we see that we can reconstruct the secret if we have access to the secret shares of

$$\text{(P and G) or (V and S and G).}$$

Notice how the formula is in disjunctive normal form (DNF), i.e. an "or" of a set of "and" clauses. We shall use this representation below to construct a secret sharing scheme for any access structure.

19.2. General Secret Sharing

We now turn to two methods for constructing secret sharing schemes for arbitrary monotone access structures. They are highly inefficient for all but the simplest access structures but they do show that one can cope with an arbitrary access structure. We assume that the space of secrets S is essentially the set of bit strings of length n bits. In both examples we let $s \in S$ denote the secret which we are trying to share.

19.2.1. Ito–Nishizeki–Saito Secret Sharing: Our first secret sharing scheme makes use of the DNF Boolean formula we presented above. In some sense every "or" gets converted into a concatenation operation and every "and" gets converted into a $\oplus$ operation. This can at first sight seem slightly counterintuitive, since usually we associate "and" with multiplication and "or" with addition.

The sharing algorithm works as follows. For every minimal qualifying set $\mathcal{O} \in m(\Gamma)$, we generate shares $s_i \in S$, for $1 \leq i \leq l$, at random, where $l = |\mathcal{O}|$ such that $s_1 \oplus \cdots \oplus s_l = s$. Then a party A is given a share s_i if A occurs at position i in the set $\mathcal{O}$.

Example: Recall we have the formula

$$\text{(P and G) or (V and S and G).}$$

We generate five elements s_i from S such that

$$\begin{aligned} s &= s_1 \oplus s_2, \\ &= s_3 \oplus s_4 \oplus s_5. \end{aligned}$$

The four shares are then defined to be:

$$\begin{aligned} s_P &\leftarrow s_1, \\ s_V &\leftarrow s_3, \\ s_S &\leftarrow s_4, \\ s_G &\leftarrow s_2 \| s_5. \end{aligned}$$

You should check that, given this sharing, any qualifying set can recover the secret, and only the qualifying sets can recover the secret. Notice that party G needs to hold two times more data than the size of the secret. Thus this scheme in this case is not efficient. Ideally we would like the parties to only hold the equivalent of n bits of information each so as to recover a secret of n bits.

Example: Now our formula is given by

$$\text{(A and B) or (A and C) or (A and D) or (B and C) or (B and D) or (C and D).}$$

We now generate twelve elements s_i from S, one for each of the distinct terms in the formula above, such that

$$\begin{aligned} s &= s_1 \oplus s_2, \\ &= s_3 \oplus s_4, \\ &= s_5 \oplus s_6, \\ &= s_7 \oplus s_8, \\ &= s_9 \oplus s_{10}, \\ &= s_{11} \oplus s_{12}. \end{aligned}$$

The four shares are then defined to be:

$$\begin{aligned} s_A &\leftarrow s_1 \| s_3 \| s_5, \\ s_B &\leftarrow s_2 \| s_7 \| s_9, \\ s_C &\leftarrow s_4 \| s_8 \| s_{11}, \\ s_D &\leftarrow s_6 \| s_{10} \| s_{12}. \end{aligned}$$

You should again check that, given this sharing, any qualifying set and only the qualifying sets can recover the secret. We see that in this case every share contains three times more information than the underlying secret.

19.2.2. Replicated Secret Sharing: The above is not the only scheme for general access structures. Here we present another one, called the replicated secret sharing scheme. In this scheme we first create the sets of all maximal non-qualifying sets; these are the sets of all parties such that if you add a single new party to each set you will obtain a qualifying set. If we label these sets $A_1, \ldots, A_t$, we then form their set-theoretic complements, i.e. $B_i = \mathcal{P} \setminus A_i$. Then a set of secret shares s_i is then generated, one for each set B_i, so that

$$s = s_1 \oplus \cdots \oplus s_t.$$

A party is given the share s_i if it is contained in the set B_i.

Example: The sets of maximal non-qualifying sets for our first example are

$$A_1 = \{P, V, S\},\ A_2 = \{V, G\} \text{ and } A_3 = \{S, G\}.$$

Forming their complements we obtain the sets

$$B_1 = \{G\},\ B_2 = \{P, S\} \text{ and } B_3 = \{P, V\}.$$

We generate three shares s_1, s_2 and s_3 such that $s = s_1 \oplus s_2 \oplus s_3$ and then define the shares as

$$\begin{aligned} s_P &\leftarrow s_2 \| s_3, \\ s_V &\leftarrow s_3, \\ s_S &\leftarrow s_2, \\ s_G &\leftarrow s_1. \end{aligned}$$

Again we can check that only the qualifying sets can recover the secret.

Example: For the 2-out-of-4 threshold access structure we obtain the following maximal non-qualifying sets

$$A_1 = \{A\},\ A_2 = \{B\},\ A_3 = \{C\} \text{ and } A_4 = \{D\}.$$

On forming their complements we obtain

$$B_1 = \{B, C, D\},\ B_2 = \{A, C, D\},\ B_3 = \{A, B, D\} \text{ and } B_4 = \{A, B, C\}.$$

We form the four shares such that $s = s_1 \oplus s_2 \oplus s_3 \oplus s_4$ and set

$$\begin{aligned} s_A &\leftarrow s_2 \| s_3 \| s_4, \\ s_B &\leftarrow s_1 \| s_3 \| s_4, \\ s_C &\leftarrow s_1 \| s_2 \| s_4, \\ s_D &\leftarrow s_1 \| s_2 \| s_3. \end{aligned}$$

Whilst the above two constructions provide a mechanism to construct a secret sharing scheme for any monotone access structure, they appear to be very inefficient. In particular for the threshold access structure they are especially bad, especially as the number of parties increases. In the rest of this chapter we will examine a very efficient mechanism for threshold secret sharing due to Shamir, called Shamir secret sharing. This secret sharing scheme is itself based on the ideas behind certain error-correcting codes, called Reed–Solomon codes. So we will first have a little digression into coding theory.

19.3. Reed–Solomon Codes

An error-correcting code is a mechanism to transmit data from A to B such that any errors which occur during transmission, for example due to noise, can be corrected. They are found in many areas of electronics: they are the thing which makes your CD/DVD resistant to minor scratches; they make sure that RAM chips preserve your data correctly; they are used for communication between Earth and satellites or deep space probes.

A simpler problem is that of error detection. Here one is only interested in whether the data have been altered or not. A particularly important distinction to make between the area of coding theory and cryptography is that in coding theory one can select simpler mechanisms to detect errors. This is because in coding theory the assumption is that the errors are introduced by random noise, whereas in cryptography any errors are thought to be actively inserted by an adversary. Thus in coding theory error detection mechanisms can be very simple, whereas in cryptography we have to resort to complex mechanisms such as MACs and digital signatures.

Error correction on the other hand is not only interested in detecting errors, it also wants to correct those errors. Clearly one cannot correct all errors, but it would be nice to correct a certain number. A classic way of forming error-correcting codes is via Reed–Solomon codes. In coding theory such codes are usually presented over a finite field of characteristic two. However, we are interested in the general case and so we will be using a code over $\mathbb{F}_q$, for a prime power q. Each code-word is a vector over $\mathbb{F}_q$, with the vector length being called the length of the code-word. The set of all valid code-words forms the code, there is a mapping from the set of valid code-words to the data one wants to transmit. The invalid code-words, i.e. vectors which are not in the code, are used to perform error detection and correction.

To define a Reed–Solomon code we also require two integer parameters, n and t. The value n defines the length of each code-word, whilst the number t is related to the number of errors we can correct; indeed we will be able to correct $(n-t-1)/2$ errors. We also define a set $\mathbb{X} \subset \mathbb{F}_q$ of size n. If the characteristic of $\mathbb{F}_q$ is larger than n then we can select $\mathbb{X} = \{1, 2, \ldots, n\}$, although any subset of $\mathbb{F}_q$ will do. For our application to Shamir secret sharing later on we will assume that $0 \notin \mathbb{X}$.

Consider the set of polynomials of degree less than or equal to t over the field $\mathbb{F}_q$.

$$\mathbb{P} = \left\{ f_0 + f_1 \cdot X + \ldots + f_t \cdot X^t : f_i \in \mathbb{F}_q \right\}.$$

The set $\mathbb{P}$ represents the number of code-words in our code, i.e. the number of different data items which we can transmit in any given block, and hence this number is equal to q^{t+1}. To transmit some data, in a set of size $|\mathbb{P}|$, we first encode it as an element of $\mathbb{P}$ and then we translate it into a

code-word. To create the actual code-word we evaluate the polynomial at all elements in $\mathbb{X}$. Hence, the set of *actual* code-words is given by

$$\mathcal{C} = \{(f(x_1), \ldots, f(x_n)) : f \in \mathbb{P}, x_i \in S\}.$$

The size (in bits) of a code-word is then $n \cdot \log_2 q$. So we require $n \cdot \log_2 q$ bits to represent $(t+1) \cdot \log_2 q$ bits of information.

Example: As an example consider the Reed–Solomon code with parameters $q = 101$, $n = 7$, $t = 2$ and $\mathbb{X} = \{1, 2, 3, 4, 5, 6, 7\}$. Suppose our "data", which we represent by an element of $\mathbb{P}$, is given by the polynomial

$$f = 20 + 57 \cdot X + 68 \cdot X^2.$$

To transmit this data we compute $f(i) \pmod q$ for $i = 1, \ldots, 7$, to obtain the code-word

$$c = (44, 2, 96, 23, 86, 83, 14).$$

This code-word can now be transmitted or stored.

19.3.1. Data Recovery: At some point the code-word will need to be converted back into the data. In other words we have to recover the polynomial in $\mathbb{P}$ from the set of points at which it was evaluated, i.e. the vector of values in $\mathcal{C}$. We will first deal with the simple case and assume that no errors have occurred. The receiver is given the data $c = (c_1, \ldots, c_n)$ but has no idea as to the underlying polynomial f. Thus from the receiver's perspective he wishes to find the f_i such that

$$f = \sum_{j=0}^{t} f_j \cdot X^j.$$

It is well known, from high school, that a polynomial of degree at most t is determined completely by its values at $t+1$ points. So as long as $t < n$ we can recover f when no errors occur; the question is how?

First note that the receiver can generate n linear equations via

$$c_i = f(x_i) \text{ for } x_i \in \mathbb{X}.$$

In other words he has the system of equations:

$$\begin{aligned} c_1 &= f_0 + f_1 \cdot x_1 + \cdots + f_t \cdot x_1^t, \\ &\vdots \qquad\qquad \vdots \\ c_n &= f_0 + f_1 \cdot x_n + \cdots + f_t \cdot x_n^t. \end{aligned}$$

So by solving this system of equations over the field $\mathbb{F}_q$ we can recover the polynomial and hence the data.

Actually the polynomial f can be recovered without solving the linear system, via the use of Lagrange interpolation. Suppose we first compute the polynomials

$$\delta_i(X) \leftarrow \prod_{x_j \in \mathbb{X}, j \neq i} \frac{X - x_j}{x_i - x_j}, \ 1 \leq i \leq n.$$

Note that we have the following properties, for all i,

- $\delta_i(x_i) = 1$.
- $\delta_i(x_j) = 0$, if $i \neq j$.
- $\deg \delta_i(X) = n - 1$.

Lagrange interpolation takes the values c_i and computes

$$f(X) \leftarrow \sum_{i=1}^{n} c_i \cdot \delta_i(X).$$

The three properties above of the polynomials $\delta_i(X)$ translate into the following facts about $f(X)$:

- $f(x_i) = c_i$ for all i.
- $\deg f(X) \leq n - 1$.

Hence, Lagrange interpolation finds the unique polynomial which interpolates the n elements in the code-word.

19.3.2. Error Detection: We see that by using Lagrange interpolation on the code-word we will recover a polynomial of degree t when there are no errors, but when there are errors in the received code-word we are unlikely to obtain a valid polynomial, i.e. an element of $\mathbb{P}$. Hence, we instantly have an error-detection algorithm: we apply the method above to recover the polynomial, assuming it is a valid code-word, then if the resulting polynomial has degree greater than t we can conclude that an error has occured.

Returning to our example parameters above, suppose the following code-word was received

$$c = (44, 2, 25, 23, 86, 83, 14).$$

In other words it is equal to the sent code-word except in the third position where a 96 has been replaced by a 25. We compute once and for all the polynomials, modulo $q = 101$,

$$\begin{aligned}
\delta_1(X) &= 70 \cdot X^6 + 29 \cdot X^5 + 46 \cdot X^4 + 4 \cdot X^3 + 43 \cdot X^2 + 4 \cdot X + 7,\\
\delta_2(X) &= 85 \cdot X^6 + 12 \cdot X^5 + 23 \cdot X^4 + 96 \cdot X^3 + 59 \cdot X^2 + 49 \cdot X + 80,\\
\delta_3(X) &= 40 \cdot X^6 + 10 \cdot X^5 + 83 \cdot X^4 + 23 \cdot X^3 + 48 \cdot X^2 + 64 \cdot X + 35,\\
\delta_4(X) &= 14 \cdot X^6 + 68 \cdot X^5 + 33 \cdot X^4 + 63 \cdot X^3 + 78 \cdot X^2 + 82 \cdot X + 66,\\
\delta_5(X) &= 40 \cdot X^6 + 90 \cdot X^5 + 99 \cdot X^4 + 67 \cdot X^3 + 11 \cdot X^2 + 76 \cdot X + 21,\\
\delta_6(X) &= 85 \cdot X^6 + 49 \cdot X^5 + 91 \cdot X^4 + 91 \cdot X^3 + 9 \cdot X^2 + 86 \cdot X + 94,\\
\delta_7(X) &= 70 \cdot X^6 + 45 \cdot X^5 + 29 \cdot X^4 + 60 \cdot X^3 + 55 \cdot X^2 + 43 \cdot X + 1.
\end{aligned}$$

The receiver now tries to recover the sent polynomial given the data he has received. He obtains

$$\begin{aligned}
f(X) &\leftarrow 44 \cdot \delta_1(X) + \cdots + 14 \cdot \delta_7(X)\\
&= 60 + 58 \cdot X + 94 \cdot X^2 + 84 \cdot X^3 + 66 \cdot X^4 + 98 \cdot X^5 + 89 \cdot X^6.
\end{aligned}$$

But this is a polynomial of degree six and not of degree $t = 2$. Hence, the receiver knows that there is at least one error in the code word that he has received. He just does not know which position is in error, nor what its actual value should be.

19.3.3. Error Correction: The intuition behind error correction is the following. Consider a polynomial of degree three over the reals evaluated at seven points, such as that in Figure 19.1. Clearly there is only one cubic curve which interpolates all of the points, since we have specified seven of them and we only need four such points to define a cubic curve. Now suppose one of these evaluations is given in error, for example the point at $x = 3$, as in Figure 19.2. We see that we still have six points on the cubic curve, and so there is a unique cubic curve passing through these six valid points. However, suppose we took a different set of six points, i.e. five valid ones and one incorrect one. It is then highly likely that the curve which goes through the second set of six points would not be cubic. In other words because we have far more valid points than we need to determine the cubic curve, we are able to recover it.

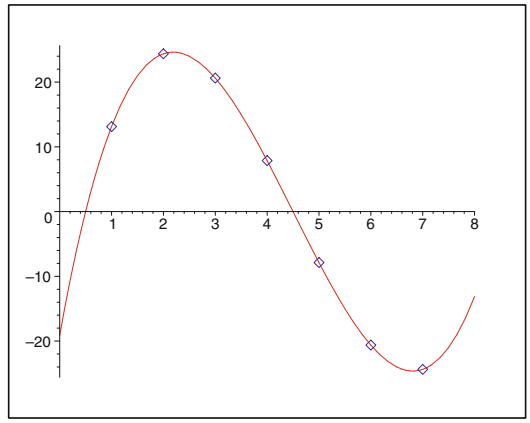

FIGURE 19.1. Cubic function evaluated at seven points

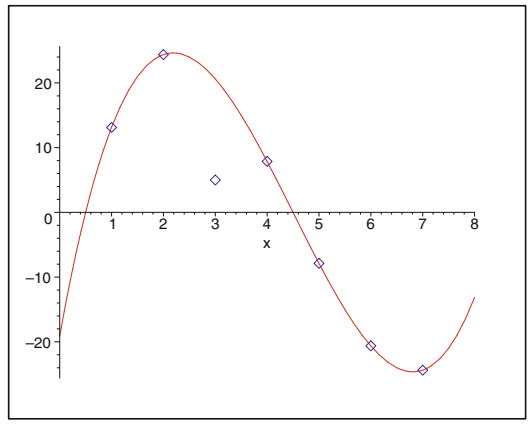

FIGURE 19.2. Cubic function going through six points and one error point

We return to the general case of a polynomial of degree t evaluated at n points. Suppose we know that there are at most e errors in our code-word. We then have the following (very inefficient) method for polynomial reconstruction.

- We produce the list of all subsets $\mathbb{X}$ of the n points with $n - e$ members.
- We then try to recover a polynomial of degree t. If we are successful then there is a good probability that the subset $\mathbb{X}$ is the valid set; if we are unsuccessful then we know that $\mathbb{X}$ contains an element which is in error.

This is clearly a silly algorithm to error correct a Reed–Solomon code. The total number of subsets we may have to take is given by

$$^{n}C_{n-e} = \frac{n!}{e! \cdot (n-e)!}.$$

But despite this we will still analyse this algorithm a bit more: To be able to recover a polynomial of degree t we must have that $t < n - e$, i.e. we must have more valid elements than there are coefficients to determine. Suppose we not only have e errors but we also have s "erasures", i.e. positions for which we do not even receive the value. Note that erasures are slightly better for the receiver than errors, since with an erasure the receiver knows the position of the erasure, whereas

with an error they do not. To recover a polynomial of degree t we will require

$$t < n - e - s.$$

But we could recover many such polynomials, for example if $t = n - e - s - 1$ then all such sets $\mathbb{X}$ of $n - e - s$ will result in a polynomial of degree at most t. To obtain a *unique* polynomial from the above method we will need to make sure we do not have too many errors/erasures.

It can be shown that if we can obtain at least $t + 2 \cdot e$ points then we can recover a unique polynomial of degree t which passes through $n-s-e$ points of the set of $n-s$ points. This gives the important equation that an error correction for Reed–Solomon codes can be performed uniquely provided

$$n > t + 2 \cdot e + s.$$

The only problem left is how to perform this error correction *efficiently*.

19.3.4. The Berlekamp–Welch Algorithm: We now present an efficient method to perform error correction for Reed–Solomon codes called the Berlekamp–Welch algorithm. The idea is to interpolate a polynomial in two variables through the points which we are given. Suppose we are given a code word with s missing values, and the number of errors is bounded by

$$e < t < \frac{n-s}{3}.$$

This means we are actually given $n - s$ supposed values of $y_i = f(x_i)$. We know the pairs (x_i, y_i) and we know that at most e of them are wrong, in that they do not come from evaluating the hidden polynomial $f(X)$. The goal of error correction is to try to recover this hidden polynomial.

We consider the bivariate polynomial

$$Q(X, Y) = f_0(X) - f_1(X) \cdot Y,$$

where f_0 (resp. f_1) is a polynomial of degree at most $2 \cdot t$ (resp. t). We impose the condition that $f_1(0) = 1$. We treat the coefficients of the f_i as variables which we want to determine. Due to the bounds on the degrees of the two polynomials, and the extra condition of $f_1(0) = 1$, we see that the number of variables we have is

$$v = (2 \cdot t + 1) + (t + 1) - 1 = 3 \cdot t + 1.$$

We would like the bivariate polynomial $Q(X, Y)$ to interpolate our points (x_i, y_i). By substituting in the values of x_i and y_i we obtain a linear equation in terms of the unknown coefficients of the polynomials f_i. Since we have $n - s$ such points, the number of linear equations we obtain is $n - s$. After determining f_0 and f_1 we then compute

$$f \leftarrow \frac{f_0}{f_1}.$$

To see that this results in the correct answer, consider the single polynomial in one variable

$$P(X) = Q(X, f(X))$$

where $f(X)$ is the polynomial we are trying to determine. We have $\deg P(X) \leq 2 \cdot t$. The polynomial $P(X)$ clearly has at least $n - s - e$ zeros, i.e. the number of valid pairs. So the number of zeros is at least

$$n - s - e > n - e - t > 3 \cdot t - t = 2 \cdot t,$$

since $e < t < \frac{n-s}{3}$. Thus $P(X)$ has more zeros than its degree, and it must hence be the zero polynomial. Hence,

$$f_0 - f_1 \cdot f = 0$$

and so $f = f_0/f_1$ since $f_1 \neq 0$.

Example: Again consider our previous example. We have received the invalid code-word

$$c = (44, 2, 25, 23, 86, 83, 14).$$

We know that the underlying code is for polynomials of degree $t = 2$. Hence, since $2 = t < \frac{n-s}{3} = 7/3 = 2.3$ we should be able to correct a single error. Using the method above we want to determine the polynomial $Q(X, Y)$ of the form

$$Q(X, Y) = f_{0,0} + f_{1,0} \cdot X + f_{2,0} \cdot X^2 + f_{3,0} \cdot X^3 + f_{4,0} \cdot X^4 - \left(1 + f_{1,1} \cdot X + f_{2,1} \cdot X^2\right) \cdot Y$$

which passes through the seven given points. Hence we have six variables to determine and we are given seven equations. These equations form the linear system, modulo $q = 101$,

$$\begin{pmatrix} 1 & 1 & 1^2 & 1^3 & 1^4 & -44 \cdot 1 & -44 \cdot 1^2 \\ 1 & 2 & 2^2 & 2^3 & 2^4 & -2 \cdot 2 & -2 \cdot 2^2 \\ 1 & 3 & 3^2 & 3^3 & 3^4 & -25 \cdot 3 & -25 \cdot 3^2 \\ 1 & 4 & 4^2 & 4^3 & 4^4 & -23 \cdot 4 & -23 \cdot 4^2 \\ 1 & 5 & 5^2 & 5^3 & 5^4 & -86 \cdot 5 & -86 \cdot 5^2 \\ 1 & 6 & 6^2 & 6^3 & 6^4 & -83 \cdot 6 & -83 \cdot 6^2 \\ 1 & 7 & 7^2 & 7^3 & 7^4 & -14 \cdot 7 & -14 \cdot 7^2 \end{pmatrix} \cdot \begin{pmatrix} f_{0,0} \\ f_{1,0} \\ f_{2,0} \\ f_{3,0} \\ f_{4,0} \\ f_{1,1} \\ f_{2,1} \end{pmatrix} = \begin{pmatrix} 44 \\ 2 \\ 25 \\ 23 \\ 86 \\ 83 \\ 14 \end{pmatrix}.$$

So we are solving the system

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 57 & 57 \\ 1 & 2 & 4 & 8 & 16 & 97 & 93 \\ 1 & 3 & 9 & 27 & 81 & 26 & 78 \\ 1 & 4 & 16 & 64 & 54 & 9 & 36 \\ 1 & 5 & 25 & 24 & 19 & 75 & 72 \\ 1 & 6 & 36 & 14 & 84 & 7 & 42 \\ 1 & 7 & 49 & 40 & 78 & 3 & 21 \end{pmatrix} \cdot \begin{pmatrix} f_{0,0} \\ f_{1,0} \\ f_{2,0} \\ f_{3,0} \\ f_{4,0} \\ f_{1,1} \\ f_{2,1} \end{pmatrix} = \begin{pmatrix} 44 \\ 2 \\ 25 \\ 23 \\ 86 \\ 83 \\ 14 \end{pmatrix} \pmod{101}.$$

We obtain the solution

$$(f_{0,0}, f_{1,0}, f_{2,0}, f_{3,0}, f_{4,0}, f_{1,1}, f_{2,1}) \leftarrow (20, 84, 49, 11, 0, 67, 0),$$

and hence the two polynomials

$$f_0(X) \leftarrow 20 + 84 \cdot X + 49 \cdot X^2 + 11 \cdot X^3 \text{ and } f_1(X) \leftarrow 1 + 67 \cdot X.$$

We find that

$$f(X) \leftarrow \frac{f_0(X)}{f_1(X)} = 20 + 57 \cdot X + 68 \cdot X^2,$$

which is precisely the polynomial we started with at the beginning of this section. Hence, we have corrected for the error in the transmitted code-word.

19.4. Shamir Secret Sharing

We now return to secret sharing schemes, and in particular the Shamir secret sharing scheme. We suppose we have n parties who wish to share a secret so that no t (or fewer) parties can recover the secret. Hence, this is going to be a $(t+1)$-out-of-n threshold secret sharing scheme.

First we suppose there is a trusted dealer who wishes to share the secret s in $\mathbb{F}_q$. He first generates a secret polynomial $f(X)$ of degree t with $f(0) = s$. That is, he generates random integers f_i in $\mathbb{F}_p$ for $i = 1, \ldots, t$ and sets

$$f(X) = s + f_1 \cdot X + \cdots + f_t \cdot X^t.$$

The trusted dealer then identifies each of the n players by an element in a set $\mathbb{X} \subset \mathbb{F}_q \setminus \{0\}$, for example we could take $\mathbb{X} = \{1, 2, \ldots, n\}$ if the characteristic of $\mathbb{F}_q$ was larger than n. Then if $i \in \mathbb{X}$, party i is given the share $s_i \leftarrow f(i)$. Notice that the vector

$$(s, s_1, \ldots, s_n)$$

is a code-word for a Reed–Solomon code. Also note that if $t+1$ parties come together then they can recover the original polynomial via Lagrange interpolation and hence the secret s. Actually, secret reconstruction can be performed more efficiently by making use of the equation

$$s \leftarrow f(0) = \sum_{i=1}^{n} s_i \cdot \delta_i(0).$$

Hence, for a set $Y \subset \mathbb{X}$, we define the vector r_Y by $r_Y = (r_{x_i,Y})_{x_i \in Y}$ to be the public "recombination" vector, where

$$r_{x_i,Y} = \prod_{x_j \in Y, x_j \neq x_i} \frac{-x_j}{x_i - x_j}.$$

Then, if we obtain a set of shares from a subset $Y \subset \mathbb{X}$ of the players, with $\#Y > t$, we can recover s via the simple summation.

$$s \leftarrow \sum_{x_i \in Y} r_{x_i,Y} \cdot s_i.$$

Also note that if we receive some possible values from a set of parties Y, then we can recover the original secret via the Berlekamp–Welch algorithm for decoding Reed–Solomon codes in the presence of errors, assuming the number of invalid shares is bounded by e where

$$e < t < \frac{\#Y}{3}.$$

Shamir secret sharing is an example of a secret sharing scheme which can be made into a pseudo-random secret sharing scheme, or PRSS. This is a secret sharing scheme which allows the parties to generate a sharing of a random value with almost no interaction. In particular the interaction can be restricted to a set-up phase only.

To define the Shamir pseudo-random secret sharing scheme we take our n parties, which we shall (for sake of concreteness) label by the set $\mathbb{X} = \{1, 2, \ldots, n\}$, and threshold value $t+1$. Then for every subset $A \subset \mathbb{X}$ of size $n - t$ we define the polynomial $f_A(X)$ of degree t by the conditions

$$f_A(0) = 1 \text{ and } f_A(i) = 0 \text{ for all } i \in \mathbb{X} \setminus A.$$

In the initialization phase for our PRSS we create a secret value $r_A \in S$, where S is some key space. For each subset A, the value of r_A is securely distributed to every player in A. The n parties also agree on a public pseudo-random function which is keyed by the secret values r_A,

$$\psi : \begin{cases} S \times S & \longrightarrow & \mathbb{F}_q \\ (r_A, a) & \longmapsto & \psi(r_A, a). \end{cases}$$

Now suppose the parties wish to generate a new secret sharing of a random value. By some means, either by interaction or by prearrangement (e.g. a counter value), they select a public random value $a \in S$. They then generate a random Shamir sharing where the underlying polynomial of degree t is given by

$$f(X) = \sum_{A \subset \mathbb{X}, |A| = n-t} \psi(r_A, a) \cdot f_A(X),$$

where the sum is over all subsets A of size $n - t$. This means that each party i receives the share

$$s_i \leftarrow \sum_{i \in A \subset \mathbb{X}, |A| = n-t} \psi(r_A, a) \cdot f_A(i)$$

where the sum is over all subsets A of size $n-t$ which contain the element i. Finally the random value which is shared, via the Shamir secret sharing scheme, is given by

$$s = \sum_{A \subset \mathbb{X}, |A|=n-t} \psi(r_A, a) \cdot f_A(0) = \sum_{A \subset \mathbb{X}, |A|=n-t} \psi(r_A, a).$$

Other secret sharing schemes can also be turned into PRSSs. Consider the n-out-of-n scheme over $\mathbb{F}_q$ for which the secret is given by $s = s_1 + \cdots + s_n$, where s_i is chosen uniformly at random from the field $\mathbb{F}_q$. This is immediately a PRSS, since to generate a sharing of a random value unknown to any party, each party only needs to generate a random value.

In Chapter 22 we shall require not only a pseudo-random secret sharing, but also a variant called pseudo-random zero sharing, or PRZS, for the Shamir secret sharing scheme. In pseudo-random zero sharing we wish to generate random sharings of the value zero, with respect to a polynomial of degree $2 \cdot t$. The exact reason why the polynomial has to be of degree $2 \cdot t$ will become apparent when we discuss our application in Chapter 22. To enable this extra functionality we require exactly the same set-up as for the PRSS, but now we use a different pseudo-random function,

$$\psi : \begin{cases} S \times S \times \{1, \ldots, t\} & \longrightarrow & \mathbb{F}_q \\ (r_A, x, j) & \longmapsto & \psi(r_A, x, j). \end{cases}$$

Then to create a degree $2 \cdot t$ Shamir secret sharing of zero, the parties pick a number a as before. The underlying polynomial is then given by

$$f(X) = \sum_{A \subset \mathbb{X}, |A|=n-t} \left(\sum_{j=1}^{t} \psi(r_A, a, j) \cdot X^j \cdot f_A(X) \right).$$

Clearly this is a polynomial which shares the zero value, as the polynomial is divisible by X. The share for party i is given by

$$s_i \leftarrow \sum_{A \subset \mathbb{X}, |A|=n-t} \left(\sum_{j=1}^{t} \psi(r_A, a, j) \cdot i^j \cdot f_A(i) \right).$$

19.5. Application: Shared RSA Signature Generation

We shall now present a simple application of a secret sharing scheme, which has applications in the real world. Having introduced digital certificates in Chapter 18, we present an application of secret sharing to distributed RSA signatures. Suppose a company is setting up a certificate authority to issue RSA signed certificates to its employees to enable them to access various corporate services. It considers the associated RSA private key to be highly sensitive, after all if the private key was compromised then the company's entire corporate infrastructure could also be compromised. Suppose the public key is (N, e) and the private key is d.

The company decides that to mitigate the risk it will divide the private key into three shares and place the three shares on three different continents. Thus, for example, there will be one server in Asia, one in America and one in Europe. As soon as the RSA key is generated, the company generates three integers d_1, d_2 and d_3 such that

$$d = d_1 + d_2 + d_3 \pmod{\phi(N)}.$$

The company then removes all knowledge of d and places d_1 on a secure computer in Asia, d_2 on a secure compute in America and d_3 on a secure computer in Europe.

Now an employee wishes to obtain a digital certificate. This is essentially the RSA signature on a (probably hashed) string m. The employee simply sends the string m to the three computers,

which respond with

$$s_i \leftarrow m^{d_i} \text{ for } i = 1, 2, 3.$$

The valid RSA signature is then obtained by multiplying the three shares together, i.e.

$$s \leftarrow s_1 \cdot s_2 \cdot s_3 = m^{d_1+d_2+d_3} = m^d.$$

Here, we have used the fact that the RSA function is multiplicatively homomorphic.

This scheme appears to solve the problem of not putting the master signature key in only one location. However, the employee now needs the three servers to be online in order to obtain his certificate. It would be much nicer if only two had to be online, since then the company could cope with outages of servers. The problem is that the above scheme essentially implements a 3-out-of-3 secret sharing scheme, whereas what we want is a 2-out-of-3. Clearly, we need to apply something along the lines of Shamir secret sharing. However, the problem is that the number $\phi(N)$ needs to be kept secret, and the denominators in the Lagrange interpolation formulae may not be coprime to $\phi(N)$.

There have been many solutions proposed to the above problem of threshold RSA, however, the most elegant and simple is due to Shoup. Suppose we want a t-out-of-n sharing of the RSA secret key d, where we assume that e is chosen so that it is a prime and $e > n$. We adapt the Shamir scheme as follows: Firstly a polynomial of degree $t-1$ is chosen, by selecting f_i modulo $\phi(N)$ at random, to obtain

$$f(X) = d + f_1 \cdot X + \cdots + f_{t-1} \cdot X^{t-1}.$$

Then each server is given the share $d_i = f(i)$. The number of parties n is assumed to be fixed and we define Δ to be the constant $\Delta = n!$.

Now suppose a user wishes to obtain a signature on the message m, i.e it wants to compute $m^d \pmod N$. It sends m to each server, which then computes the signature fragment as

$$s_i = m^{2 \cdot \Delta \cdot d_i} \pmod N.$$

These signature fragments are then sent back to the user. Suppose now that the user obtains fragments back from a subset $Y = \{i_1, \ldots, i_t\} \subset \{1, \ldots, n\}$, of size greater than or equal to t. This set defines a "recombination" vector $\mathbf{r}_Y = (r_{i_j,Y})_{i_j \in Y}$ defined by

$$r_{i_j,Y} \leftarrow \prod_{i_k \in Y, i_j \neq i_k} \frac{-i_k}{i_j - i_k}.$$

We really want to be able to compute this modulo $\phi(N)$, but that is impossible since $\phi(N)$ is not known to any of the participants. In addition the denominator may not be invertible modulo $\phi(N)$. However, we note that the denominator in the above divides Δ and so we have that $\Delta \cdot r_{i_j,Y} \in \mathbb{Z}$. Hence, the user can compute

$$\sigma \leftarrow \prod_{i_j \in Y} s_{i_j}^{2 \cdot \Delta \cdot r_{i_j,Y}} \pmod N.$$

We find that this is equal to

$$\sigma = \left(m^{4 \cdot \Delta^2}\right)^{\sum_{i_j \in Y} r_{i_j,Y} \cdot d_{i_j}} = m^{4 \cdot \Delta^2 \cdot d} \pmod N,$$

with the last equality working due to Lagrange interpolation modulo $\phi(N)$. From this partial signature we need to recover the real signature. To do this we use the fact that we have assumed that $e > n$ and that e is a prime. These latter two facts mean that e is coprime to $4 \cdot \Delta^2$, and so via the extended Euclidean algorithm we can compute integers u and v such that

$$u \cdot e + v \cdot 4 \cdot \Delta^2 = 1,$$

from which the signature is computed as

$$s \leftarrow m^u \cdot \sigma^v \pmod{N}.$$

That s is the valid RSA signature for this public/private key pair can be verified since

$$\begin{aligned} s^e &= (m^u \cdot \sigma^v)^e = m^{e \cdot u} \cdot m^{4 \cdot e \cdot v \cdot \Delta^2 \cdot d}, \\ &= m^{u \cdot e + 4 \cdot v \cdot \Delta^2} = m. \end{aligned}$$

One problem with the protocol as we have described it is that the signature shares s_i may be invalid. See the paper by Shoup in the Further Reading section to see how this problem can be removed using zero-knowledge proofs.

Chapter Summary

- We have defined the general concept of secret sharing schemes and shown how these can be constructed, albeit inefficiently, for any access structure.
- We have introduced Reed–Solomon error-correcting codes and presented the Berlekamp–Welch decoding algorithm.
- We presented Shamir's secret sharing scheme, which produces a highly efficient, and secure, secret sharing scheme in the case of threshold access structures.
- We extended the Shamir scheme to give both pseudo-random secret sharing and pseudo-random zero sharing.
- Finally we showed how one can adapt the Shamir scheme to enable the creation of a threshold RSA signature scheme.

Further Reading

Shamir's secret sharing scheme is presented in his short ACM paper from 1979. Shoup's threshold RSA scheme is presented in his Eurocrypt 2000 paper; this paper also explains the occurrence of the Δ^2 term in the above discussion, rather than a single Δ term. A good description of secret sharing schemes for general access structures, including some relatively efficient constructions, is presented in the relevant chapter in Stinson's book.

A. Shamir. *How to share a secret.* Communications of the ACM, **22**, 612–613, 1979.

V. Shoup. *Practical threshold signatures.* In Advances in Cryptology – Eurocrypt 2000, LNCS 1807, 207–220, Springer, 2000.

D. Stinson. *Cryptography: Theory and Practice.* Third Edition. CRC Press, 2005.