1 Introduction

Today, we will consider the robustness threshold. We begin by showing that Shamir’s scheme has \( t_r = (n + t_f)/2 \); the proof will rely on decoding Reed-Solomon codes (with the Berlekamp-Welch algorithm). We give a brief introduction to linear error-correcting codes and describe their relationship to secret sharing schemes. Next, we consider several techniques for achieving optimal robustness: (1) Using digital signatures; (2) using (information-theoretic) MACs; (3) using commitments and Merkle trees. Finally, we discuss Verifiable Secret Sharing (VSS). We describe a general schema: combining Shamir’s SSS with homomorphic commitments.

2 Robustness

Proposition 1 For Shamir’s SSS, \( t_r = \frac{t_f + n}{2} \).

Proof Let \( k \) denote \( t_f \) and let \( t \) denote \( t_r \). Suppose we are given \( n \) points, \((z_1, y_1), \ldots, (z_n, y_n)\) and we assert that there exists a polynomial \( P \) of degree \( k - 1 \) such that at least \( t \) of our \( n \) points agree with \( P \): that is, \( P(z_i) = y_i \). The problem is to find \( P \). We will describe the Berlekamp-Welch algorithm (not the most efficient, but the simplest). It requires the assumption \( t \geq \frac{n + k}{2} \).

Let \( B \) be the set of ‘bad’ points. (That is, for the particular \( P \) that we care about, \( P(z_i) \neq y_i \) for \((z_i, y_i) \in B\).) We know that

\[
|B| \leq n - t \leq \frac{n - k}{2}.
\]

Define the polynomial \( E(x) := \prod_{i \in B} (x - z_i) \). This is called the error locator polynomial; the roots of the polynomial are the ‘bad’ points, so assuming we can factor it, \( E \) will tell us which shares are bad (namely, which servers lied).

Define another polynomial: \( Q(x) = E(x)P(x) \). What is its degree? The degree of \( E \) is at most \((n - k)/2\), and the degree of \( P \) is \( k - 1 \), so the degree of \( Q \) is at most \( \frac{n + k}{2} - 1 \).

\[\text{“You will see why. Well, there’s no reason why.”} \quad \text{–YD}\]
What happens when $Q$ is evaluated at a point $z_i$? We have:

$$Q(z_i) = E(z_i)P(z_i)$$

$$= \begin{cases} 
E(z_i) \cdot y_i & \text{if } i \notin B; \\
0 \cdot P(z_i) & \text{if } i \in B.
\end{cases}$$

$$= \begin{cases} 
E(z_i) \cdot y_i & \text{if } i \notin B; \\
0 \cdot y_i & \text{if } i \in B.
\end{cases}$$

$$= E(z_i) \cdot y_i \text{ for all } i.$$

At this stage, what do we know, and what are the unknowns? In the equation $Q(z_i) = E(z_i) \cdot y_i$, we know $y_i$ and $z_i$, but not $Q(x)$ and $E(x)$. The polynomial $Q$ gives $(n + k)/2$ (one more than the degree) unknowns, and $E$ gives $(n - k)/2$ (exactly the degree, since it’s monic) unknowns, so we have $(n + k)/2 + (n - k)/2 = n$ unknowns in each equation, and they are exactly the coefficients of the polynomials. Thus: the family of equations \{\[Q(z_i) = E(z_i) \cdot y_i : 1 \leq i \leq n\]\} is a set of $n$ linear equations in $n$ unknowns. (It might appear at first glance that the equations are non-linear—we have powers of $z_i$, after all—but $z_i$ is a constant, and raising it to a power gives another fixed value, which becomes the ‘coefficient’ of the unknown $a_i$.)

First, we observe that if there does exist a polynomial $P(x)$ which agrees with at least $t$ of the $n$ points, then the above system has at least one solution. (Possibly more than one.)

Second, we will show that $Q(z_i)E'(z_i) = Q'(z_i)E(z_i)$ for all $i$; we will deduce that the two polynomials $Q(x)E'(x)$ and $Q'(x)E(x)$ are equal and argue that it makes sense to divide by $E$ and $E'$.

We have:

$$Q(z_i)E'(z_i) = E(z_i) \cdot y_i \cdot E'(z_i)$$

$$= E(z_i) \cdot E'(z_i) \cdot y_i$$

$$= E(z_i)Q'(z_i) \text{ for all } i.$$

But:

$$\text{deg} \left( Q(z_i)E'(z_i) \right) = \frac{n + k}{2} - 1 + \frac{n - k}{2} = n - 1 = \text{deg} \left( E(z_i)Q'(z_i) \right).$$

Two polynomials of degree $n - 1$ which agree at $n$ points must be equal, so $QE' = Q'E$. Finally, it makes sense to divide by $E$ and $E'$ because we know that for the one solution whose existence we posit in advance, it must be the case that $Q$ is divisible by $E$; this implies the same divisibility relation for every other solution.

This secret sharing scheme turns out to be deeply related to error-correcting codes, and in particular to the Reed-Solomon Code. We will (in part) describe this relationship here, beginning with an introduction to coding theory.

## 3 Error-Correcting Codes (ECC)

### 3.1 A Brief Primer

Let $\mathbb{F}$ be a finite field of cardinality $q$ and consider the vector space $\mathbb{F}^n$. A code $\mathcal{C}$ is a subset of $\mathbb{F}^n$ with cardinality $q^k$; we call $q$ the **alphabet** and $k$ the **dimension** of the code. (Most codes are **subspaces** of dimension $k$.) Enumerate $\mathcal{C} = \{c_1, \ldots, c_{q^k}\}$. 

L2-2
Given \( x, y \in \mathbb{F}^n \), define the **Hamming Distance** between \( x \) and \( y \) to be

\[
\text{Ham}(x, y) = \# \{ i : x_i \neq y_i \}.
\]

(For example, if \( \mathbb{F}^n = \{0, 1\}^3 \), \( x = (011) \) and \( y = (111) \), then \( \text{Ham}(x, y) = 1 \) because \( x \) and \( y \) differ in one entry.) Let \( d \) denote the **minimal pairwise distance** among the codewords (that is, \( d \) is the smallest positive integer such that for any \( i \neq j \), we have \( \text{Ham}(c_i, c_j) \geq d \)).

Finally, let \( n \) denote the block length of the code. We will describe a code \( C \) and all of its parameters with the compact notation \( C = [n, k, d]_q \).

### 3.2 ECC’s and SSS’s

Consider a secret sharing scheme with \( t_p = 0 \) (so there is no privacy). We will draw an analogy between the scheme and an ECC, and we will find that the fault tolerance \( t_f \) is exactly \( n - d + 1 \). That is: given \( x \in \mathbb{F}^n \), if at most \( d - 1 \) coordinates are \( \perp \) (so at least \( n - d + 1 \) coordinates are good), then you can correct \( x \) to the appropriate \( c \in C \).

How do we use codes for secret sharing schemes? Write an efficient procedure which is a bijection between \( \mathbb{F}^k \) and \( C \). View a message as \( M = m_1 \ldots m_k \). Encode it as \( c \in C \). (Usually, this is a linear map \( \mathbb{F}^k \rightarrow \mathbb{F}^n \).) Then somebody erases a few (that is to say, fewer than \( d \)) shares, but you still recover it because your incorrect code word lies in a unique ball centered at a correct code word.

Error-correcting codes therefore correspond exactly to information dispersal schemes, with the additional property that if we draw balls around each code word \( c \in C \) of radius \( \frac{d-1}{2} \), the balls do not intersect. In principle, we can actually correct \( \frac{d-1}{2} \) errors and detect \( d - 1 \) erasures. (A translation: “missing shares” in a SSS are “erasures” in a code; “wrong shares” in a SSS are “errors” in a code.) In the language of secret sharing schemes: \( t_f = n - d + 1 \) (and \( t_f = k \), where \( k \) is the number of points needed to determine the corresponding polynomial), and \( t_r = n - \frac{d-1}{2} = \frac{n+(n-d-1)}{2} = \frac{n+t_f}{2} \).

### 3.3 Reed-Solomon

Let \( P(x_i) = M_k x^{k-1} + \cdots + M_1 \). Then set \( c = P(z_1), \ldots, P(z_n) \). (This gives you redundancy.) We get an \( [n, k, d]_q \)-code where \( q \geq n \) and \( d = n - k + 1 \).

For all messages \( M_0, M_1 \), we have \( P_0(z_i) = P_1(z_i) \), so \( (P_0 - P_1)(z_i) = 0 \), which forces the number of points \( z_i \) to be at most \( k - 1 \). Thus \( P_0 \) and \( P_1 \) disagree on \( n - k + 1 \) points; this is exactly the Hamming distance.

The Reed-Solomon code corrects and detects the optimal number of points. Given that Shamir’s SSS is information-theoretically optimal, it turns out that distance \( d = n - k + 1 \) is also optimal.

One of the few quibbles with this code is that the alphabet size is large – we like binary codes. (Using \( n = 2^x \) for some \( x \) makes the alphabet easily expressable in binary, but we still have to be able to do finite field operations.)
4 Secret Sharing with $t_r = t_f$

Assume we have a SSS, given by Share and Rec, with fault tolerance $t_f$ and privacy $t_p$. We will use this to create a second SSS, given (of course) by Share' and Rec', such that $t'_r = t'_f$.

Some of the schemes we develop (here and elsewhere) will introduce a new public parameter, pub'. If this is not desirable, then we can copy the contents of pub' and add it to each player's share, at the expense of forcing $t'_r = \max(t_r, \frac{n}{2} + 1)$ and increasing each local share by $|\text{pub}'|$. Assuming a majority of honest players, we can reconstitute pub by taking a 'majority vote' among the shares. This technique is called "replication."\(^2\) Conversely, if pub is empty, then you would need at least $\frac{n}{2} + 1$ honest shares to recover the message.

4.1 Approach 1: Signature Schemes

Let \((\text{Sig-Gen}, \text{Sig}, \text{Ver})\) be generation, signature, and verification algorithms of a secure digital signature scheme.

Define

\[
\begin{align*}
\text{Share}'(M) & : (S_1, \ldots, S_n, \text{pub}) \leftarrow \text{Share}(M) \\
(SK, VK) & \leftarrow \text{Sig-Gen}(1^k) \\
S'_i & = (S_i, \text{Sig}_SK(S_i), VK) \\
\text{pub}' & = (\text{pub}, VK)
\end{align*}
\]

The recovery algorithm Rec' will use the digital signature to determine which shares are correct and then use only the correct shares to decode the message.

It’s clear that $t_r = t_f$ and that $t_s = 1$ (i.e., the soundness threshold is optimal).

On the other hand: what about information theoretic security? Instead of a signature scheme, let’s use a message authentication code. Assume we have a trusted dealer.

For any $P_i$ and $P_j$, the dealer $D$ chooses a one-time MAC key $K_{ij}$; she sends $K_{ij}$ to $P_i$, and sends $S_j$ and $\text{MAC}_{K_{ij}}(S_j)$ to $S_j$. All in all, a player $P_i$ gets $S_i, \text{MAC}_{K_{i1}}(S_i), \ldots, \text{MAC}_{K_{in}}(S_i)$ and $K_{i1}, K_{i2}, \ldots, K_{in}$.

Remark In the literature, this is the information checking protocol; people use

\[K = (a, b), \text{MAC}_{a,b}(m) = am + b \mod p.\]

Recall that in a MAC, the sender and receiver share a key and only the receiver can verify the authenticity of the message. FYI: the optimal one-time MAC on an $n$-bit message with security $\varepsilon$ has tag length $\log(1/\varepsilon) + O(1)$ and key length $O(\log n + \log(1/\varepsilon))$.

Let’s consider both possibilities for recovery. If the recovery is attempted by a server $P_i$, then that server knows all the keys, so she only accepts correctly authenticated shares. Thus, we get $t_r = t_f$ and $t_s = 1$. On the other hand, what is the optimal threshold you can get for an external

\(^2\)Recall that in the last lecture, we briefly introduced replication along side of information dispersal. Replication is optimally robust, but costly; IDS is not optimally robust, but reduces share size. See, for example, Shamir’s SSS.
Instead of signatures, let’s use commitment schemes.

### 4.2 Approach 2: Commitment Schemes

Instead of signatures, let’s use commitment schemes. A commitment scheme is a nice cryptographic building block; it’s given by Setup, Commit, and Open. Let’s ignore Setup but review the other components of a commitment scheme. Define:

\[
\begin{align*}
\text{Commit}(m; r) & \rightarrow (c, d) \\
\text{Open}(c', d') & \rightarrow m'
\end{align*}
\]

where \( m' \in M \cup \{\bot\} \). Think of \( c \) as a lock box and \( d \) as a key. After the first stage (in which Alice sends \( c \) to Bob), Bob has the box but cannot open it without \( d \) (hiding), but Alice can no longer change her mind about the contents of the box (binding). The correctness requirement is that \( \text{Open}(\text{Commit}(m)) = m \). The hiding requirement is that \( c(m_0) \approx c(m_1) \) for all \( m_0, m_1 \). The binding property is that it’s hard to come up with \( c, d_0 \) and \( d_1 \) such that if both \( \text{Open}(c, d_0) = m_0 \) and \( \text{Open}(c, d_1) = m_1 \), then \( m_0 \neq \bot, m_1 \neq \bot \) and \( m_0 \neq m_1 \) (that is, you can’t open the box in two different ways).

Our goal is to use this scheme to add robustness to a secret sharing scheme. Let’s start with the variant in which there is a public share. The sharing algorithm will be:

\[
\begin{align*}
\text{Share}'(m) & : (S_1, \ldots, S_n, \text{pub}') \leftarrow \text{Share}(M) \\
(c_i, d_i) & \leftarrow \text{Commit}(S_i) \\
\text{pub}' & \leftarrow (\text{pub}, c_1, \ldots, c_n) \\
S'_i & = (S_i, c_i, d_i)
\end{align*}
\]

The recovery algorithm \( \text{Rec}'() \) will only use a share \( S'_i \) if both \( c'_i = c_i \) and \( \text{Open}(c_i, d'_i) = S'_i \) are satisfied.

This scheme has two issues: first, that \( \text{pub}' \) exists (and must be authenticated), and second, that it’s linear in the number of players (which is quite long). To eliminate \( \text{pub}' \) entirely, we could replicate it among the shares, but since it’s so long, this will be inefficient. Hence, let’s reduce its size first.

To make \( \text{pub}' \) shorter, build a Merkle tree: store data in the leaves of the tree, and has as you go up. (Nodes further up in the tree are the hashes of their respective children.) We build a complete binary tree, calling the root \( c \), using a collision-resistant hash function. \( c \) binds the tree. Now make

\[\text{“I hope I’m slow enough that, if anything, I’m underwhelming you.”} - YD\]
the public share $\text{pub}' = (\text{pub}, c)$ and make the secret shares $S'_i = (S_i, c_i, d_i$, the verification path). (The verification path is $(\log n) + 1$ values.) The size of $\text{pub}$ is now $O(k)$ (where $k$ is the security parameter), and the size of $S'_i$ has grown by an additive factor of $O(\log n)$, and the scheme achieves the same robustness property. (Now, if you want, you can use replication on $c$ to eliminate $\text{pub}'$ completely.)

5 Verifiable Secret Sharing (VSS) Schemes

In all the applications so far, we have assumed the dealer to be honest. In some applications, we will want to assume dishonest dealer. We can then no longer afford to have a noninteractive protocol because the players cannot determine whether the dealer is dishonest just by looking at their shares; Share will need to illuminate inconsistencies with a dishonest dealer. (It’s possible that Rec is noninteractive.) We introduce yet another parameter, $t_v$ (which will turn out to equal $t_r$ in our applications). If there are $t_v$ honest players in the protocol and they accept, then they know that a consistent value is shared. We will only consider VSS schemes in the robust settings where the dealer can collude with some of the players. Therefore, for VSS, we will simply use the same threshold $t_r$, but strengthen its meaning as follows.

1. A stronger robustness property. If $t_r$ people accent in the sharing protocol (whether or not the dealer is honest), then, with all but negligible probability, those $t_r$ people will succeed in the reconstruction protocol to recover the unique secret $S$ (irrespective of the number of times they repeat the protocol).

2. Correctness. If $D$ is honest and there are at least $t_r$ honest players, then all the honest players accept with the dealer’s value.

3. Privacy. Of course, we have the usual privacy property which states that when the dealer is honest, no $t_p$ players can determine the message.

We will assume that $t_r$ and $t_f$ are greater than $n/2$. (Else, the dealer could honestly share with bad guys and dishonestly share with good guys.)

The High-Level Overview We postpone the proof of the generic scheme; instead, we’ll discuss the high-level idea of a VSS scheme and then give two instantiations. The actual protocol for VSS schemes will share some properties with the scheme described in section 4.2, but we will need the extra property of a specially constructed (namely homomorphic) commitment scheme. Today, we’ll discuss computational VSS schemes; next week, we will mention information-theoretic schemes.

We assume in these schemes that the dealer $D$ is present at the time of Share(). Sometimes, $D$ may also be present at Rec() (but usually not). Begin by using Shamir’s SSS. For now, let $t_f = t_r = t$; use $t$-out-of-$n$. Generate $S_1, \ldots, S_n$, shares of $M$. Recall that $S_i = a_{i-1}z_i^{-1} + \cdots + a_1z_i + M$.

Using a special commitment scheme, we will commit to the coefficients of the polynomial in Shamir’s SSS. By ‘special’ here we indicate two properties: the commitment scheme must be “strongly homomorphic”: that is, from $\text{Commit}(m_1, r_1)$ and $\text{Commit}(m_2, r_2)$, we should be able to determine $\text{Commit}(m_1 + m_2, r_1 * r_2)$ without knowing $m_i$ and $r_i$. (As a consequence of these two properties, we can extend by linearity: for any $\alpha \in \mathbb{Z}^{\geq 0}$, given $\text{Commit}(m, r)$, we should be able to compute $\text{Commit}(\alpha m, \alpha r)$ where $\alpha m = m + m + \cdots + m$ and $\alpha r = r * r * \cdots * r$.)
Now: the dealer commits to the coefficients, with \( c_i = \text{Commit}(a_i, r_i) \) for \( 1 \leq i \leq t - 1 \) and \( c_0 = \text{Commit}(M, r_0) \). Players can use the properties of the scheme to compute \( \text{Commit}(S_i, v_i) \) for each share \( S_i = \sum_j a_j z_i^j \). The dealer gives each player one decommitment, \( v_i \), to her own share, and now each player can make a local check that her share is correct. If \( P_i \) is happy (i.e., her check passes), great. If not, then (still in the sharing phase) \( P_i \) announces that she is unhappy, and \( D \) announces \( (S_i, v_i) \). (Note that when the dealer announces these shares, the other players must remember the ‘free’ shares.) Players disqualify the dealer if either the number of ‘complaining’ people is at least \( n/2 \), or if an \( (S_i, v_i) \) revealed is incorrect (and every other player can check this). If the dealer is not disqualified, then simply ignore the handful of guys who complained. This is the end of the sharing phase; recovery is the same as in the previous schemes, except we only use the shares which decommit correctly and reconstruct the message with these shares.

**Feldman’s VSS Scheme** Define \( \text{Commit}(a) = g^a \mod p \). This scheme provides perfect binding, but only computational hiding. The homomorphic property of the scheme gives

\[
g^{a_1} \cdot g^{a_2} = g^{a_1 + a_2}.
\]

**Pedersen’s VSS Scheme** Define \( \text{Commit}(a; r) = g^a h^r \). This scheme is perfectly hiding but only computationally binding. The homomorphic property of the scheme gives

\[
g^{a_1} h^{r_1} \cdot g^{a_2} h^{r_2} = g^{a_1 + a_2} h^{r_1 + r_2}.
\]

**References**

