# Summary of Lecture on Secret Sharing

**Abstract**

This is a summary of Lecture 25 (12/3/19) in Fall 2019, taught by Dr. Tal Rabin. It is written by Prof. Malkin rather quickly, and may have some omissions or errors. The notes are not in the order the material was presented in class.

## 1 Motivation and Definition

Intuitively, a secret sharing scheme allows a *dealer* to share a secret $m$ among $n$ parties $P_1, \ldots, P_n$, such that any *authorized* subset of parties can use all their shares to reconstruct the secret, while any other (non-authorized) subset learns nothing about the secret from their shares.

Secret sharing has some direct applications, where we need to distribute a secret to several parties/servers, in order to distribute the required trust, as well as to allow reconstruction even if some of the parties fail. Examples include storing a digital wallet or a master key (DNSSEC used 5-out-of-7 secret sharing scheme for its root key). Nuclear codes are also often given as an example, though I don't know what is actually used in practice for those. Additionally, secret sharing is a useful tool in many larger cryptographic systems (notably, secure computation).

Note that simply giving some of the bits of the secret to each party certainly reveals information (and it cuts the search space and significantly reduces security – in some cases, depending on what the secret is used for, a fraction of the secret bits can be used to completely recover the entire secret). We want no information whatsoever to be leaked to any unauthorized subset.

Secret sharing can be defined with respect to any access structure that specifies the set of authorized subsets, as long as that access structure is monotone (namely, if a subset is authorized, any larger subset should also be authorized). In class, we gave a definition for the common case of *t-out-of-n* , or *threshold* secret sharing, where authorized subsets are all those of size at least $t$, while sets of size less than $t$ are not authorized.

**Definition 1** ($t$-out-of-$n$ secret sharing syntax and correctness). *A* $t$-out-of-$n$ secret sharing scheme *over message space $\mathcal{M}$ is a pair of algorithms* ($\texttt{Share}, \texttt{Reconstruct}$) *such that:*

- `Share` *is a randomized algorithm that on any input $m \in \mathcal{M}$ outputs a $n$-tuple of shares $(s_1, \ldots, s_n)$.*
- `Reconstruct` *is a deterministic algorithm that given a $t$-tuple of shares outputs a message in $\mathcal{M}$*

*and satisfying the following* correctness *requirement:*
$\forall m \in \mathcal{M}, \forall S = \{i_1, \ldots, i_t\} \subseteq \{1, \ldots, n\}$ *of size $t$,*

$$\Pr_{\texttt{Share}(m) \to (s_1, \ldots, s_n)} [\texttt{Reconstruct}(s_{i_1}, \ldots, s_{i_t}) = m] = 1$$

**Definition 2** (secret sharing security)**.** *A $t$-out-of-$n$ secret sharing scheme* $(\texttt{Share}, \texttt{Reconstruct})$ *over $\mathcal{M}$ is perfectly secure if:*

*$\forall m, m' \in \mathcal{M}$, $\forall S \subseteq \{1, \ldots, n\}$ s.t. $|S| < t$, the following distributions are identical:*

$$\{(s_i | i \in S) : (s_1, \ldots, s_n) \leftarrow \texttt{Share}(m)\}$$

$$\{(s'_i | i \in S) : (s'_1, \ldots, s'_n) \leftarrow \texttt{Share}(m')\}$$

Recall that two distributions are identical if they give exactly the same probability for every possible value. Thus, the above definition can be restated as follows: $\forall m, m' \in \mathcal{M}$, $\forall S \subseteq \{1, \ldots, n\}$ s.t. $|S| < t$, and for any set $\alpha = (\alpha_1, \ldots, \alpha_{|S|})$, we have that

$$\Pr_{\texttt{Share}(m) \to (s_1, \ldots, s_n)} [(s_i | i \in S) = \alpha] = \Pr_{\texttt{Share}(m') \to (s'_1, \ldots, s'_n)} [(s'_i | i \in S) = \alpha]$$

## 2 A 2-out-of-2 Secret Sharing Scheme

Consider the following 2-out-of-2 secret sharing scheme for $\mathcal{M} = \mathbb{Z}_p$.

$\texttt{Share}_{2-2}$: On input $m \in \mathbb{Z}_p$,

- select $s_1 \in \mathbb{Z}_p$ uniformly at random.
- set $s_2 = m - s_1 \pmod p$
- output $(s_1, s_2)$

$\texttt{Reconstruct}_{2-2}$: On input $(s_1, s_2) \in \mathbb{Z}_p \times \mathbb{Z}_p$,

- output $s_1 + s_2 \pmod p$

Note that in this scheme, the size of each share is the size of the secret $|m|$ (since both can be any element of $\mathbb{Z}_p$). It can be shown that this size is required (there's no scheme where each share is shorter than the secret).

**Theorem 1.** *The scheme* $(\texttt{Share}_{2-2}, \texttt{Reconstruct}_{2-2})$ *above is a correct and secure 2-out-of-2 secret sharing scheme over $\mathcal{M} = \mathbb{Z}_p$.*

*Proof.* Correctness follows immediately from the way $\texttt{Share}_{2-2}$ and $\texttt{Reconstruct}_{2-2}$ were defined: $\forall m \in \mathbb{Z}_p$,

$$\Pr_{\texttt{Share}(m) \to (s_1, s_2)} [\texttt{Reconstruct}(s_1, s_2) = m] = \Pr_{\texttt{Share}(m) \to (s_1, s_2)} [s_1 + m - s_1 = m \pmod p] = 1$$

Security follows since each share individually is distributed uniformly at random, so does not contain any information about the secret. To prove it formally, fix any $m, m' \in \mathbb{Z}_p$, and consider a subset of 1 share. In case it is the first share $s_1$, since it is chosen uniformly (and $|\mathbb{Z}_p| = p$), we have that $\forall \alpha \in \mathbb{Z}_p$,

$$\Pr_{\texttt{Share}(m) \to (s_1, s_2)} [s_1 = \alpha] = \frac{1}{p} = \Pr_{\texttt{Share}(m') \to (s'_1, s'_2)} [s'_1 = \alpha]$$

In case it is the second share $s_2$, we have that $\forall \alpha \in \mathbb{Z}_p$,

$$\Pr_{\texttt{Share}(m)\to(s_1,s_2)}[s_2 = \alpha] = \Pr_{\texttt{Share}(m)\to(s_1,s_2)}[m - s_1 = \alpha \pmod p] = \Pr_{\texttt{Share}(m)\to(s_1,s_2)}[s_1 = m - \alpha \pmod p] = \frac{1}{p}$$

Similarly,

$$\Pr_{\texttt{Share}(m')\to(s_1',s_2')}[s_2' = \alpha] = \Pr_{\texttt{Share}(m')\to(s_1',s_2')}[s_1' = m - \alpha \pmod p] = \frac{1}{p}$$

and again both probabilities are equal. This completes the proof. $\square$

## 3 An $n$-out-of-$n$ Secret Sharing Scheme

The above scheme is sometimes referred to as "additive secret sharing". We note that 2-out-of-2 additive secret sharing can easily be extended to any $n$-out-of-$n$ additive secret sharing. The sharing algorithm chooses $n$ strings $(s_1, \ldots, s_n)$ uniformly at random subject to the requirement that $\Sigma_{i=1}^{n} s_i = m \pmod p$ (this can be done by choosing $s_1, \ldots, s_{n-1} \in \mathbb{Z}_p$ uniformly at random, and then setting $s_n = m - \Sigma_{i=1}^{n-1} s_i \pmod p$). The reconstruction algorithm simply adds all the shares modulo $p$. The proof of security is similar to the above, and is omitted.

## 4 A 2-out-of-$n$ Secret Sharing Scheme

Suppose we are given a 2-out-of-2 secret sharing scheme ($\texttt{Share}_{2-2}, \texttt{Reconstruct}_{2-2}$) (e.g., the one we showed above). We want to use it to construct a 2-out-of-$n$ secret sharing scheme, namely sharing the secret among $n$ parties, so that any two of them can reconstruct, but any single party learns nothing about the secret.

A first idea is to just share the secret via a fresh 2-out-of-2 sharing for every possible pair of parties. The new share of each party consists of $n-1$ 2-out-of-2 shares (one for each of the other parties). When two parties get together, they can use the two corresponding 2-out-of-2 shares to reconstruct the secret.

The above idea works (satisfies correctness and security). However, the size of the share for each party is $(n-1)$ times the 2-out-of-2 share size, so at least $(n-1)|m|$ for a secret of size $|m|$. Can we do better?

In the above, we shared the secret afresh $n$ times. The idea for the improved 2-out-of-$n$ scheme is to have a smaller number of 2-out-of-2 sharings, as long as we have the property that any pair of parties has at least one complete pair of two corresponding shares (or "two halves" from the same sharing). We can achieve this property with only $\log n$ 2-out-of-2 sharings, where each party gets one half of each of the $\log n$ pairs of shares, corresponding to the binary representation of the party's index. The resulting share size will be $\log n$ times the 2-out-of-2 share size, so we can do this with $(\log n)|m|$ share size. The scheme is specified below (we renamed the parties to $P_0, \ldots, P_{n-1}$ for convenience).

`Share`$_{2-n}$: On input $m \in \mathbb{Z}_p$,

- For $k = 1$ to $\log n$
  - Run `Share`$_{2-2}(m) \to (s_0^k, s_1^k)$
- For $i = 0$ to $n-1$, if binary representation of $i$ is $i_1 \dots i_{\log n}$, set $S_i = (i, s_{i_1}^1, \dots, s_{i_{\log n}}^{\log n})$
- Output $(S_0, \dots, S_{n-1})$.

`Reconstruct`$_{2-n}$: On input $(S_i, S_j)$,

- Consider the binary representations of the indices $i = i_1 \dots i_{\log n}$ and $j = j_1 \dots j_{\log n}$.
- Find a bit position $k$ where they differ, namely $i_k \neq j_k$ (thus, $s_{i_k}^k = s_0^k, s_{j_k}^k = s_1^k$ or vice versa).
- Run `Reconstruct`$_{2-2}(s_0^k, s_1^k)$ and output the same.

**Theorem 2.** *If* (`Share`$_{2-2}$, `Reconstruct`$_{2-2}$) *is a correct and secure 2-out-of-2 secret sharing scheme, then the scheme* (`Share`$_{2-n}$, `Reconstruct`$_{2-n}$) *specified above is a correct and secure 2-out-of-$n$ secret sharing scheme.*

We do not provide the proof here, but the intuition is as follows. Correctness follows from the fact that any $i \neq j$ has at least one bit where $i_k \neq j_k$. The parties $P_i, P_j$ thus have shares $s_0^k, s_1^k$ and can use them to reconstruct the secret.

For example, party $P_{1001}$ holds the shares $s_1^1, s_0^2, s_0^3, s_1^4$ and party $P_{1010}$ holds the shares $s_1^1, s_0^2, s_1^3, s_0^4$, so if they get together, they can use the third element and run `Reconstruct`$_{2-2}(s_0^3, s_1^3)$ to obtain the secret $m$ (they could also have used the fourth element).

Security follows since any individual party is holding $\log n$ shares, where each one is an individual share from an independent 2-out-of-2 sharing. Thus, the share of an individual party is independent of the secret $m$. A formal proof can be shown by reduction to the security of the 2-out-of-2 scheme, and is omitted here.

## Note: Shamir's $t$-out-of-$n$ Secret-Sharing

This was not covered in class (and won't be covered here), but we mention it just for completeness. As mentioned on HW6, there's a secret sharing scheme that allows share size $\max(|m|, \log n)$, and can work for any $t$-out-of-$n$ secret sharing. This is Shamir's secret sharing scheme (which uses Reed-Solomon error correcting codes). While this scheme works for any $t \leq n$, if $t \leq n/3$, the scheme enjoys another important property of *robustness.* Specifically, it can be used with a reconstruction algorithm by Berlekamp-Welch, which allows to reconstruct the secret from all $n$ shares, even if $t-1$ of them are wrong shares contributed by malicious parties.

## 5 Using Secret Sharing for Secure Computation

This part was covered very informally in class (and will similarly be covered informally here). It will not be considered part of the class material (and in particular, will not be on the final).

Secure computation is a major area of cryptography, with lots of work starting in the 1980s, and still going strong. Very roughly speaking, parties $P_1, \ldots, P_n$ want to compute some function $f(x_1, \ldots, x_n)$ of their respective inputs, so that no information is revealed other than the output. Defining security is a challenging and interesting area on its own, and we won't discuss it here. But we will try to give a flavor of how secret sharing can be useful.

Consider $n$ parties, where each party $P_j$ holds an input $x_j \in \mathbb{Z}_p$. They want to compute the function $f(x_1, \ldots, x_n) = \Sigma_{j=1}^n x_j \pmod p$ (note that if we know that $p$ is certainly larger than the sum of the inputs, then this is the same as summing over the integers. This can be used to compute things like average score on an exam, or an average salary, etc).

One protocol we proposed is the following. $P_1$ starts by choosing $r \in \mathbb{Z}_p$ uniformly at random, and sending $t_1 = r + x_1 \pmod p$ to $P_2$ (over a private communication channel). Next, $P_2$ sends $t_2 = t_1 + x_2 \pmod p$ to $P_3$, and so on. Finally, $P_n$, after receiving $t_{n-1}$ from $P_{n-1}$, sends $t_n = t_{n-1} + x_n \pmod p$ to $P_1$. $P_1$ computes $t_n - r \pmod p$ to obtain the final sum (which it can then announce to the other parties).

In this protocol, the message $t_j$ received by party $j$ is uniformly distributed in $\mathbb{Z}_p$, and thus does not reveal any information about the inputs. However, if two parties collude, security is compromised. For example, if $P_1$ and $P_3$ collude, they can find the input of $P_2$ by computing $x_2 = t_2 - t_1 \pmod p$. The scheme also requires $n$ sequential rounds of communication.

We can use $n$-out-of-$n$ secret sharing to solve both above problems. The new protocol proceeds as follows. Each party $P_j$ creates an additive $n$-out-of-$n$ secret sharing of its input $x_j$ into shares $(s_{j,1}, \ldots, s_{j,n})$, where $\Sigma_{i=1}^n s_{j,i} = x_j$ (everything is in $\mathbb{Z}_p$ and we omit the $\pmod p$ notation from now on). $P_j$ then sends share $s_{j,i}$ to party $P_i$. After all parties sent the shares (which can be done in parallel), each party $P_i$ sums up all the shares it has received, to obtain $S_i = \Sigma_{j=1}^n s_{j,i}$. Now every party $P_i$ can broadcast $S_i$, and all parties compute $\Sigma_{i=1}^n S_i$ which equals $\Sigma_{j,i} s_{j,i} = \Sigma_{j=1}^n x_j$ as desired.

The intuition for security is that the shares received by each party $P_i$ are just a random sharing of $S_i$, where the $S_i$'s themselves are also all random, subject to the constraint that they have to sum up to the correct output. Thus, any collusion of fewer than $n$ parties only has random values that are independent of the honesty parties' inputs.

What about secure computation of more complex functions, that are not just sums? One of the general approaches, is to represent the function as an arithmetic circuit with addition and multiplication gates (in $\mathbb{Z}_p$). We start by sharing the inputs (as above), and then the parties jointly compute each gate in the circuit, in a way that preserves secret sharing. That is, given shared inputs to a gate, the parties obtain shared outputs of the gate, all the way until they obtain shares of the output. Then they announce the shares and reconstruct the final output. To do this for an addition gate is easy, as we saw above – the parties just locally sum their shares. Doing this for multiplication gates is more complex, and requires some further interaction. But it is possible to do (in various settings).