

3

Secret Sharing

DNS is the system that maps human-memorable Internet domains like `irs.gov` to machine-readable IP addresses like `166.123.218.220`. If an attacker can masquerade as the DNS system and convince your computer that `irs.gov` actually resides at some other IP address, it's not hard to imagine the kind of trouble you might be facing.

To protect against these kinds of attacks, a replacement called DNSSEC is slowly being rolled out. DNSSEC uses cryptography to make it impossible to falsify a domain-name mapping. The cryptography required to authenticate DNS mappings is certainly interesting, but an even more fundamental question is, who can be trusted with the master cryptographic keys to the system? The non-profit organization in charge of these kinds of things (ICANN) has chosen the following system. The master key is split into 7 pieces and distributed on smart cards to 7 geographically diverse people, who keep them in safe-deposit boxes.

At least five key-holding members of this fellowship would have to meet at a secure data center in the United States to reboot [DNSSEC] in case of a very unlikely system collapse.

"If you round up five of these guys, they can decrypt [the root key] should the West Coast fall in the water and the East Coast get hit by a nuclear bomb," Richard Lamb, program manager for DNSSEC at ICANN, told TechNewsDaily.¹

How is it possible that *any* 5 out of the 7 key-holders can reconstruct the master key, but (presumably) 4 out of the 7 cannot? The solution lies in a cryptographic tool called a **secret-sharing scheme**, the topic of this chapter.

3.1 Definitions

Definition 3.1
(Secret-sharing)

A *t-out-of-n threshold secret-sharing scheme (TSSS)* consists of the following algorithms:

- **Share**: a randomized algorithm that takes a **message** $m \in \mathcal{M}$ as input, and outputs a sequence $s = (s_1, \dots, s_n)$ of **shares**.
- **Reconstruct**: a deterministic algorithm that takes a collection of t or more shares as input, and outputs a message.

We call \mathcal{M} the **message space** of the scheme, and t its **threshold**. As usual, we refer to the parameters/components of a scheme Σ as $\Sigma.t$, $\Sigma.n$, $\Sigma.\mathcal{M}$, $\Sigma.\text{Share}$, $\Sigma.\text{Reconstruct}$.

The **correctness** property for such a scheme is that any subset of at least t shares is enough to reconstruct the message m . That is, for all sets $U \subseteq \{1, \dots, n\}$ with $|U| \geq t$ and for all $s \leftarrow \text{Share}(m)$, we have $\text{Reconstruct}((s_i)_{i \in U}) = m$.

¹<http://www.livescience.com/6791-internet-key-holders-insurance-cyber-attack.html>

In typical usage, the n shares are distributed to n different users. In the above definition, the set U corresponds to a set of users. If $|U| \geq t$, we say that the set is **authorized**; otherwise the set is **unauthorized**.

Just like the encryption definition does not address the problem of key distribution, the secret-sharing definition does not address the problem of *who* should run the Share algorithm (if its input m is so secret that it cannot be entrusted to any single person), nor of *how* the shares are meant to be sent securely to the n different users. Those concerns are considered out of scope by the problem of secret-sharing (although we later discuss clever approaches to the first problem). Rather, the focus is simply on whether it is even possible to encode data in such a way that it can be recovered (only) if a threshold number of shares are present.

Security Definition

We'd like a security guarantee that says something like:

if you have an unauthorized set of shares, then you learn no information about the choice of secret message.

To translate this informal statement into a formal security definition, we follow the The Central Principle of Security DefinitionsTM and define two libraries that allow the calling program to learn a set of shares (for an *unauthorized* set), and that differ only in which secret is shared. If the two libraries are interchangeable, then we conclude that seeing an unauthorized set of shares leaks no information about the choice of secret message. The definition looks like this:

Definition 3.2
(TSSS security)

Let Σ be a threshold secret-sharing scheme. We say that Σ is **secure** if $\mathcal{L}_{\text{tsss-L}}^\Sigma \equiv \mathcal{L}_{\text{tsss-R}}^\Sigma$, where:

$\mathcal{L}_{\text{tsss-L}}^\Sigma$	$\mathcal{L}_{\text{tsss-R}}^\Sigma$
$\text{QUERY}(m_L, m_R \in \Sigma.\mathcal{M}, U):$ <hr/> if $ U \geq \Sigma.t$: return err $s \leftarrow \Sigma.\text{Share}(m_L)$ return $(s_i)_{i \in U}$	$\text{QUERY}(m_L, m_R \in \Sigma.\mathcal{M}, U):$ <hr/> if $ U \geq \Sigma.t$: return err $s \leftarrow \Sigma.\text{Share}(m_R)$ return $(s_i)_{i \in U}$

In an attempt to keep the notation uncluttered, we have not written the type of the argument U , which is $U \subseteq \{1, \dots, \Sigma.n\}$.

Discussion

- Similar to the definition of one-time secrecy of encryption, we let the calling program choose the two secret messages that will be shared. As before, this models the fact that an adversary may have partial knowledge or influence over what inputs might be used in the secret-sharing scheme.
- The calling program also chooses the set U of users' shares to obtain. The libraries make it impossible for the calling program to obtain the shares of an *authorized* set (returning **err** in that case).

- Consider a 6-out-of-10 threshold secret-sharing scheme. With the libraries above, the calling program can receive the shares of users $\{1, \dots, 5\}$ (an unauthorized set) in one call to `QUERY`, and then receive the shares of users $\{6, \dots, 10\}$ in another call. It might seem like the calling program can then combine these shares to reconstruct the secret (if the same message was shared in both calls). However, this is *not* the case because these two sets of shares came from two *independent executions* of the Share algorithm. Shares generated by one call to Share should not be expected to function with shares generated by another call, even if both calls to Share used the same secret message.
- The calling program sees shares corresponding to the *same* set of users in both libraries. Recall that interchangeable libraries hide their internal *differences*. In this case, the set of users is the same in the two libraries, so the security definition does not require shares to hide the *identity* of the users they belong to. Only the secret *message* needs to be hidden according to this definition. Indeed, if each user's identity were appended to his/her corresponding share, it would have no effect on whether $\mathcal{L}_{\text{tsss-L}}^\Sigma \equiv \mathcal{L}_{\text{tsss-R}}^\Sigma$.

One could certainly write a security definition that required shares to be “anonymous” in this way, hiding the identity of their associated user. This could be accomplished by having the calling program provide two sets U_L and U_R , with the two libraries using different sets.² The result would be a perfectly reasonable definition, it just wouldn't be the most standard one used to describe secret sharing. In many applications of secret sharing, it is in fact very convenient to give different users very different-looking kinds of values as their shares.

3.2 A Simple 2-out-of-2 Scheme

Believe it or not, we have already seen a simple secret-sharing scheme! In fact, it might even be best to think of one-time pad as the simplest secret-sharing scheme, since by itself it is not so useful for encryption.

Construction 3.3
(2-out-of-2 TSSS)

$\mathcal{M} = \{0, 1\}^\ell$	<u>Share(m):</u>	<u>Reconstruct(s_1, s_2):</u>
$t = 2$	$s_1 \leftarrow \{0, 1\}^\ell$	<u>return $s_1 \oplus s_2$</u>
$n = 2$	$s_2 := s_1 \oplus m$	
	return (s_1, s_2)	

Since it's a 2-out-of-2 scheme, the only authorized set of users is $\{1, 2\}$, so we write Reconstruct to expect as input both shares s_1 and s_2 . Correctness follows easily from what we've already learned about the properties of XOR.

Theorem 3.4 *Construction 3.3 is a secure 2-out-of-2 threshold secret-sharing scheme.*

Proof Let Σ denote Construction 3.3. We will show that $\mathcal{L}_{\text{tsss-L}}^\Sigma \equiv \mathcal{L}_{\text{tsss-R}}^\Sigma$ using a hybrid proof.

²These two sets would have to be the same size, since we cannot disguise the *number* of shares being returned by the `QUERY` subroutine.

$\mathcal{L}_{\text{tsss-L}}^\Sigma$:

$$\mathcal{L}_{\text{tsss-L}}^\Sigma$$

$$\text{QUERY}(m_L, m_R, U):$$
 if $|U| \geq 2$: return **err**
 $s_1 \leftarrow \{0, 1\}^\ell$
 $s_2 := s_1 \oplus m_L$
 return $(s_i)_{i \in U}$

$$\text{QUERY}(m_L, m_R, U):$$
 if $|U| \geq 2$: return **err**
 if $U = \{1\}$:
 $s_1 \leftarrow \{0, 1\}^\ell$
 $s_2 := s_1 \oplus m_L$
 return s_1
 elseif $U = \{2\}$:
 $s_1 \leftarrow \{0, 1\}^\ell$
 $s_2 := s_1 \oplus m_L$
 return s_2
 else return **null**

$$\text{QUERY}(m_L, m_R, U):$$
 if $|U| \geq 2$: return **err**
 if $U = \{1\}$:
 $s_1 \leftarrow \{0, 1\}^\ell$
 $s_2 := s_1 \oplus m_R$
 return s_1
 elseif $U = \{2\}$:
 $s_1 \leftarrow \{0, 1\}^\ell$
 $s_2 := s_1 \oplus m_L$
 return s_2
 else return **null**

$$\text{QUERY}(m_L, m_R, U):$$
 if $|U| \geq 2$: return **err**
 if $U = \{1\}$:
 $s_1 \leftarrow \{0, 1\}^\ell$
 $s_2 := s_1 \oplus m_R$
 return s_1
 elseif $U = \{2\}$:
 $s_2 \leftarrow \text{QUERY}'(m_L, m_R)$
 return s_2
 else return **null**

$$\mathcal{L}_{\text{ots-L}}^{\text{OTP}}$$

$$\text{QUERY}'(m_L, m_R):$$
 $k \leftarrow \{0, 1\}^\ell$
 $c := k \oplus m_L$
 return c

As usual, the starting point is $\mathcal{L}_{\text{tsss-L}}^\Sigma$, shown here with the details of the secret-sharing scheme filled in (and the types of the subroutine arguments omitted to reduce clutter).

It has no effect on the library's behavior if we duplicate the main body of the library into 3 branches of a new if-statement. The reason for doing so is that the scheme generates s_1 and s_2 differently. This means that our proof will eventually handle the 3 different unauthorized sets ($\{1\}$, $\{2\}$, and \emptyset) in fundamentally different ways.

The definition of s_2 has been changed in the first if-branch. This has no effect on the library's behavior since s_2 is never actually used in this branch.

Recognizing the second branch of the if-statement as a one-time pad encryption (of m_L under key s_1), we factor out the generation of s_2 in terms of the library $\mathcal{L}_{\text{ots-L}}^{\text{OTP}}$ from the one-time secrecy definition. This has no effect on the library's behavior. Importantly, the subroutine in $\mathcal{L}_{\text{ots-L}}^{\text{OTP}}$ expects *two arguments*, so that is what we must pass. We choose to pass m_L and m_R for reasons that should become clear very soon.

```

QUERY( $m_L, m_R, U$ ):
  if  $|U| \geq 2$ : return err
  if  $U = \{1\}$ :
     $s_1 \leftarrow \{0, 1\}^\ell$ 
     $s_2 := s_1 \oplus m_R$ 
    return  $s_1$ 
  elsif  $U = \{2\}$ :
     $s_2 \leftarrow \text{QUERY}'(m_L, m_R)$ 
    return  $s_2$ 
  else return null
    
```

◇

$\mathcal{L}_{\text{ots-R}}^{\text{OTP}}$
$\text{QUERY}'(m_L, m_R)$: $k \leftarrow \{0, 1\}^\ell$ $c := k \oplus m_R$ return c

We have replaced $\mathcal{L}_{\text{ots-L}}^{\text{OTP}}$ with $\mathcal{L}_{\text{ots-R}}^{\text{OTP}}$. From the one-time secrecy of one-time pad (and the composition lemma), this change has no effect on the library's behavior.

```

QUERY( $m_L, m_R, U$ ):
  if  $|U| \geq 2$ : return err
  if  $U = \{1\}$ :
     $s_1 \leftarrow \{0, 1\}^\ell$ 
     $s_2 := s_1 \oplus m_R$ 
    return  $s_1$ 
  elsif  $U = \{2\}$ :
     $s_1 \leftarrow \{0, 1\}^\ell$ 
     $s_2 := s_1 \oplus m_R$ 
    return  $s_2$ 
  else return null
    
```

A subroutine has been inlined; no effect on the library's behavior.

$\mathcal{L}_{\text{tsss-R}}^\Sigma$

$\mathcal{L}_{\text{tsss-R}}^\Sigma$
$\text{QUERY}(m_L, m_R, U)$: if $ U \geq 2$: return err $s_1 \leftarrow \{0, 1\}^\ell$ $s_2 := s_1 \oplus m_R$ return $(s_i)_{i \in U}$

The code has been simplified. Specifically, the branches of the if-statement can all be unified, with no effect on the library's behavior. The result is $\mathcal{L}_{\text{tsss-R}}^\Sigma$.

We showed that $\mathcal{L}_{\text{tsss-L}}^\Sigma \equiv \mathcal{L}_{\text{hyb-1}} \equiv \dots \equiv \mathcal{L}_{\text{hyb-5}} \equiv \mathcal{L}_{\text{tsss-R}}^\Sigma$, and so the secret-sharing scheme is secure. ■

We in fact proved a slightly more general statement. The only property of one-time pad we used was its one-time secrecy. Substituting one-time pad for any other one-time secret encryption scheme would still allow the same proof to go through. So we actually proved the following:

Theorem 3.5 *If Σ is an encryption scheme with one-time secrecy, then the following 2-out-of-2 threshold secret-sharing scheme \mathcal{S} is secure:*

$\mathcal{M} = \Sigma.\mathcal{M}$	<u>Share(m):</u>	<u>Reconstruct(s_1, s_2):</u>
$t = 2$	$s_1 \leftarrow \Sigma.\text{KeyGen}$	
$n = 2$	$s_2 \leftarrow \Sigma.\text{Enc}(s_1, m)$	return $\Sigma.\text{Dec}(s_1, s_2)$
	return (s_1, s_2)	

3.3 Polynomial Interpolation

You are probably familiar with the fact that two points determine a line (in Euclidean geometry). It is also true that 3 points determine a parabola, and so on. The next secret-sharing scheme we discuss is based on the following principle:

$d + 1$ points determine a *unique* degree- d polynomial, and this is true even working modulo a prime.

A note on terminology: If f is a polynomial that can be written as $f(x) = \sum_{i=0}^d f_i x^i$, then we say that f is a **degree- d** polynomial. It would be more technically correct to say that the degree of f is *at most* d since we allow the leading coefficient f_d to be zero. For convenience, we'll stick to saying "degree- d " to mean "degree at most d ".

Theorem 3.6
(Poly Interpolation)

Let p be a prime, and let $\{(x_1, y_1), \dots, (x_{d+1}, y_{d+1})\} \subseteq (\mathbb{Z}_p)^2$ be a set of points whose x_i values are all distinct. Then there is a **unique** degree- d polynomial f with coefficients in \mathbb{Z}_p that satisfies $y_i \equiv_p f(x_i)$ for all i .

Proof

Let $f(x) = \sum_{i=0}^d f_i x^i$ be a degree- d polynomial. Consider the problem of evaluating f on a set of points x_1, \dots, x_{d+1} . We can express the values of f as a linear function in the following way:

$$\begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{d+1}) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & x_2^3 & \cdots & x_2^d \\ & & \vdots & & \ddots & \vdots \\ 1 & x_{d+1} & x_{d+1}^2 & x_{d+1}^3 & \cdots & x_{d+1}^d \end{bmatrix}}_V \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_d \end{bmatrix}.$$

What's notable about this expression is that V is a special kind of matrix called a **Vandermonde** matrix. A Vandermonde matrix is determined by the values x_1, \dots, x_{d+1} . Then the (i, j) entry of the matrix is x_i^{j-1} . One important property of Vandermonde matrices (that we won't prove here) is that the determinant of a square Vandermonde matrix is:

$$\det(V) = \prod_{i < j} (x_j - x_i).$$

The Vandermonde matrix V in our expression is square, having dimensions $(d+1) \times (d+1)$. Also, since all of the x_i values are distinct, the expression for the determinant must be non-zero. That means V is **invertible**!

So, knowing $\{(x_1, y_1), \dots, (x_{d+1}, y_{d+1})\}$, we can solve for the coefficients of f in the following equation:

$$\begin{aligned} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{d+1} \end{bmatrix} &= \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & x_2^3 & \cdots & x_2^d \\ & & \vdots & & \ddots & \vdots \\ 1 & x_{d+1} & x_{d+1}^2 & x_{d+1}^3 & \cdots & x_{d+1}^d \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_d \end{bmatrix} \\ \Rightarrow \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_d \end{bmatrix} &= \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & x_2^3 & \cdots & x_2^d \\ & & \vdots & & \ddots & \vdots \\ 1 & x_{d+1} & x_{d+1}^2 & x_{d+1}^3 & \cdots & x_{d+1}^d \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{d+1} \end{bmatrix}. \end{aligned}$$

Note the matrix inverse in the second equation. There is a unique solution for the vector $f = (f_0, \dots, f_d)$, hence a unique degree- d polynomial f fitting the points.

The proof was written as if the linear algebra was over the real numbers (or complex numbers if you prefer). Indeed, the statement is true for real and complex numbers. However, all of the logic still holds when the linear equations are over \mathbb{Z}_p , when p is a prime. Formally, this is because \mathbb{Z}_p is a *field* (working modulo p , you have addition, multiplication, and *inverses* of nonzero elements). Most of what you know about linear algebra “just works” when matrix operations are in a field. Replacing the “=” sign (integer equality) with “ \equiv_p ” (congruence modulo p), and all the steps of the proof still hold. ■

3.4 Shamir Secret Sharing

Part of the challenge in designing a secret-sharing scheme is making sure that *any* authorized set of users can reconstruct the secret. We have just seen that *any* $d + 1$ points on a degree- d polynomial are enough to reconstruct the polynomial. So a natural approach for secret sharing is to let each user’s share be a point on a polynomial.

That’s exactly what **Shamir secret sharing** does. To share a secret $m \in \mathbb{Z}_p$ with threshold t , we choose a degree- $(t - 1)$ polynomial f that satisfies $f(0) \equiv_p m$, with all other coefficients chosen uniformly in \mathbb{Z}_p . The i th user receives the point $(i, f(i) \% p)$ on the polynomial. The interpolation theorem shows that any t shares can uniquely determine the polynomial f , and hence recover the secret $f(0) \equiv_p m$.

Construction 3.7
(Shamir SSS)

	<u>Share(m):</u>
	$f_1, \dots, f_{t-1} \leftarrow \mathbb{Z}_p$
	$f(x) := m + \sum_{j=1}^{t-1} f_j x^j$
	for $i = 1$ to n :
$\mathcal{M} = \mathbb{Z}_p$	$s_i := (i, f(i) \% p)$
$p : \text{prime}$	return $\mathbf{s} = (s_1, \dots, s_n)$
$n < p$	
$t \leq n$	<u>Reconstruct($(s_i)_{i \in U}$):</u>
	$f(x) := \text{unique degree-}(t-1)$
	polynomial mod p passing
	through points $\{s_i \mid i \in U\}$
	return $f(0)$

Correctness follows from the interpolation theorem. To show the scheme's security, we first show a convenient lemma about the distribution of shares in an unauthorized set:

Lemma 3.8 *Let p be a prime and define $\mathbb{Z}_p^+ \stackrel{\text{def}}{=} \mathbb{Z}_p \setminus \{0\}$. Then following two libraries are interchangeable:*

$\mathcal{L}_{\text{SSSS-real}}$	$\mathcal{L}_{\text{SSSS-real}}$
<u>QUERY($m, t, U \subseteq \mathbb{Z}_p^+$):</u>	<u>QUERY($m, t, U \subseteq \mathbb{Z}_p^+$):</u>
if $ U \geq t$: return err	if $ U \geq t$: return err
$f_1, \dots, f_{t-1} \leftarrow \mathbb{Z}_p$	for $i \in U$:
$f(x) := m + \sum_{j=1}^{t-1} f_j x^j$	$y_i \leftarrow \mathbb{Z}_p$
for $i \in U$:	$s_i := (i, y_i)$
$s_i := (i, f(i) \% p)$	return $(s_i)_{i \in U}$
return $(s_i)_{i \in U}$	

In other words, if we evaluate a uniformly chosen degree $t-1$ polynomial on fewer than t points, the results are (jointly) uniformly distributed.

Proof We will prove the lemma here for the special case where the calling program always provides a set U with $|U| = t-1$. [Exercise 3.4](#) deals with the more general case.

Fix a message $m \in \mathbb{Z}_p$, fix set U of users with $|U| = t-1$, and for each $i \in U$ fix a value $y_i \in \mathbb{Z}_p$. We wish to consider the probability that a call to $\text{QUERY}(m, t, U)$ outputs $((i, y_i))_{i \in U}$, in each of the two libraries.

In library $\mathcal{L}_{\text{SSSS-rand}}$, this event happens with probability $1/p^{t-1}$ since QUERY chooses the $t-1$ different y_i values uniformly in \mathbb{Z}_p .

In library $\mathcal{L}_{\text{SSSS-real}}$, the event happens if and only if the degree- $(t-1)$ polynomial $f(x)$ chosen by QUERY happens to pass through the set of points $\mathcal{P} = \{(i, y_i) \mid i \in U\} \cup \{(0, m)\}$. These are t points with distinct x -coordinates, so by [Theorem 3.6](#) there is a *unique* degree- $(t-1)$ polynomial f with coefficients in \mathbb{Z}_p passing through these points.

The QUERY subroutine picks f uniformly from the set of degree- $(t-1)$ polynomials satisfying $f(0) \equiv_p m$, of which there are p^{t-1} . Exactly one such polynomial causes the event in question, so the probability of the event is $1/p^{t-1}$.

Since the two libraries assign the same probability to all outcomes, we have $\mathcal{L}_{\text{SSSS-real}} \equiv \mathcal{L}_{\text{SSSS-rand}}$. ■

Theorem 3.9 *Shamir's secret-sharing scheme (Construction 3.7) is secure according to Definition 3.2.*

Proof Let \mathcal{S} denote the Shamir secret-sharing scheme. We prove that $\mathcal{L}_{\text{tsss-L}}^{\mathcal{S}} \equiv \mathcal{L}_{\text{tsss-R}}^{\mathcal{S}}$ via a hybrid argument.

$\mathcal{L}_{\text{tsss-L}}^{\mathcal{S}}$:

$\mathcal{L}_{\text{tsss-L}}^{\mathcal{S}}$
$\text{QUERY}(m_L, m_R, U):$ if $ U \geq t$: return err $f_1, \dots, f_{t-1} \leftarrow \mathbb{Z}_p$ $f(x) := m_L + \sum_{j=1}^{t-1} f_j x^j$ for $i \in U$: $s_i := (i, f(i) \% p)$ return $(s_i)_{i \in U}$

Our starting point is $\mathcal{L}_{\text{tsss-L}}^{\mathcal{S}}$, shown here with the details of Shamir secret-sharing filled in.

$\text{QUERY}(m_L, m_R, U):$ return $\text{QUERY}'(m_L, t, U)$

$\mathcal{L}_{\text{ssss-real}}$
$\text{QUERY}'(m, t, U):$ if $ U \geq t$: return err $f_1, \dots, f_{t-1} \leftarrow \mathbb{Z}_p$ $f(x) := m + \sum_{j=1}^{t-1} f_j x^j$ for $i \in U$: $s_i := (i, f(i) \% p)$ return $(s_i)_{i \in U}$

Almost the entire body of the `QUERY` subroutine has been factored out in terms of the $\mathcal{L}_{\text{ssss-real}}$ library defined above. The only thing remaining is the “choice” of whether to share m_L or m_R . Restructuring the code in this way has no effect on the library's behavior.

$\text{QUERY}(m_L, m_R, U):$ return $\text{QUERY}'(m_L, t, U)$

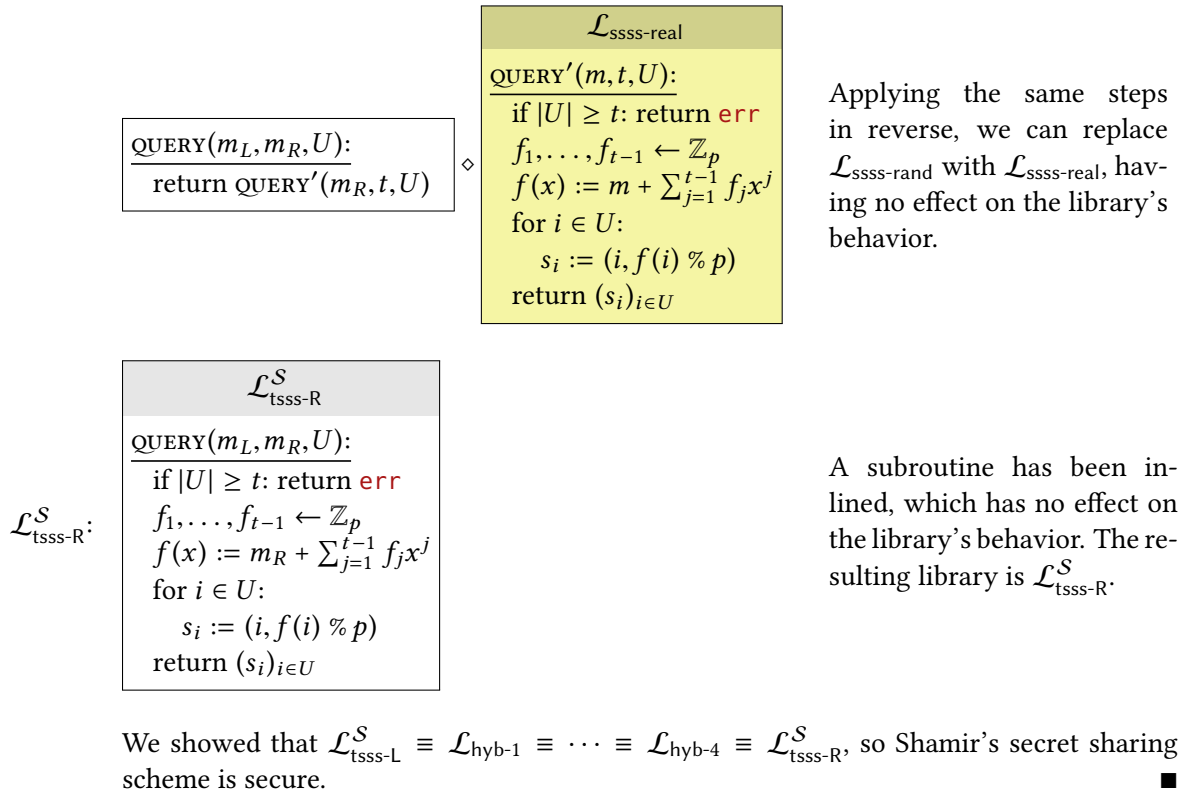
$\mathcal{L}_{\text{ssss-rand}}$
$\text{QUERY}'(m, t, U):$ if $ U \geq t$: return err for $i \in U$: $y_i \leftarrow \mathbb{Z}_p$ $s_i := (i, y_i)$ return $(s_i)_{i \in U}$

By Lemma 3.8, we can replace $\mathcal{L}_{\text{ssss-real}}$ with $\mathcal{L}_{\text{ssss-rand}}$, having no effect on the library's behavior.

$\text{QUERY}(m_L, m_R, U):$ return $\text{QUERY}'(m_R, t, U)$

$\mathcal{L}_{\text{ssss-rand}}$
$\text{QUERY}'(m, t, U):$ if $ U \geq t$: return err for $i \in U$: $y_i \leftarrow \mathbb{Z}_p$ $s_i := (i, y_i)$ return $(s_i)_{i \in U}$

The argument to `QUERY'` has been changed from m_L to m_R . This has no effect on the library's behavior, since `QUERY'` is actually ignoring its argument in these hybrids.



3.5 Sharing Without a Trusted Dealer

to-do

Improve or expand this section. Truly understanding the security of the distributed sharing scheme below requires one to know about the additive n -out-of- n scheme. We've only really talked about 2-out-of-2 (and only with XOR), and deferred n -out-of- n to the exercises. Either prove security of the scheme below (that the resulting shares constitute a secure 3-out-of-5 scheme) or make an exercise about it.

It might also be nice to discuss additive homomorphism feature of secret sharing (currently explored in the exercises). If I'm not careful, I'll end up writing a few chapters about secret-sharing-based MPC, though.

In a secret sharing scheme, it is important that each party see only their designated part of the output of the Share algorithm. Earlier we hinted that it might be difficult for a group of people to find a trustworthy “dealer” to run the Share algorithm on their behalf.

There are clever ways around this, and we'll present one now. Suppose we have 5 parties (numbered 1 through 5) who want to share a randomly chosen secret in \mathbb{Z}_p in a 3-out-of-5 scheme:

1. Each party i (privately) chooses a random share $s_i \leftarrow \mathbb{Z}_p$. The final secret will be $S = s_1 + \dots + s_5 \pmod{p}$, and indeed, all 5 shares are required to recover the secret.
2. Party i distributes its own secret s_i to the other parties using a 3-out-of-4 scheme. We'll call $s_{i \rightarrow j}$ the share of s_i that the party i sends to party j .

3. At the end, party j knows its own s_j and also each $s_{i \rightarrow j}$.

The secret can be reconstructed by any set of 3 parties as follows. Suppose parties $\{1, 2, 3\}$ want to learn the secret:

1. They already know s_1, s_2, s_3 .
2. They can learn s_4 by combining their shares $s_{4 \rightarrow 1}, s_{4 \rightarrow 2}, s_{4 \rightarrow 3}$. After all, these are 3 shares of a 3-out-of-4 sharing of s_4 .
3. Similarly they can learn s_5 by combining their shares $s_{5 \rightarrow 1}, s_{5 \rightarrow 2}, s_{5 \rightarrow 3}$.
4. Now, having all of s_1, \dots, s_5 , they can learn the final secret S .

Exercises

- 3.1. Generalize [Construction 3.3](#) to be an n -out-of- n secret-sharing scheme, and prove that your scheme is correct and secure.
- 3.2. Prove [Theorem 3.5](#).
- 3.3. Suppose T is an invertible $n \times n$ matrix over \mathbb{Z}_p , where p is a prime. Show that the following two libraries are interchangeable:

$$\mathcal{L}_{\text{left}} = \boxed{\begin{array}{l} \text{QUERY}(\mathbf{v} \in (\mathbb{Z}_p)^n): \\ \mathbf{r} \leftarrow (\mathbb{Z}_p)^n \\ \mathbf{z} := \mathbf{v} + T\mathbf{r} \\ \text{return } \mathbf{z} \end{array}}; \quad \mathcal{L}_{\text{right}} = \boxed{\begin{array}{l} \text{QUERY}(\mathbf{v} \in (\mathbb{Z}_p)^n): \\ \mathbf{z} \leftarrow (\mathbb{Z}_p)^n \\ \text{return } \mathbf{z} \end{array}}.$$

- 3.4. The text gives a proof of [Lemma 3.8](#) for the special case where the calling program always calls `QUERY` with $|U| = t - 1$. This exercise shows one way to complete the proof. Define the following “wrapper” library:

$$\mathcal{L}_{\text{wrap}} \stackrel{\text{def}}{=} \boxed{\begin{array}{l} \text{QUERY}(m, t, U \subseteq \mathbb{Z}_p^+): \\ U' := U \\ \text{while } |U'| < t - 1: \\ \quad \text{add an arbitrary element of } \mathbb{Z}_p^+ \setminus U' \text{ to } U' \\ \mathbf{s} \leftarrow \text{QUERY}'(m, t, U') \\ \text{return } (s_i)_{i \in U} \end{array}}.$$

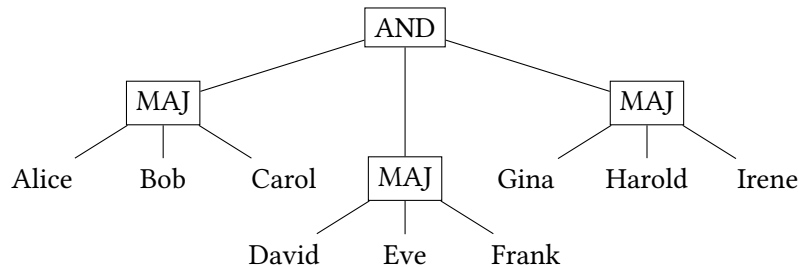
- (a) Argue that $\mathcal{L}_{\text{sss-real}} \equiv \mathcal{L}_{\text{wrap}} \diamond \mathcal{L}'_{\text{sss-real}}$, where on the right-hand side $\mathcal{L}'_{\text{sss-real}}$ refers to $\mathcal{L}_{\text{sss-real}}$ but with its `QUERY` subroutine renamed to `QUERY'`.
- (b) Argue that $\mathcal{L}_{\text{sss-rand}} \equiv \mathcal{L}_{\text{wrap}} \diamond \mathcal{L}'_{\text{sss-rand}}$, with the same interpretation as above.
- (c) Argue that for any calling program \mathcal{A} , the combined program $\mathcal{A} \diamond \mathcal{L}_{\text{wrap}}$ only calls `QUERY'` with $|U| = t - 1$. Hence, the proof presented in the text applies when the calling program has the form $\mathcal{A} \diamond \mathcal{L}_{\text{wrap}}$.

- (d) Combining the previous parts, show that $\mathcal{L}_{\text{SSS-real}} \equiv \mathcal{L}_{\text{SSS-rand}}$ (i.e., the two libraries are interchangeable with respect to *arbitrary* calling programs).
- 3.5. Let \mathcal{S} be a TSSS with $\mathcal{M} = \{0, 1\}^\ell$ and where each share is also a string of bits. Prove that if \mathcal{S} satisfies security then every user's share must be at least ℓ bits long.
- Hint:* Prove the contrapositive. Suppose the first user's share is less than ℓ bits (and that this fact is known to everyone). Show how users 2 through t can violate security by enumerating all possibilities for the first user's share.
- 3.6. n users have shared two secrets using Shamir secret sharing. User i has a share $s_i = (i, y_i)$ of the secret m , and a share $s'_i = (i, y'_i)$ of the secret m' . Both sets of shares use the same prime modulus p .
- Suppose each user i locally computes $z_i = y_i + y'_i \% p$.
- (a) Prove that if the shares of m and shares of m' had the same threshold, then the resulting $\{(i, z_i) \mid i \leq n\}$ are a valid secret-sharing of the secret $m + m'$.
- (b) Describe what the users get when the shares of m and m' had different thresholds (say, t and t' , respectively).
- 3.7. Suppose there are 5 people on a committee: Alice (president), Bob, Charlie, David, Eve. Suggest how they can securely share a secret so that it can only be opened by:
- Alice and any one other person
 - Any three people

Describe in detail how the sharing algorithm works and how the reconstruction works (for all authorized sets of users).

- 3.8. Suppose there are 9 people on an important committee: Alice, Bob, Carol, David, Eve, Frank, Gina, Harold, & Irene. Alice, Bob & Carol form a subcommittee; David, Eve & Frank form another subcommittee; and Gina, Harold & Irene form another subcommittee. Suggest how they can share a secret so that it can only be opened when a majority of each subcommittee is present. Describe why a 6-out-of-9 threshold secret-sharing scheme does **not** suffice.

Hint:



- ★ 3.9. Generalize the previous exercise. A **monotone formula** is a boolean function $\phi : \{0,1\}^n \rightarrow \{0,1\}$ that when written as a formula uses only AND and OR operations (no NOTs). For a set $A \subseteq \{1, \dots, n\}$, let χ_A be the bitstring where whose i th bit is 1 if and only if $i \in A$.

For every monotone formula $\phi : \{0,1\}^n \rightarrow \{0,1\}$, construct a secret-sharing scheme whose authorized sets are $\{A \subseteq \{1, \dots, n\} \mid \phi(\chi_A) = 1\}$. Prove that your scheme is secure.

Hint: express the formula as a tree of AND and OR gates.