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 might result in a bad day for you.

To protect against these kinds of attacks, a replacement called DNSSEC has been proposed. 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 remains: *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," [said] Richard Lamb, program manager for DNSSEC at ICANN.*[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

We begin by introducing the syntax of a secret-sharing scheme:

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 $\boldsymbol{s} = (s_1, \ldots, s_n)$ of **shares**.*
- Reconstruct*: a deterministic algorithm that takes a collection of t or more shares as input, and outputs a message.*

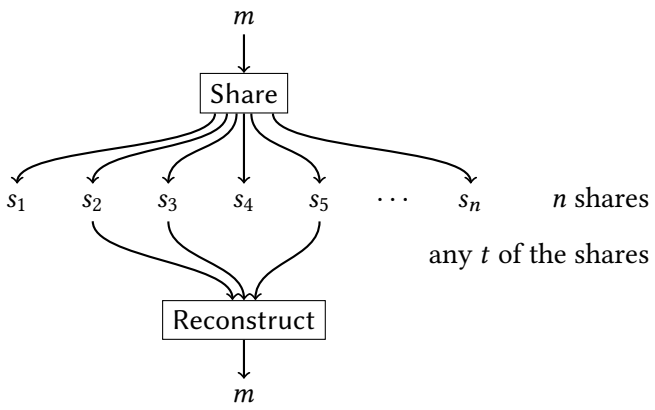
*We call $\mathcal{M}$ the **message space** of the scheme, and t its **threshold**. As usual, we refer to the parameters/components of a scheme Σ as $\Sigma.t$, $\Sigma.n$, $\Sigma.\mathcal{M}$, Σ.Share, Σ.Reconstruct.*

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

In secret-sharing, we number the users as $\{1, \ldots, n\}$, with user i receiving share s_i. Let $U \subseteq \{1, \ldots, n\}$ be a subset of users. Then $\{s_i \mid i \in U\}$ refers to the set of shares belonging to users U. If $|U| \geqslant t$, we say that U is **authorized**; otherwise it is **unauthorized**. The goal of secret sharing is for all authorized sets of users/shares to be able to reconstruct the secret, while all unauthorized sets learn nothing.

Definition 3.2 (TSSS correctness)

*A t-out-of-n TSSS satisfies **correctness** if, for all authorized sets $U \subseteq \{1, \ldots, n\}$ (i.e., $|U| \geqslant t$) and for all $s \leftarrow \text{Share}(m)$, we have $\text{Reconstruct}(\{s_i \mid i \in U\}) = m$.*



Security Definition

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

> *if you know only 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 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 3.3 (TSSS security)

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

$\mathcal{L}^{\Sigma}_{\text{tsss-L}}$
SHARE$(m_L, m_R \in \Sigma.\mathcal{M}, U)$:
if $\lvert U\rvert \geqslant \Sigma.t$: return `err`
$s \leftarrow \Sigma.\text{Share}(m_L)$
return $\{s_i \mid i \in U\}$

$\mathcal{L}^{\Sigma}_{\text{tsss-R}}$
SHARE$(m_L, m_R \in \Sigma.\mathcal{M}, U)$:
if $\lvert U\rvert \geqslant \Sigma.t$: return `err`
$s \leftarrow \Sigma.\text{Share}(m_R)$
return $\{s_i \mid i \in U\}$

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

Discussion & Pitfalls

- 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 an attack scenario in which the adversary has partial knowledge or influence on the secret m being shared.
- 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). This does **not** mean that a user is never allowed to distribute an authorized number of shares (this would be strange indeed, since it would make any future reconstruction impossible). It just means that we want a *security definition* that says something about an attacker who sees only an unauthorized set of shares, so we formalize security in terms of libraries with this restriction.
- Consider a 6-out-of-10 threshold secret-sharing scheme. With the libraries above, the calling program can receive the shares of users $\{1, \ldots, 5\}$ (an unauthorized set) in one call to SHARE, and then receive the shares of users $\{6, \ldots, 10\}$ 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.
- Recall that in our style of defining security using libraries, it is only the internal *differences* between the libraries that must be hidden. Anything that is the *same* between the two libraries need not be hidden. One thing that is the same for the two libraries here is the fact that they output the shares belonging to the same set of users U. This security definition does not require shares to hide *which user they belong to.* Indeed, you can modify a secret-sharing scheme so that each user's identity is appended to his/her corresponding share, and the result would still satisfy the security definition above.
- 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), or *how* the shares should be delivered 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 an unauthorized set of shares gives no information about the secret, while any authorized set completely reveals the secret.

An Insecure Approach

One way to understand the security of secret sharing is to see an example of an "obvious" but insecure approach for secret sharing, and study why it is insecure.

Let's consider a 5-out-of-5 secret-sharing scheme. This means we want to split a secret into 5 pieces so that any 4 of the pieces leak nothing. One way you might think to do this is to *literally chop up the secret* into 5 pieces. For example, if the secret is 500 bits, you might give the first 100 bits to user 1, the second 100 bits to user 2, and so on.

Construction 3.4 (Insecure TSSS)

$\mathcal{M} = \{0,1\}^{500}$, $t = 5$, $n = 5$

Share(m):
split m into $m = s_1 \| \cdots \| s_5$, where each $|s_i| = 100$
return $(s_1, \ldots, s_5)$

Reconstruct($s_1, \ldots, s_5$):
return $s_1 \| \cdots \| s_5$

It is true that the secret can be constructed by concatenating all 5 shares, and so this construction satisfies the correctness property. (The only authorized set is the set of all 5 users, so we write Reconstruct to expect all 5 shares.)

However, the scheme is **insecure** (as promised). Suppose you have even just 1 share. It is true that you don't know the secret *in its entirety,* but the security definition (for 5-out-of-5 secret sharing) demands that a single share reveals *nothing* about the secret. Of course knowing 100 bits of something is not the same as than knowing *nothing* about it.

We can leverage this observation to make a more formal attack on the scheme, in the form of a distinguisher between the two $\mathcal{L}_{\text{tsss-}\star}$ libraries. As an extreme case, we can distinguish between shares of an all-0 secret and shares of an all-1 secret:

$\mathcal{A}$
$s_1 := \text{SHARE}(0^{500}, 1^{500}, \{1\})$
return $s_1 \stackrel{?}{=} 0^{100}$

Let's link this calling program to both of the $\mathcal{L}_{\text{tsss-}\star}$ libraries and see what happens:

$\mathcal{A}$
$s_1 := \text{SHARE}(0^{500}, 1^{500}, \{1\})$
return $s_1 \stackrel{?}{=} 0^{100}$

$\diamond$

$\mathcal{L}_{\text{tsss-L}}$
SHARE(m_L, m_R, U):
if $|U| \geqslant t$: return err
$s \leftarrow$ Share(m_L)
return $\{s_i \mid i \in U\}$

When $\mathcal{A}$ is linked to $\mathcal{L}_{\text{tsss-L}}$, it receives a share of 0^{500}, which will itself be a string of all zeroes. In this case, $\mathcal{A}$ outputs 1 with probability 1.

$\mathcal{A}$
$s_1 := \text{SHARE}(0^{500}, 1^{500}, \{1\})$
return $s_1 \stackrel{?}{=} 0^{100}$

$\diamond$

$\mathcal{L}_{\text{tsss-R}}$
SHARE(m_L, m_R, U):
if $|U| \geqslant t$: return err
$s \leftarrow$ Share(m_R)
return $\{s_i \mid i \in U\}$

When $\mathcal{A}$ is linked to $\mathcal{L}_{\text{tsss-R}}$, it receives a share of 1^{500} which will be a string of all ones. In this case, $\mathcal{A}$ outputs 1 with probability 0.

We have constructed a calling program which behaves very differently (indeed, as differently as possible) in the presence of the two libraries. Hence, this secret-sharing scheme is not secure.

Hopefully this example demonstrates one of the main challenges (and amazing things) about secret-sharing schemes. It is easy to reveal information about the secret *gradually* as

more shares are obtained, like in this insecure example. However, the security definition of secret sharing is that the shares must leak *absolutely no information* about the secret, until the number of shares passes the threshold value.

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.

Construction 3.5 (2-out-of-2 TSSS)

$\mathcal{M} = \{0,1\}^\ell$
$t = 2$
$n = 2$

Share(m):
$s_1 \leftarrow \{0,1\}^\ell$
$s_2 := s_1 \oplus m$
return (s_1, s_2)

Reconstruct(s_1, s_2):
return $s_1 \oplus s_2$

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

Example *If we want to share the string* $m = \texttt{1101010001}$ *then the* Share *algorithm might choose*

$$\begin{aligned} s_1 &:= \texttt{0110000011} \\ s_2 &:= s_1 \oplus m \\ &= \texttt{0110000011} \oplus \texttt{1101010001} = \texttt{1011010010}. \end{aligned}$$

Then the secret can be reconstructed by XORing *the two shares together, via:*

$$s_1 \oplus s_2 = \texttt{0110000011} \oplus \texttt{1011010010} = \texttt{1101010001} = m.$$

Remember that this example shows just one possible *execution of* Share(1101010001)*, but* Share *is a randomized algorithm and many other values of* (s_1, s_2) *are possible.*

Theorem 3.6 *Construction 3.5 is a secure 2-out-of-2 threshold secret-sharing scheme.*

Proof Let Σ denote Construction 3.5. We will show that $\mathcal{L}^\Sigma_{\text{tsss-L}} \equiv \mathcal{L}^\Sigma_{\text{tsss-R}}$ using a hybrid proof.

$\mathcal{L}^\Sigma_{\text{tsss-L}}$:

$\mathcal{L}^\Sigma_{\text{tsss-L}}$
SHARE(m_L, m_R, U):
if $|U| \geqslant 2$: return err
$s_1 \leftarrow \{0,1\}^\ell$
$s_2 := s_1 \oplus m_L$
return $\{s_i \mid i \in U\}$

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

```
SHARE(m_L, m_R, U):
  if |U| ⩾ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return {s_1}
  elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return {s_2}
  else return ∅
```

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.

```
SHARE(m_L, m_R, U):
  if |U| ⩾ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return {s_1}
  elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_L
    return {s_2}
  else return ∅
```

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.

```
SHARE(m_L, m_R, U):
  if |U| ⩾ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return {s_1}
  elsif U = {2}:
    s_2 ← EAVESDROP(m_L, m_R)
    return {s_2}
  else return ∅
```

◇

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

```
EAVESDROP(m_L, m_R):
  k ← {0,1}^ℓ
  c := k ⊕ m_L
  return c
```

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 $\mathcal{L}^{\text{OTP}}_{\text{ots-L}}$ from the one-time secrecy definition. This has no effect on the library's behavior. Importantly, the subroutine in $\mathcal{L}^{\text{OTP}}_{\text{ots-L}}$ 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.

```
SHARE(m_L, m_R, U):
  if |U| ⩾ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return {s_1}
  elsif U = {2}:
    s_2 ← EAVESDROP(m_L, m_R)
    return {s_2}
  else return ∅
```

◇

```
L^OTP_ots-R
EAVESDROP(m_L, m_R):
  k ← {0,1}^ℓ
  c := k ⊕ m_R
  return c
```

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

```
SHARE(m_L, m_R, U):
  if |U| ⩾ 2: return err
  if U = {1}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return {s_1}
  elsif U = {2}:
    s_1 ← {0,1}^ℓ
    s_2 := s_1 ⊕ m_R
    return {s_2}
  else return ∅
```

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

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

```
L^Σ_tsss-R
SHARE(m_L, m_R, U):
  if |U| ⩾ 2: return err
  s_1 ← {0,1}^ℓ
  s_2 := s_1 ⊕ m_R
  return {s_i | i ∈ 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 $\mathcal{L}^{\Sigma}_{\text{tsss-R}}$.

We showed that $\mathcal{L}^{\Sigma}_{\text{tsss-L}} \equiv \mathcal{L}_{\text{hyb-1}} \equiv \cdots \equiv \mathcal{L}_{\text{hyb-5}} \equiv \mathcal{L}^{\Sigma}_{\text{tsss-R}}$, 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 3.7 *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:*

$\mathcal{M} = \Sigma.\mathcal{M}$, $t = 2$, $n = 2$

Share(m):
$s_1 \leftarrow \Sigma.\mathsf{KeyGen}$
$s_2 \leftarrow \Sigma.\mathsf{Enc}(s_1, m)$
return (s_1, s_2)

Reconstruct(s_1, s_2):
return $\Sigma.\mathsf{Dec}(s_1, s_2)$

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.

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."

Polynomials Over the Reals

Theorem 3.8 (Poly Interpolation) *Let $\{(x_1, y_1), \ldots, (x_{d+1}, y_{d+1})\} \subseteq \mathbb{R}^2$ be a set of points whose x_i values are all distinct. Then there is a **unique** degree-d polynomial f with real coefficients that satisfies $y_i = f(x_i)$ for all i.*

Proof To start, consider the following polynomial:

$$\ell_1(\boldsymbol{x}) = \frac{(\boldsymbol{x} - x_2)(\boldsymbol{x} - x_3)\cdots(\boldsymbol{x} - x_{d+1})}{(x_1 - x_2)(x_1 - x_3)\cdots(x_1 - x_{d+1})}.$$

The notation is potentially confusing. ℓ_1 is a polynomial with formal variable $\boldsymbol{x}$ (written in bold). The non-bold x_i values are just plain numbers (scalars), given in the theorem statement. Therefore the numerator in ℓ_1 is a degree-d polynomial in $\boldsymbol{x}$. The denominator is just a scalar, and since all of the x_i's are distinct, we are not dividing by zero. Overall, ℓ_1 is a degree-d polynomial.

What happens when we evaluate ℓ_1 at one of the special x_i values?

- Evaluating $\ell_1(x_1)$ makes the numerator and denominator the same, so $\ell_1(x_1) = 1$.
- Evaluating $\ell_1(x_i)$ for $i \neq 1$ leads to a term $(x_i - x_i)$ in the numerator, so $\ell_1(x_i) = 0$.

Of course, ℓ_1 can be evaluated at any point (not just the special points $x_1, \ldots, x_{d+1}$), but we don't care about what happens in those cases.

We can similarly define other polynomials ℓ_j:

$$\ell_j(\boldsymbol{x}) = \frac{(\boldsymbol{x} - x_1)\cdots(\boldsymbol{x} - x_{j-1})(\boldsymbol{x} - x_{j+1})\cdots(\boldsymbol{x} - x_{d+1})}{(x_j - x_1)\cdots(x_j - x_{j-1})(x_j - x_{j+1})\cdots(x_j - x_{d+1})}.$$

The pattern is that the numerator is "missing" the term $(\boldsymbol{x} - x_j)$ and the denominator is missing the term $(x_j - x_j)$, because we don't want a zero in the denominator. Polynomials

of this kind are called **LaGrange polynomials**. They are each degree-d polynomials, and they satisfy the property:

$$\ell_j(x_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Now consider the following polynomial:

$$f(\boldsymbol{x}) = y_1\ell_1(\boldsymbol{x}) + y_2\ell_2(\boldsymbol{x}) + \cdots + y_{d+1}\ell_{d+1}(\boldsymbol{x}).$$

Note that f is a degree-d polynomial since it is the sum of degree-d polynomials (again, the y_i values are just scalars).

What happens when we evaluate f on one of the special x_i values? Since $\ell_i(x_i) = 1$ and $\ell_j(x_i) = 0$ for $j \neq i$, we get:

$$\begin{aligned} f(x_i) &= y_1\ell_1(x_i) + \cdots + y_i\ell_i(x_i) + \cdots + y_{d+1}\ell_{d+1}(x_i) \\ &= y_1 \cdot 0 + \cdots + y_i \cdot 1 + \cdots + y_{d+1} \cdot 0 \\ &= y_i \end{aligned}$$

So $f(x_i) = y_i$ for every x_i, which is what we wanted. This shows that there is *some* degree-d polynomial with this property.

Now let's argue that this f is unique. Suppose there are two degree-d polynomials f and f' such that $f(x_i) = f'(x_i) = y_i$ for $i \in \{1, \ldots, d+1\}$. Then the polynomial $g(\boldsymbol{x}) = f(\boldsymbol{x}) - f'(\boldsymbol{x})$ also is degree-d, and it satisfies $g(x_i) = 0$ for all i. In other words, each x_i is a *root* of g, so g has at least $d+1$ roots. But the only degree-d polynomial with $d+1$ roots is the identically-zero polynomial $g(\boldsymbol{x}) = 0$. If $g(\boldsymbol{x}) = 0$ then $f = f'$. In other words, any degree-d polynomial f' that satisfies $f'(x_i) = y_i$ must be equal to f. So f is the unique polynomial with this property. ■

Example *Let's figure out the degree-3 polynomial that passes through the points* $(3,1), (4,1), (5,9), (2,6)$*:*

i	1	2	3	4
x_i	3	4	5	2
y_i	1	1	9	6

First, let's construct the appropriate LaGrange polynomials:

$$\ell_1(\boldsymbol{x}) = \frac{(\boldsymbol{x}-x_2)(\boldsymbol{x}-x_3)(\boldsymbol{x}-x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)} = \frac{(\boldsymbol{x}-4)(\boldsymbol{x}-5)(\boldsymbol{x}-2)}{(3-4)(3-5)(3-2)} = \frac{\boldsymbol{x}^3 - 11\boldsymbol{x}^2 + 38\boldsymbol{x} - 40}{2}$$

$$\ell_2(\boldsymbol{x}) = \frac{(\boldsymbol{x}-x_1)(\boldsymbol{x}-x_3)(\boldsymbol{x}-x_4)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)} = \frac{(\boldsymbol{x}-3)(\boldsymbol{x}-5)(\boldsymbol{x}-2)}{(4-3)(4-5)(4-2)} = \frac{\boldsymbol{x}^3 - 10\boldsymbol{x}^2 + 31\boldsymbol{x} - 30}{-2}$$

$$\ell_3(\boldsymbol{x}) = \frac{(\boldsymbol{x}-x_1)(\boldsymbol{x}-x_2)(\boldsymbol{x}-x_4)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)} = \frac{(\boldsymbol{x}-3)(\boldsymbol{x}-4)(\boldsymbol{x}-2)}{(5-3)(5-4)(5-2)} = \frac{\boldsymbol{x}^3 - 9\boldsymbol{x}^2 + 26\boldsymbol{x} - 24}{6}$$

$$\ell_4(\boldsymbol{x}) = \frac{(\boldsymbol{x}-x_1)(\boldsymbol{x}-x_2)(\boldsymbol{x}-x_3)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)} = \frac{(\boldsymbol{x}-3)(\boldsymbol{x}-4)(\boldsymbol{x}-5)}{(2-3)(2-4)(2-5)} = \frac{\boldsymbol{x}^3 - 12\boldsymbol{x}^2 + 47\boldsymbol{x} - 60}{-6}$$

As a sanity check, notice how:

$$\ell_1(x_1) = \ell_1(3) = \frac{3^3 - 11 \cdot 3^2 + 38 \cdot 3 - 40}{2} = \frac{2}{2} = 1$$

$$\ell_1(x_2) = \ell_1(4) = \frac{4^3 - 11 \cdot 4^2 + 38 \cdot 4 - 40}{2} = \frac{0}{2} = 0$$

It will make the next step easier if we rewrite all LaGrange polynomials to have the same denominator 6:

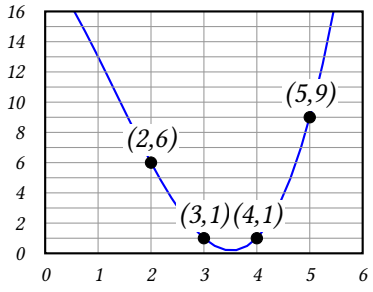
$$\ell_1(x) = \frac{3x^3 - 33x^2 + 114x - 120}{6} \qquad \ell_3(x) = \frac{x^3 - 9x^2 + 26x - 24}{6}$$

$$\ell_2(x) = \frac{-3x^3 + 30x^2 - 93x + 90}{6} \qquad \ell_4(x) = \frac{-x^3 + 12x^2 - 47x + 60}{6}$$

Our desired polynomial is

$$\begin{aligned} f(x) &= y_1 \cdot \ell_1(x) + y_2 \cdot \ell_2(x) + y_3 \cdot \ell_3(x) + y_4 \cdot \ell_4(x) \\ &= 1 \cdot \ell_1(x) + 1 \cdot \ell_2(x) + 9 \cdot \ell_3(x) + 6 \cdot \ell_4(x) \\ &= \frac{1}{6}\left(\begin{array}{rrrr} 1 \cdot (& 3x^3 & - 33x^2 & + 114x & - 120) \\ + 1 \cdot (& - 3x^3 & + 30x^2 & - 93x & + 90) \\ + 9 \cdot (& x^3 & - 9x^2 & + 26x & - 24) \\ + 6 \cdot (& - x^3 & + 12x^2 & - 47x & + 60) \end{array}\right) \\ &= \frac{1}{6}\left(3x^3 - 12x^2 - 27x + 114\right) \\ &= \frac{x^3}{2} - 2x^2 - \frac{9x}{2} + 19 \end{aligned}$$

And indeed, f gives the correct values:



$$f(x_1) = f(3) = \frac{3^3}{2} - 2 \cdot 3^2 - \frac{9 \cdot 3}{2} + 19 = 1 = y_1$$

$$f(x_2) = f(4) = \frac{4^3}{2} - 2 \cdot 4^2 - \frac{9 \cdot 4}{2} + 19 = 1 = y_2$$

$$f(x_3) = f(5) = \frac{5^3}{2} - 2 \cdot 5^2 - \frac{9 \cdot 5}{2} + 19 = 9 = y_3$$

$$f(x_4) = f(2) = \frac{2^3}{2} - 2 \cdot 2^2 - \frac{9 \cdot 2}{2} + 19 = 6 = y_4$$

Polynomials mod p

We will see a secret-sharing scheme based on polynomials, whose Share algorithm must choose a polynomial with uniformly random coefficients. Since we cannot have a uniform distribution over the real numbers, we must instead consider polynomials with coefficients in $\mathbb{Z}_p$.

It is still true that $d+1$ points determine a unique degree-d polynomial when working modulo p, *if p is a prime!*

Theorem 3.9 (Interp mod p)

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

The proof is the same as the one for Theorem 3.8, if you interpret all arithmetic modulo p. Addition, subtraction, and multiplication mod p are straight forward; the only non-trivial question is how to interpret "division mod p," which is necessary in the definition of the ℓ_j polynomials. For now, just accept that you can always "divide" mod p (except by zero) when p is a prime. If you are interested in how division mod p works, look ahead to Chapter 13.

We can also generalize the observation that $d+1$ points uniquely determine a degree-d polynomial. It turns out that:

> For any k points, there are exactly p^{d+1-k} polynomials of degree-d that hit those points, mod p.

Note how when $k = d + 1$, the statement says that there is just a single polynomial hitting the points.

Corollary 3.10 (# of polys)

Let $\mathcal{P} = \{(x_1, y_1), \ldots, (x_k, y_k)\} \subseteq (\mathbb{Z}_p)^2$ be a set of points whose x_i values are distinct. Let d satisfy $k \leqslant d + 1$ and $p > d$. Then the number of degree-d polynomials f with coefficients in $\mathbb{Z}_p$ that satisfy the condition $y_i \equiv_p f(x_i)$ for all i is exactly p^{d+1-k}.

Proof

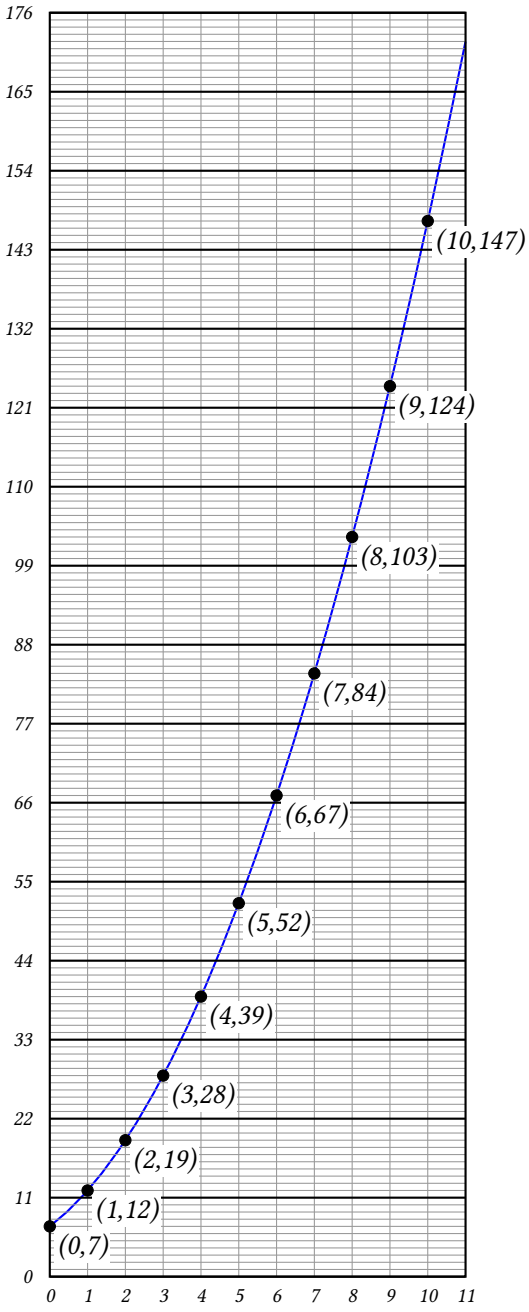
The proof is by induction on the value $d + 1 - k$. The base case is when $d + 1 - k = 0$. Then we have $k = d + 1$ distinct points, and Theorem 3.9 says that there is a *unique* polynomial satisfying the condition. Since $p^{d+1-k} = p^0 = 1$, the base case is true.

For the inductive case, we have $k \leqslant d$ points in $\mathcal{P}$. Let $x^* \in \mathbb{Z}_p$ be a value that does not appear as one of the x_i's. Every polynomial must give *some* value when evaluated at x^*. So,

$$
\begin{aligned}
&[\#\text{ of degree-}d\text{ polynomials passing through points in } \mathcal{P}] \\
&\quad = \sum_{y^* \in \mathbb{Z}_p} [\#\text{ of degree-}d\text{ polynomials passing through points in } \mathcal{P} \cup \{(x^*, y^*)\}] \\
&\quad \overset{(\star)}{=} \sum_{y^* \in \mathbb{Z}_p} p^{d+1-(k+1)} \\
&\quad = p \cdot \left(p^{d+1-k-1}\right) = p^{d+1-k}
\end{aligned}
$$

The equality marked ($\star$) follows from the inductive hypothesis, since each of the terms involves a polynomial passing through a specified set of $k + 1$ points with distinct x-coordinates. ∎

Example

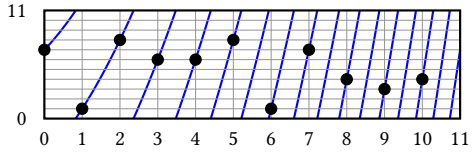


What does a "polynomial mod p" look like? *Consider an example degree-2 polynomial:*

$$f(x) = x^2 + 4x + 7$$

When we plot this polynomial over the real numbers (the picture on the left), we get a familiar parabola.

*Let's see what this polynomial "looks like" modulo 11 (*i.e.*, in $\mathbb{Z}_{11}$). Working mod 11 means to "wrap around" every time the polynomial crosses over a multiple of 11 along the y-axis. This results in the blue plot below:*



This is a picture of a mod-11 parabola. In fact, since we care only about $\mathbb{Z}_{11}$ inputs to f, you could rightfully say that ***just the 11 highlighted points alone*** *(not the blue curve) are a picture of a mod-11 parabola.*

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 uniquely 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, first 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 ith user receives the point $(i, f(i) \% p)$ on the polynomial. The interpolation theorem says that *any* t of the shares can uniquely determine the polynomial f, and hence recover the secret $f(0)$.

Construction 3.11 (Shamir SSS)

$\mathcal{M} = \mathbb{Z}_p$
p : *prime*
$n < p$
$t \leqslant n$

Share(m):
  $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
  $f(x) := m + \sum_{j=1}^{t-1} f_j x^j$
  for $i = 1$ to n:
    $s_i := (i, f(i) \% p)$
  return $s = (s_1, \ldots, s_n)$

Reconstruct($\{s_i \mid i \in U\}$):
  $f(x) :=$ 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.

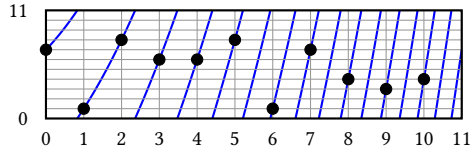
Example

Here is an example of 3-out-of-5 secret sharing over $\mathbb{Z}_{11}$ *(so* $p = 11$*). Suppose the secret being shared is* $m = 7 \in \mathbb{Z}_{11}$*. The* Share *algorithm chooses a random degree-2 polynomial with constant coefficient 7.*

Let's say that the remaining two coefficients are chosen as $f_2 = 1$ *and* $f_1 = 4$*, resulting in the following polynomial:*

$$f(x) = 1\,x^2 + 4\,x + 7$$

This is the same polynomial illustrated in the previous example:



For each user $i \in \{1, \ldots, 5\}$*, we distribute the share* $(i, f(i) \% 11)$*. These shares correspond to the highlighted points in the mod-11 picture above.*

user (i)	$f(i)$	*share* $(i, f(i) \% 11)$
1	$f(1) = 12$	$(1, 1)$
2	$f(2) = 19$	$(2, 8)$
3	$f(3) = 28$	$(3, 6)$
4	$f(4) = 39$	$(4, 6)$
5	$f(5) = 52$	$(5, 8)$

Remember that this example illustrates just one possible execution of Share*. Because* Share *is a randomized algorithm, there are many valid sharings of the same secret (induced by different choices of the highlighted coefficients in* f*).*

Security

To show the security of Shamir secret sharing, we first show a convenient lemma about the distribution of shares in an unauthorized set:

Lemma 3.12 *Let p be a prime and define the following two libraries:*

$\mathcal{L}_{\text{shamir-real}}$

POLY$(m, t, U \subseteq \{1, \ldots, p\})$:
- if $|U| \geqslant t$: return err
- $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
- $f(\boldsymbol{x}) := m + \sum_{j=1}^{t-1} f_j \boldsymbol{x}^j$
- for $i \in U$:
  - $s_i := (i, f(i) \,\%\, p)$
- return $\{s_i \mid i \in U\}$

$\mathcal{L}_{\text{shamir-rand}}$

POLY$(m, t, U \subseteq \{1, \ldots, p\})$:
- if $|U| \geqslant t$: return err
- for $i \in U$:
  - $y_i \leftarrow \mathbb{Z}_p$
  - $s_i := (i, y_i)$
- return $\{s_i \mid i \in U\}$

$\mathcal{L}_{\text{shamir-real}}$ chooses a random degree-$(t-1)$ polynomial that passes through the point $(0, m)$, then evaluates it at the given x-coordinates (specified by U). $\mathcal{L}_{\text{shamir-rand}}$ simply gives uniformly chosen points, unrelated to any polynomial.

The claim is that these libraries are interchangeable: $\mathcal{L}_{\text{shamir-real}} \equiv \mathcal{L}_{\text{shamir-rand}}$.

Proof Fix a message $m \in \mathbb{Z}_p$, fix set U of users with $|U| < t$, 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 POLY(m, t, U) outputs $\{(i, y_i) \mid i \in U\}$, in each of the two libraries.[2]

In library $\mathcal{L}_{\text{shamir-real}}$, the subroutine chooses a random degree-$(t-1)$ polynomial f such that $f(0) \equiv_p m$. From Corollary 3.10, we know there are p^{t-1} such polynomials.

In order for POLY to output points consistent with our chosen y_i's, the library must have chosen one of the polynomials that passes through $(0, m)$ *and* all of the $\{(i, y_i) \mid i \in U\}$ points. The library must have chosen one of the polynomials that passes through a specific choice of $|U| + 1$ points, and Corollary 3.10 tells us that there are $p^{t-(|U|+1)}$ such polynomials.

The only way for POLY to give our desired output is for it to choose one of the $p^{t-(|U|+1)}$ "good" polynomials, out of the p^{t-1} possibilities. This happens with probability exactly

$$\frac{p^{t-|U|-1}}{p^{t-1}} = p^{-|U|}$$

Now, in library $\mathcal{L}_{\text{shamir-rand}}$, POLY chooses its $|U|$ output values uniformly in $\mathbb{Z}_p$. There are $p^{|U|}$ ways to choose them. But only one of those ways causes POLY(m, t, U) to output our specific choice of $\{(i, y_i) \mid i \in U\}$. Hence, the probability of receiving this output is $p^{-|U|}$.

For all possible inputs to POLY, both libraries assign the same probability to every possible output. Hence, the libraries are interchangeable. ∎

Theorem 3.13 *Shamir's secret-sharing scheme (Construction 3.11) is secure according to Definition 3.3.*

[2] This is similar to how, in Claim 2.7, we fixed a particular m and c and computed the probability that EAVESDROP$(m) = c$.

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

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

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

SHARE(m_L, m_R, U):
  if $|U| \geqslant t$: return err
  $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
  $f(\boldsymbol{x}) := m_L + \sum_{j=1}^{t-1} f_j \boldsymbol{x}^j$
  for $i \in U$:
    $s_i := (i, f(i) \,\%\, p)$
  return $\{s_i \mid i \in U\}$

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

SHARE(m_L, m_R, U):
  return POLY(m_L, t, U)

$\diamond$

$\mathcal{L}_{\text{shamir-real}}$

POLY(m, t, U):
  if $|U| \geqslant t$: return err
  $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
  $f(\boldsymbol{x}) := m + \sum_{j=1}^{t-1} f_j \boldsymbol{x}^j$
  for $i \in U$:
    $s_i := (i, f(i) \,\%\, p)$
  return $\{s_i \mid i \in U\}$

Almost the entire body of the SHARE subroutine has been factored out in terms of the $\mathcal{L}_{\text{shamir-real}}$ 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.

SHARE(m_L, m_R, U):
  return POLY(m_L, t, U)

$\diamond$

$\mathcal{L}_{\text{shamir-rand}}$

POLY(m, t, U):
  if $|U| \geqslant t$: return err
  for $i \in U$:
    $y_i \leftarrow \mathbb{Z}_p$
    $s_i := (i, y_i)$
  return $\{s_i \mid i \in U\}$

By Lemma 3.12, we can replace $\mathcal{L}_{\text{shamir-real}}$ with $\mathcal{L}_{\text{shamir-rand}}$, having no effect on the library's behavior.

SHARE(m_L, m_R, U):
  return POLY(m_R, t, U)

$\diamond$

$\mathcal{L}_{\text{shamir-rand}}$

POLY(m, t, U):
  if $|U| \geqslant t$: return err
  for $i \in U$:
    $y_i \leftarrow \mathbb{Z}_p$
    $s_i := (i, y_i)$
  return $\{s_i \mid i \in U\}$

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

$\mathcal{L}_{\text{shamir-real}}$

```
SHARE(m_L, m_R, U):
  return POLY(m_R, t, U)
```

$\diamond$

```
POLY(m, t, U):
  if |U| ⩾ t: return err
  f_1, ..., f_{t-1} ← Z_p
  f(x) := m + Σ_{j=1}^{t-1} f_j x^j
  for i ∈ U:
    s_i := (i, f(i) % p)
  return {s_i | i ∈ U}
```

Applying the same steps in reverse, we can replace $\mathcal{L}_{\text{shamir-rand}}$ with $\mathcal{L}_{\text{shamir-real}}$, having no effect on the library's behavior.

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

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

```
SHARE(m_L, m_R, U):
  if |U| ⩾ t: return err
  f_1, ..., f_{t-1} ← Z_p
  f(x) := m_R + Σ_{j=1}^{t-1} f_j x^j
  for i ∈ U:
    s_i := (i, f(i) % p)
  return {s_i | i ∈ U}
```

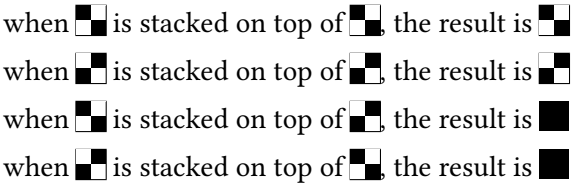
A subroutine has been inlined, which has no effect on the library's behavior. The resulting library is $\mathcal{L}^{\mathcal{S}}_{\text{tsss-R}}$.

We showed that $\mathcal{L}^{\mathcal{S}}_{\text{tsss-L}} \equiv \mathcal{L}_{\text{hyb-1}} \equiv \cdots \equiv \mathcal{L}_{\text{hyb-4}} \equiv \mathcal{L}^{\mathcal{S}}_{\text{tsss-R}}$, so Shamir's secret sharing scheme is secure. ■

⋆ 3.5 Visual Secret Sharing

Here is a fun variant of 2-out-of-2 secret-sharing called **visual secret sharing.** In this variant, both the secret and the shares are black-and-white images. We require the same security property as traditional secret-sharing — that is, a single share (image) by itself reveals no information about the secret (image). What makes visual secret sharing different is that we require the *reconstruction* procedure to be done visually.

More specifically, each share should be printed on transparent sheets. When the two shares are stacked on top of each other, the secret image is revealed visually. We will discuss a simple visual secret sharing scheme that is inspired by the following observations:



Importantly, when stacking shares on top of each other in the first two cases, the result is a 2×2 block that is half-black, half-white (let's call it "gray"); while in the other cases the result is completely black.

The idea is to process each pixel of the source image independently, and to encode each pixel as a 2×2 block of pixels in each of the shares. A white pixel should be shared in

a way that the two shares stack to form a "gray" 2×2 block, while a black pixel is shared in a way that results in a black 2×2 block.

More formally:

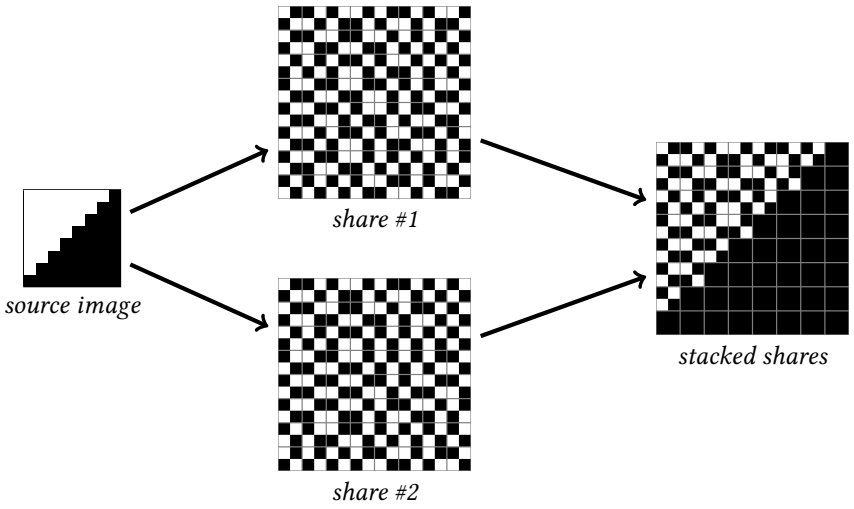
Construction 3.14

Share(m):
  initialize empty images s_1, s_2, with dimensions twice that of m
  for each position (i, j) in m:
    randomly choose $b_1 \leftarrow$ {  }
    if $m[i, j]$ is a white pixel: set $b_2 := b_1$
    if $m[i, j]$ is a black pixel: set b_2 to the "opposite" of b_1 (*i.e.*, {} $\setminus \{b_1\}$)
    add 2×2 block b_1 to image s_1 at position $(2i, 2j)$
    add 2×2 block b_2 to image s_2 at position $(2i, 2j)$
  return (s_1, s_2)

It is not hard to see that share s_1 leaks no information about the secret image m, because it consists of uniformly chosen 2×2 blocks. In the exercises you are asked to prove that s_2 also individually leaks nothing about the secret image.

Note that whenever the source pixel is white, the two shares have identical 2×2 blocks (so that when stacked, they make a "gray" block). Whenever a source pixel is black, the two shares have opposite blocks, so stack to make a black block.

Example



Exercises

3.1. Generalize Construction 3.5 to be an n-out-of-n secret-sharing scheme, and prove that your scheme is correct and secure.

3.2. Prove Theorem 3.7.

3.3. Fill in the details of the following alternative proof of Theorem 3.6: Starting with $\mathcal{L}_{\text{tsss-L}}$, apply the first step of the proof as before, to duplicate the main body into 3 branches of a new if-statement. Then apply Exercise 2.3 to the second branch of the if-statement. Argue that m_L can be replaced with m_R and complete the proof.

3.4. Suppose T is a fixed (publicly known) invertible $n \times n$ matrix over $\mathbb{Z}_p$, where p is a prime.

(a) Show that the following two libraries are interchangeable:

$\mathcal{L}_{\text{left}}$
QUERY():
$\boldsymbol{r} \leftarrow (\mathbb{Z}_p)^n$
return $\boldsymbol{r}$

;

$\mathcal{L}_{\text{right}}$
QUERY():
$\boldsymbol{r} \leftarrow (\mathbb{Z}_p)^n$
return $T \times \boldsymbol{r}$

.

(b) Show that the following two libraries are interchangeable:

$\mathcal{L}_{\text{left}}$
QUERY($\boldsymbol{v} \in (\mathbb{Z}_p)^n$):
$\boldsymbol{r} \leftarrow (\mathbb{Z}_p)^n$
$\boldsymbol{z} := \boldsymbol{v} + T\boldsymbol{r}$
return $\boldsymbol{z}$

;

$\mathcal{L}_{\text{right}}$
QUERY($\boldsymbol{v} \in (\mathbb{Z}_p)^n$):
$\boldsymbol{z} \leftarrow (\mathbb{Z}_p)^n$
return $\boldsymbol{z}$

.

3.5. Consider a t-out-of-n threshold secret sharing scheme with $\mathcal{M} = \{0,1\}^\ell$, and where each user's share is also a string of bits. Prove that if the scheme is secure, 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. Give your answer in the form of an distinguisher on the relevant libraries.

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 $\{(i, z_i) \mid i \leqslant n\}$ 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

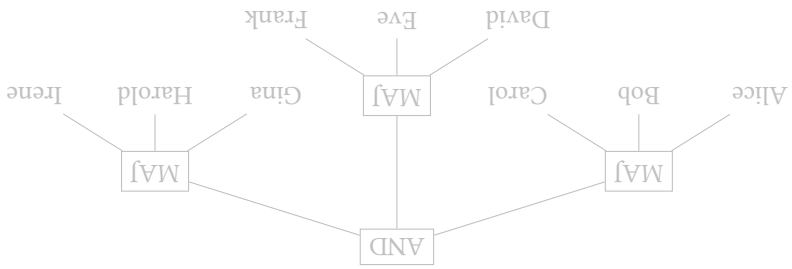
Describe in detail how the sharing algorithm works and how the reconstruction works (for all authorized sets of users).

Note: It is fine if different users have shares which are of different sizes (*e.g.*, different number of bits to represent), and it is also fine if the Reconstruct algorithm depends on the identities of the users who are contributing their shares.

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

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

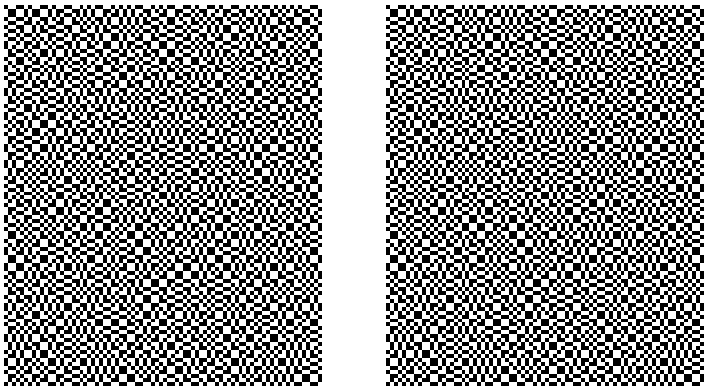
Hint: express the formula as a tree of AND and OR gates.

(b) Give a construction of a t-out-of-n secret-sharing scheme in which all shares are binary strings, and the only operation required of Share and Reconstruct is XOR (so no mod-p operations).

How big are the shares, compared to the Shamir scheme?

3.10. Prove that share s_2 in Construction 3.14 is distributed independently of the secret m.

3.11. Using actual transparencies or with an image editing program, reconstruct the secret shared in these two images:



★ 3.12. Construct a 3-out-of-3 visual secret sharing scheme. Any two shares should together reveal nothing about the source image, but any three reveal the source image when stacked together.