Advanced Cryptography

（Provable Security）

Yi LIU

An Example of Hybrid Proof

- Double OTP
  - KeyGen: $k_1 \leftarrow \{0,1\}^\lambda$, $k_2 \leftarrow \{0,1\}^\lambda$, return (k_1, k_2).
  - Enc($(k_1, k_2), m$): $c_1 \coloneqq k_1 \oplus m$, $c_2 \coloneqq k_2 \oplus c_1$, return c_2.
  - Dec($(k_1, k_2), c$): $c_1 \coloneqq k_2 \oplus c_2$, $m \coloneqq k_1 \oplus c_1$, return m.

Claim The construction of Double OTP has one-time uniform ciphertexts.

Proof we must show that:

$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$
ctxt$(m \in \Sigma.\mathcal{M})$:
$\quad (k_1, k_2) \leftarrow \{0,1\}^{2\lambda}$
$\quad c_1 \coloneqq k_1 \oplus m$
$\quad c_2 \coloneqq k_2 \oplus c_1$
$\quad$ return c_2

$\equiv$

$\mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$
ctxt$(m \in \Sigma.\mathcal{M})$:
$\quad c \leftarrow \{0,1\}^\lambda$
$\quad$ return c

An Example of Hybrid Proof

Claim The construction of Double OTP has one-time uniform ciphertexts.

Proof we must show that:

$$\boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots\$-real}} \\ \hline \textbf{ctxt}(m \in \Sigma.\mathcal{M}): \\ \quad (k_1, k_2) \leftarrow \{0,1\}^{2\lambda} \\ \quad c_1 := k_1 \oplus m \\ \quad c_2 := k_2 \oplus c_1 \\ \quad \text{return } c_2 \end{array}} \equiv \boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \\ \hline \textbf{ctxt}(m \in \Sigma.\mathcal{M}): \\ \quad c \leftarrow \{0,1\}^{\lambda} \\ \quad \text{return } c \end{array}}$$

$$\boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots\$-real}} \\ \hline \underline{\text{CTXT}(m):} \\ \quad k_1 \leftarrow \{0,1\}^{\lambda} \\ \quad k_2 \leftarrow \{0,1\}^{\lambda} \\ \quad c_1 := k_1 \oplus m \\ \quad c_2 := k_2 \oplus c_1 \\ \quad \text{return } c_2 \end{array}} \equiv \underbrace{\boxed{\begin{array}{l} \underline{\text{CTXT}(m):} \\ \quad k_1 \leftarrow \{0,1\}^{\lambda} \\ \quad c_1 := k_1 \oplus m \\ \quad c_2 := \text{CTXT}'(c_1) \\ \quad \text{return } c_2 \end{array}} \diamond \boxed{\begin{array}{l} \mathcal{L}_{\text{otp-real}} \\ \hline \underline{\text{CTXT}'(m'):} \\ \quad k \leftarrow \{0,1\}^{\lambda} \\ \quad \text{return } k \oplus m' \end{array}}}_{\mathcal{L}_{\text{hyb-1}}}$$

An Example of Hybrid Proof

Claim The construction of Double OTP has one-time uniform ciphertexts.

Proof we must show that:

$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$
ctxt$(m \in \Sigma.\mathcal{M})$: $(k_1, k_2) \leftarrow \{0,1\}^{2\lambda}$ $c_1 := k_1 \oplus m$ $c_2 := k_2 \oplus c_1$ return c_2

$\equiv$

$\mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$
ctxt$(m \in \Sigma.\mathcal{M})$: $c \leftarrow \{0,1\}^{\lambda}$ return c

$$\underbrace{\boxed{\begin{array}{l}\textsc{ctxt}(m):\\ \quad k_1 \leftarrow \{0,1\}^{\lambda}\\ \quad c_1 := k_1 \oplus m\\ \quad c_2 := \textsc{ctxt}'(c_1)\\ \quad \text{return } c_2\end{array}} \diamond \boxed{\begin{array}{l}\mathcal{L}_{\text{otp-real}}\\ \hline \textsc{ctxt}'(m'):\\ \quad k \leftarrow \{0,1\}^{\lambda}\\ \quad \text{return } k \oplus m'\end{array}}}_{\mathcal{L}_{\text{hyb-1}}} \equiv \underbrace{\boxed{\begin{array}{l}\textsc{ctxt}(m):\\ \quad k_1 \leftarrow \{0,1\}^{\lambda}\\ \quad c_1 := k_1 \oplus m\\ \quad c_2 := \textsc{ctxt}'(c_1)\\ \quad \text{return } c_2\end{array}} \diamond \boxed{\begin{array}{l}\mathcal{L}_{\text{otp-rand}}\\ \hline \textsc{ctxt}'(m'):\\ \quad c \leftarrow \{0,1\}^{\lambda}\\ \quad \text{return } c\end{array}}}_{\mathcal{L}_{\text{hyb-2}}}$$

An Example of Hybrid Proof

Claim The construction of Double OTP has one-time uniform ciphertexts.

Proof we must show that:

$$\boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots\$-real}} \\ \hline \textbf{ctxt}(m \in \Sigma.\mathcal{M}): \\ \quad (k_1, k_2) \leftarrow \{0,1\}^{2\lambda} \\ \quad c_1 := k_1 \oplus m \\ \quad c_2 := k_2 \oplus c_1 \\ \quad \text{return } c_2 \end{array}} \equiv \boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \\ \hline \textbf{ctxt}(m \in \Sigma.\mathcal{M}): \\ \quad c \leftarrow \{0,1\}^{\lambda} \\ \quad \text{return } c \end{array}}$$

$$\underbrace{\boxed{\begin{array}{l} \text{CTXT}(m): \\ \hline \quad k_1 \leftarrow \{0,1\}^{\lambda} \\ \quad c_1 := k_1 \oplus m \\ \quad c_2 := \text{CTXT}'(c_1) \\ \quad \text{return } c_2 \end{array}} \diamond \boxed{\begin{array}{l} \mathcal{L}_{\text{otp-rand}} \\ \hline \text{CTXT}'(m'): \\ \quad c \leftarrow \{0,1\}^{\lambda} \\ \quad \text{return } c \end{array}}}_{\mathcal{L}_{\text{hyb-2}}} \equiv \underbrace{\boxed{\begin{array}{l} \text{CTXT}(m): \\ \hline \quad k_1 \leftarrow \{0,1\}^{\lambda} \\ \quad c_1 := k_1 \oplus m \\ \quad c_2 \leftarrow \{0,1\}^{\lambda} \\ \quad \text{return } c_2 \end{array}}}_{\mathcal{L}_{\text{hyb-3}}}$$

An Example of Hybrid Proof

Claim The construction of Double OTP has one-time uniform ciphertexts.

Proof we must show that:

$$\boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots\$-real}} \\ \hline \textbf{ctxt}(m \in \Sigma.\mathcal{M}): \\ \quad (k_1, k_2) \leftarrow \{0,1\}^{2\lambda} \\ \quad c_1 := k_1 \oplus m \\ \quad c_2 := k_2 \oplus c_1 \\ \quad \text{return } c_2 \end{array}} \equiv \boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \\ \hline \textbf{ctxt}(m \in \Sigma.\mathcal{M}): \\ \quad c \leftarrow \{0,1\}^{\lambda} \\ \quad \text{return } c \end{array}}$$

$$\underbrace{\boxed{\begin{array}{l} \underline{\textsc{ctxt}(m):} \\ \quad k_1 \leftarrow \{0,1\}^{\lambda} \\ \quad c_1 := k_1 \oplus m \\ \quad c_2 \leftarrow \{0,1\}^{\lambda} \\ \quad \text{return } c_2 \end{array}}}_{\mathcal{L}_{\text{hyb-3}}} \equiv \boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \\ \hline \underline{\textsc{ctxt}(m):} \\ \quad c_2 \leftarrow \{0,1\}^{\lambda} \\ \quad \text{return } c_2 \end{array}}$$

Summary of the Hybrid Technique

- Proving security means showing that two particular libraries, say $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$, are interchangeable.
- Often $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ are significantly different, The idea is to break up the large "gap" between $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ into smaller ones that are easier to justify.
- We must justify why each modification doesn't affect the calling program (i.e., why the two libraries before/after the modification are interchangeable).

Another Construction

- KeyGen: $k \leftarrow \{0,1\}^{\lambda}$, return k.
- Enc$((k), m)$: $c_1 \coloneqq k \oplus m$, $c_2 \coloneqq k \oplus c_1$, return c_2.
- Dec$((k), c)$: $c_1 \coloneqq k \oplus c_2$, $m \coloneqq k \oplus c_1$, return m.

Let's try to repeat the steps of our previous security proof on this (insecure) scheme and see where things break down.

$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$

$\underline{\textsc{ctxt}(m):}$

$\quad k \leftarrow \{0,1\}^{\lambda}$ // KeyGen

$\left.\begin{array}{l} c_1 \coloneqq k \oplus m \\ c_2 \coloneqq k \oplus c_1 \\ \text{return } c_2 \end{array}\right\}$ Enc

$\overset{?}{\equiv}$

$\mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$

$\underline{\textsc{ctxt}(m):}$

$\quad c \leftarrow \{0,1\}^{\lambda}$

$\quad \text{return } c$

Another Construction

- KeyGen: $k \leftarrow \{0,1\}^\lambda$, return k.
- Enc$((k), m)$: $c_1 \coloneqq k \oplus m$, $c_2 \coloneqq k \oplus c_1$, return c_2.
- Dec$((k), c)$: $c_1 \coloneqq k \oplus c_2$, $m \coloneqq k \oplus c_1$, return m.

Let's try to repeat the steps of our previous security proof on this (insecure) scheme and see where things break down

$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$

```
ctxt(m):
  k ← {0,1}^λ
  c1 := k ⊕ m
  c2 := k ⊕ c1
  return c2
```

$\overset{?}{\equiv}$

```
ctxt(m):
  c1 := k ⊕ m   // ??
  c2 := ctxt'(c1)
  return c2
```

$\diamond$

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

```
ctxt'(m'):
  k ← {0,1}^λ
  return k ⊕ m'
```

$\mathcal{L}_{\text{hyb}}$

$\mathcal{L}_{\text{otp-real}}$ only gives us a way to use k in one xor expression, whereas we need to use the same k in two xor expressions to match the behavior of $\mathcal{L}_{\text{ots\$-real}}$. Failed!

How to Compare/Contrast Security Definitions

- A definition can’t really be “wrong,” but it can be “not as useful as you hoped” or it can “fail to adequately capture your intuition”.
- One way to compare/contrast two security definitions is to prove that one implies the other.
  - if an encryption scheme satisfies definition #1, then it also satisfies definition #2.

One Security Definition Implies Another

Theorem If an encryption scheme Σ has one-time uniform ciphertexts, then Σ also has one-time secrecy. In other words:

$$\mathcal{L}^{\Sigma}_{\text{ots\$-real}} \equiv \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \Longrightarrow \mathcal{L}^{\Sigma}_{\text{ots-L}} \equiv \mathcal{L}^{\Sigma}_{\text{ots-R}}$$

Proof

- We will start with the library $\mathcal{L}^{\Sigma}_{\text{ots-L}}$, and make a small sequence of justifiable changes to it, until finally reaching $\mathcal{L}^{\Sigma}_{\text{ots-R}}$.
- Along the way, we can use the fact that $\mathcal{L}^{\Sigma}_{\text{ots\$-real}} \equiv \mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$.

One Security Definition Implies Another

Theorem If an encryption scheme Σ has one-time uniform ciphertexts, then Σ also has one-time secrecy. In other words:

$$\mathcal{L}^{\Sigma}_{\text{ots\$-real}} \equiv \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \Longrightarrow \mathcal{L}^{\Sigma}_{\text{ots-L}} \equiv \mathcal{L}^{\Sigma}_{\text{ots-R}}$$

Proof

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

EAVESDROP(m_L, m_R):
  $k \leftarrow \Sigma.\text{KeyGen}$
  $c \leftarrow \Sigma.\text{Enc}(k, m_L)$
  return c

EAVESDROP(m_L, m_R):
  $c :=$ CTXT(m_L)
  return c

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$

CTXT(m):
  $k \leftarrow \Sigma.\text{KeyGen}$
  $c \leftarrow \Sigma.\text{Enc}(k, m)$
  return c

One Security Definition Implies Another

Theorem If an encryption scheme Σ has one-time uniform ciphertexts, then Σ also has one-time secrecy. In other words:

$$\mathcal{L}^{\Sigma}_{\text{ots\$-real}} \equiv \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \Longrightarrow \mathcal{L}^{\Sigma}_{\text{ots-L}} \equiv \mathcal{L}^{\Sigma}_{\text{ots-R}}$$

Proof

$\textsc{eavesdrop}(m_L, m_R)$:
$c := \textsc{ctxt}(m_L)$
return c

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$
$\textsc{ctxt}(m)$:
$k \leftarrow \Sigma.\text{KeyGen}$
$c \leftarrow \Sigma.\text{Enc}(k, m)$
return c

$\Longrightarrow$

$\textsc{eavesdrop}(m_L, m_R)$:
$c := \textsc{ctxt}(m_L)$
return c

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$
$\textsc{ctxt}(m)$:
$c \leftarrow \Sigma.\mathcal{C}$
return c

One Security Definition Implies Another

Theorem If an encryption scheme Σ has one-time uniform ciphertexts, then Σ also has one-time secrecy. In other words:

$$\mathcal{L}^{\Sigma}_{\text{ots\$-real}} \equiv \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \Longrightarrow \mathcal{L}^{\Sigma}_{\text{ots-L}} \equiv \mathcal{L}^{\Sigma}_{\text{ots-R}}$$

Proof

$\textsc{eavesdrop}(m_L, m_R)$:
  $c := \textsc{ctxt}(m_L)$
  return c

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$
$\textsc{ctxt}(m)$:
  $c \leftarrow \Sigma.\mathcal{C}$
  return c

$\Longrightarrow$

$\textsc{eavesdrop}(m_L, m_R)$:
  $c := \textsc{ctxt}(m_R)$
  return c

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$
$\textsc{ctxt}(m)$:
  $c \leftarrow \Sigma.\mathcal{C}$
  return c

One Security Definition Implies Another

Theorem If an encryption scheme Σ has one-time uniform ciphertexts, then Σ also has one-time secrecy. In other words:

$$\mathcal{L}^{\Sigma}_{\text{ots\$-real}} \equiv \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \Longrightarrow \mathcal{L}^{\Sigma}_{\text{ots-L}} \equiv \mathcal{L}^{\Sigma}_{\text{ots-R}}$$

Proof Perform the same sequence of steps, but in reverse.

$\underline{\text{EAVESDROP}(m_L, m_R):}$ $c := \text{CTXT}(\boxed{m_R})$ return c	$\diamond$	$\mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$ $\underline{\text{CTXT}(m):}$ $c \leftarrow \Sigma.\mathcal{C}$ return c

$\Longrightarrow$

$\underline{\text{EAVESDROP}(m_L, m_R):}$ $c := \text{CTXT}(m_R)$ return c	$\diamond$	$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$ $\underline{\text{CTXT}(m):}$ $k \leftarrow \Sigma.\text{KeyGen}$ $c \leftarrow \Sigma.\text{Enc}(k, m)$ return c

One Security Definition Implies Another

Theorem If an encryption scheme Σ has one-time uniform ciphertexts, then Σ also has one-time secrecy. In other words:

$$\mathcal{L}^{\Sigma}_{\text{ots\$-real}} \equiv \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \Longrightarrow \mathcal{L}^{\Sigma}_{\text{ots-L}} \equiv \mathcal{L}^{\Sigma}_{\text{ots-R}}$$

Proof Perform the same sequence of steps, but in reverse.

EAVESDROP(m_L, m_R):
$c := \text{CTXT}(m_R)$
return c

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$
CTXT(m):
$k \leftarrow \Sigma.\text{KeyGen}$
$c \leftarrow \Sigma.\text{Enc}(k, m)$
return c

$\Rightarrow$

$\mathcal{L}^{\Sigma}_{\text{ots-R}}$
EAVESDROP(m_L, m_R):
$k \leftarrow \Sigma.\text{KeyGen}$
$c \leftarrow \Sigma.\text{Enc}(k, m_R)$
return c

One Security Definition Doesn't Imply Another

- If we have two security definitions that both capture our intuitions rather well, then any scheme which satisfies one definition and not the other is bound to appear unnatural and contrived.
- The point is to gain more understanding of the security definitions themselves, and unnatural/contrived schemes are just a means to do that.

One Security Definition Doesn't Imply Another

Theorem There is an encryption scheme that satisfies one-time secrecy but not one-time uniform ciphertexts. In other words, one-time secrecy does not necessarily imply one-time uniform ciphertexts.

Proof

KeyGen: return $k \leftarrow \{0,1\}^{\lambda}$.

Enc$(k, m \in \{0,1\}^{\lambda})$: $c' \coloneqq k \oplus m$, $c \coloneqq c' || 00$, return c.

Dec$(k, c \in \{0,1\}^{\lambda+2})$: $c' \coloneqq$first λ bits of c, return $k \oplus c'$.

One Security Definition Doesn't Imply Another

Theorem There is an encryption scheme that satisfies one-time secrecy but not one-time uniform ciphertexts.

Proof

KeyGen: return $k \leftarrow \{0,1\}^\lambda$.

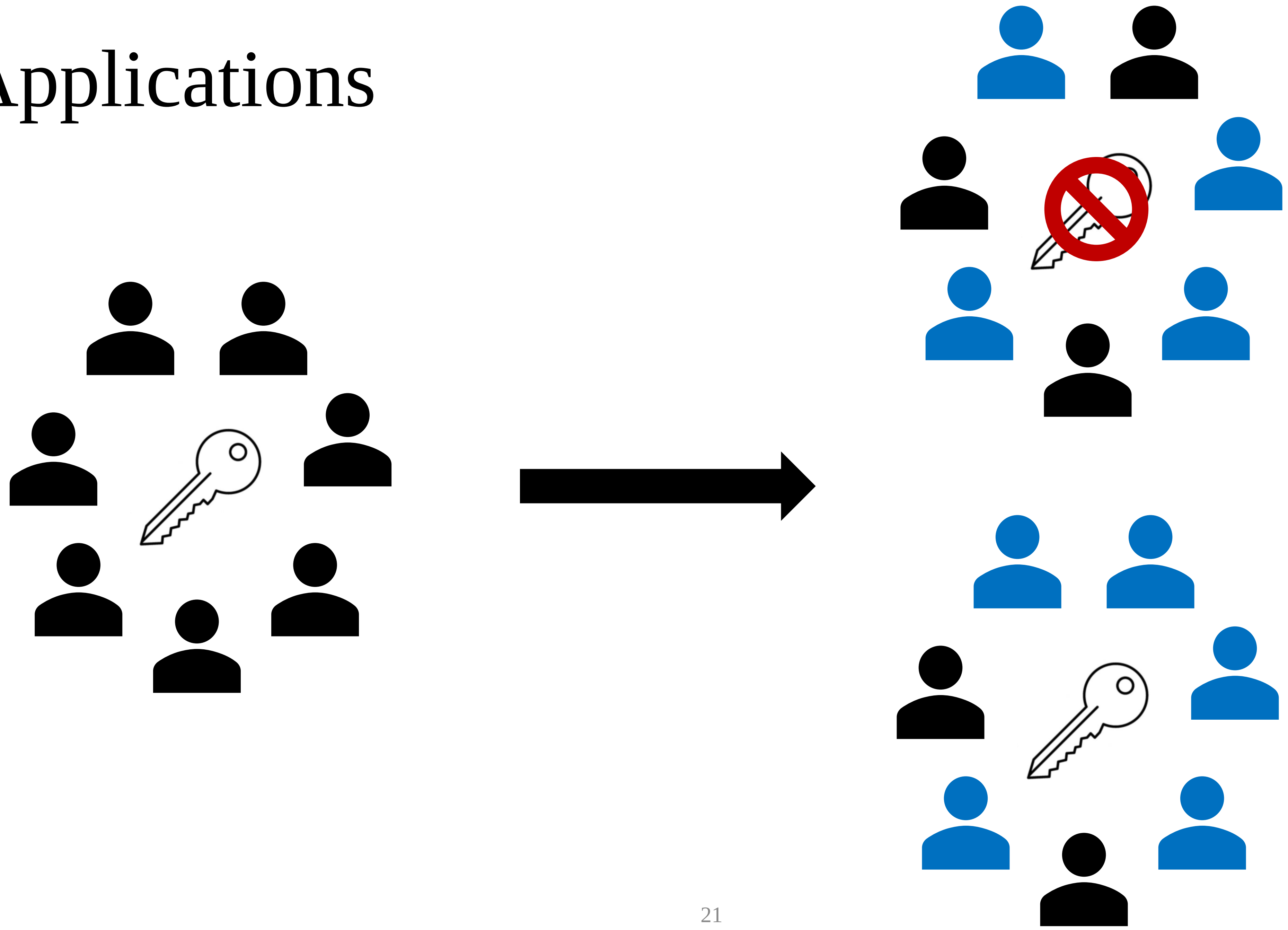
Enc$(k, m \in \{0,1\}^\lambda)$: $c' := k \oplus m$, $c := c' || 00$, return c.

Dec$(k, c \in \{0,1\}^{\lambda+2})$: $c' :=$first λ bits of c, return $k \oplus c'$.

- This scheme satisfies one-time secrecy. Encryptions of m_L are distributed identically to encryptions of m_R for all (m_L, m_R).
- This scheme does not satisfy the one-time uniform ciphertexts property.
  - Its ciphertexts always end with 00, whereas uniform strings end with 00 with probability $1/4$.
- This can be fixed by redefining the ciphertext space as $\mathcal{C}$ as the set of $\lambda + 2$-bit strings whose last two bits are 00.

Secret Sharing

Applications



Secret-Sharing Scheme

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

- $\mathcal{M}$ is the message space, t is the threshold.

Secret-Sharing Scheme

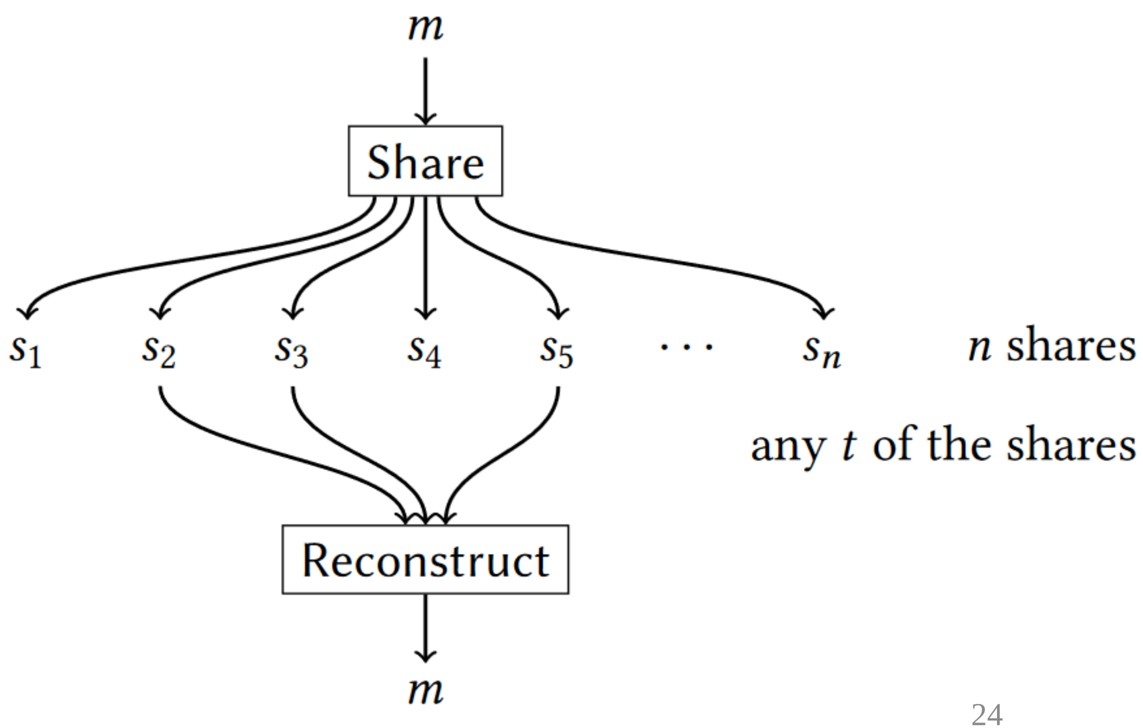
Definition A t-out-of-n threshold secret-sharing scheme (TSSS) consists of the following algorithms:

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

- Let $U \subseteq \{1, \ldots, n\}$ be a subset of users. If $|U| \geq t$, 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.

Secret-Sharing Scheme - Correctness

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



Security Definition

- Intuition

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

- 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.

Security Definition

- 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 $U \in \{1, \ldots, \Sigma.n\}$ and:

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

- Return err means we want security that the attackers see only unauthorized set of shares.

More About Secret Sharing

- Two independent executions of the Share algorithm
  - Share algorithm 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.

An Construction

- $\mathcal{M} = \{0,1\}^{500}, t = 5, n = 5$
  - i.e., we want to split a secret into 5 pieces so that any 4 of the pieces leak nothing
- Share(m): split m into $m = s_1 || \cdots || s_5$, where $|s_i| = 100$, return $(s_1, \ldots, s_5)$.
- Reconstruct($s_1, \ldots, s_5$): return $s_1 || \cdots || s_5$.

- This construction satisfies the correctness property.
- But this construction is insecure.

An Construction

- $\mathcal{M} = \{0,1\}^{500}$, $t = 5$, $n = 5$
- $\text{Share}(m)$: split m into $m = s_1 || \cdots || s_5$, where $|s_i| = 100$, return $(s_1, \dots, s_5)$.
- $\text{Reconstruct}(s_1, \dots, s_5)$: return $s_1 || \cdots || s_5$.
- But this construction is insecure.

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

An Construction

- $\mathcal{M} = \{0,1\}^{500}$, $t = 5$, $n = 5$
- $\text{Share}(m)$: split m into $m = s_1 || \cdots || s_5$, where $|s_i| = 100$, return $(s_1, \dots, s_5)$.
- $\text{Reconstruct}(s_1, \dots, s_5)$: return $s_1 || \cdots || s_5$.
- But this construction is insecure.

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

$\diamond$

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

$\mathcal{A}$ outputs 1 with probability 1 for $\mathcal{L}_{\text{tsss-L}}$.

An Construction

- $\mathcal{M} = \{0,1\}^{500}$, $t = 5$, $n = 5$
- Share(m): split m into $m = s_1 || \cdots || s_5$, where $|s_i| = 100$, return $(s_1, \ldots, s_5)$.
- Reconstruct$(s_1, \ldots, s_5)$: return $s_1 || \cdots || s_5$.

- But this construction is insecure.

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

$\diamond$

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

$\mathcal{A}$ outputs 1 with probability 1 for $\mathcal{L}_{\text{tsss}-\text{L}}$.

$\mathcal{A}$ outputs 1 with probability 0 for $\mathcal{L}_{\text{tsss}-\text{R}}$.

A Simple 2-out-of-2 Scheme

- $\mathcal{M} = \{0,1\}^\ell, t = 2, n = 2$
- Share(m): $s_1 \leftarrow \{0,1\}^\ell$, $s_2 \coloneqq s_1 \oplus m$, return (s_1, s_2).
- Reconstruct(s_1, s_2): return $s_1 \oplus s_2$.

Theorem This construction is a secure 2-out-of-2 threshold secret-sharing scheme.

A Simple 2-out-of-2 Scheme

- $\mathcal{M} = \{0,1\}^\ell$, $t = 2$, $n = 2$
- Share(m): $s_1 \leftarrow \{0,1\}^\ell$, $s_2 \coloneqq s_1 \oplus m$, return (s_1, s_2).
- Reconstruct(s_1, s_2): return $s_1 \oplus s_2$.

Theorem This construction is a secure 2-out-of-2 threshold secret-sharing scheme.

Proof

A Simple 2-out-of-2 Scheme

- $\mathcal{M} = \{0,1\}^\ell$, $t = 2$, $n = 2$
- Share(m): $s_1 \leftarrow \{0,1\}^\ell$, $s_2 \coloneqq s_1 \oplus m$, return (s_1, s_2).
- Reconstruct(s_1, s_2): return $s_1 \oplus s_2$.

Theorem This construction is a secure 2-out-of-2 threshold secret-sharing scheme.

Proof

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

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 $\{s_i \mid i \in U\}$

SHARE(m_L, m_R, U):
  if $|U| \geqslant 2$: return err
  if $U = \{1\}$:
    $s_1 \leftarrow \{0,1\}^\ell$
    $s_2 := s_1 \oplus m_L$
    return $\{s_1\}$
  elsif $U = \{2\}$:
    $s_1 \leftarrow \{0,1\}^\ell$
    $s_2 := s_1 \oplus m_L$
    return $\{s_2\}$
  else return $\emptyset$

A Simple 2-out-of-2 Scheme

- $\mathcal{M} = \{0,1\}^\ell$, $t = 2$, $n = 2$
- $\text{Share}(m)$: $s_1 \leftarrow \{0,1\}^\ell$, $s_2 \coloneqq s_1 \oplus m$, return (s_1, s_2).
- $\text{Reconstruct}(s_1, s_2)$: return $s_1 \oplus s_2$.

Theorem This construction is a secure 2-out-of-2 threshold secret-sharing scheme.

Proof

```
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 ∅
```

$\Rightarrow$

```
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 ∅
```

Because s_2 is never used in this branch.

A Simple 2-out-of-2 Scheme

- $\mathcal{M} = \{0,1\}^\ell$, $t = 2$, $n = 2$
- Share(m): $s_1 \leftarrow \{0,1\}^\ell$, $s_2 \coloneqq s_1 \oplus m$, return (s_1, s_2).
- Reconstruct(s_1, s_2): return $s_1 \oplus s_2$.

Theorem This construction is a secure 2-out-of-2 threshold secret-sharing scheme.

Proof

$\underline{\textsc{share}(m_L, m_R, U):}$
- if $|U| \geqslant 2$: return $\texttt{err}$
- if $U = \{1\}$:
  - $s_1 \leftarrow \{\texttt{0},\texttt{1}\}^\ell$
  - $s_2 := s_1 \oplus \boxed{m_R}$
  - return $\{s_1\}$
- elsif $U = \{2\}$:
  - $s_1 \leftarrow \{\texttt{0},\texttt{1}\}^\ell$
  - $s_2 := s_1 \oplus m_L$
  - return $\{s_2\}$
- else return $\emptyset$

$\Rightarrow$

$\underline{\textsc{share}(m_L, m_R, U):}$
- if $|U| \geqslant 2$: return $\texttt{err}$
- if $U = \{1\}$:
  - $s_1 \leftarrow \{\texttt{0},\texttt{1}\}^\ell$
  - $s_2 := s_1 \oplus m_R$
  - return $\{s_1\}$
- elsif $U = \{2\}$:
  - $\boxed{s_2 \leftarrow \textsc{eavesdrop}(m_L, m_R)}$
  - return $\{s_2\}$
- else return $\emptyset$

$\diamond$

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

$\underline{\textsc{eavesdrop}(m_L, m_R):}$
- $k \leftarrow \{\texttt{0},\texttt{1}\}^\ell$
- $c := k \oplus m_L$
- return c

A Simple 2-out-of-2 Scheme

- $\mathcal{M} = \{0,1\}^\ell$, $t = 2$, $n = 2$
- $\text{Share}(m)$: $s_1 \leftarrow \{0,1\}^\ell$, $s_2 \coloneqq s_1 \oplus m$, return (s_1, s_2).
- $\text{Reconstruct}(s_1, s_2)$: return $s_1 \oplus s_2$.

Theorem This construction is a secure 2-out-of-2 threshold secret-sharing scheme.

Proof

```
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 ∅
```

$\diamond$

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

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

$\Rightarrow$

```
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 ∅
```

$\diamond$

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

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

A Simple 2-out-of-2 Scheme

- $\mathcal{M} = \{0,1\}^\ell$, $t = 2$, $n = 2$
- $\text{Share}(m)$: $s_1 \leftarrow \{0,1\}^\ell$, $s_2 \coloneqq s_1 \oplus m$, return (s_1, s_2).
- $\text{Reconstruct}(s_1, s_2)$: return $s_1 \oplus s_2$.

Theorem This construction is a secure 2-out-of-2 threshold secret-sharing scheme.

Proof

$\textsc{share}(m_L, m_R, U)$:
  if $|U| \geqslant 2$: return err
  if $U = \{1\}$:
    $s_1 \leftarrow \{0,1\}^\ell$
    $s_2 \coloneqq s_1 \oplus m_R$
    return $\{s_1\}$
  elsif $U = \{2\}$:
    $s_2 \leftarrow \textsc{eavesdrop}(m_L, m_R)$
    return $\{s_2\}$
  else return $\emptyset$

$\diamond$

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

$\textsc{eavesdrop}(m_L, m_R)$:
  $k \leftarrow \{0,1\}^\ell$
  $c \coloneqq k \oplus m_R$
  return c

$\textsc{share}(m_L, m_R, U)$:
  if $|U| \geqslant 2$: return err
  if $U = \{1\}$:
    $s_1 \leftarrow \{0,1\}^\ell$
    $s_2 \coloneqq s_1 \oplus m_R$
    return $\{s_1\}$
  elsif $U = \{2\}$:
    $s_1 \leftarrow \{0,1\}^\ell$
    $s_2 \coloneqq s_1 \oplus m_R$
    return $\{s_2\}$
  else return $\emptyset$

A Simple 2-out-of-2 Scheme

- $\mathcal{M} = \{0,1\}^\ell$, $t = 2$, $n = 2$
- Share(m): $s_1 \leftarrow \{0,1\}^\ell$, $s_2 \coloneqq s_1 \oplus m$, return (s_1, s_2).
- Reconstruct(s_1, s_2): return $s_1 \oplus s_2$.

Theorem This construction is a secure 2-out-of-2 threshold secret-sharing scheme.

Proof

$\textsc{share}(m_L, m_R, U)$:
  if $|U| \geqslant 2$: return err
  if $U = \{1\}$:
    $s_1 \leftarrow \{0,1\}^\ell$
    $s_2 \coloneqq s_1 \oplus m_R$
    return $\{s_1\}$
  elsif $U = \{2\}$:
    $s_1 \leftarrow \{0,1\}^\ell$
    $s_2 \coloneqq s_1 \oplus m_R$
    return $\{s_2\}$
  else return $\emptyset$

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

$\textsc{share}(m_L, m_R, U)$:
  if $|U| \geqslant 2$: return err
  $s_1 \leftarrow \{0,1\}^\ell$
  $s_2 \coloneqq s_1 \oplus m_R$
  return $\{s_i \mid i \in U\}$

Rewrite the Construction

- $\mathcal{M} = \{0,1\}^\ell$, $t = 2$, $n = 2$
- Share(m): $s_1 \leftarrow \Sigma.\text{KeyGen}$, $s_2 \coloneqq \Sigma.\text{Enc}(s_1, m)$, return (s_1, s_2).
- Reconstruct(s_1, s_2): return $\Sigma.\text{Dec}(s_1, s_2)$.

Theorem If Σ is an encryption scheme with one-time secrecy, then this 2-out-of-2 threshold secret-sharing scheme is secure.

Polynomial Interpolation

- Two points determine a line.
- Three points determine a parabola.
- $d + 1$ points determine a unique degree-d polynomial.

- 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.

Polynomial Interpolation

Theorem 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

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

It is clear that ℓ_1 is a degree-d polynomial.

- $\ell_1(x_1) = 1$
- $\ell_1(x_i) = 0$ for $i \neq 1$

Polynomial Interpolation

Theorem 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

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

It is clear that ℓ_j is a degree-d polynomial.

- $\ell_j(x_j) = 1$
- $\ell_j(x_i) = 0$ for $i \neq j$

Polynomial Interpolation

Theorem 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

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

ℓ_j is called LaGrange polynomials. It is a degree-d polynomials and

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

Polynomial Interpolation

Theorem 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

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

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

$$f(x_i) = y_1 \cdot 0 + \cdots + y_i \cdot 1 + y_{d+1} \cdot 0 = y_i$$

This shows that there is some degree-d polynomial satisfying $y_i = f(x_i)$ for all i.

Polynomial Interpolation

Theorem 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

Suppose there are two degree-d polynomials f and f', such that $f(x_i) = f'(x_i) = y_i$. Then $g(x) = f(x) - f'(x)$ is also degree-d, and it satisfies $g(x_i) = 0$ for all i.

But the only degree-d polynomial with $d + 1$ roots is the identically-zero polynomial $g(x) = 0$. Hence, $f = f'$. So f is the unique polynomial.

Polynomials mod p

- 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!

Polynomial Interpolation mod p

Theorem 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.

- $d + 1$ points uniquely determine a degree-d polynomial
- **Generalization**: For any k points, there are exactly p^{d+1-k} polynomials of degree-d that hit those points, mod p.

Polynomial Interpolation mod p

Corollary 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 $k \leq d + 1$ 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

Prove by induction on the value $d + 1 - k$.

Base case: $d + 1 - k = 0$, then we have $k = d + 1$ distinct points. Has been proved.

Inductive case ($k + 1$ case holds): $k \leq d$ points in $\mathcal{P}$. Let $x^* \in \mathbb{Z}_p \neq x_i$ for all i. Every polynomial must give some value when evaluated at x^*.

Compute [# of degree-d polynomials pass through points in $\mathcal{P}$]

Polynomial Interpolation mod p

Corollary 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 $k \leq d + 1$ 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

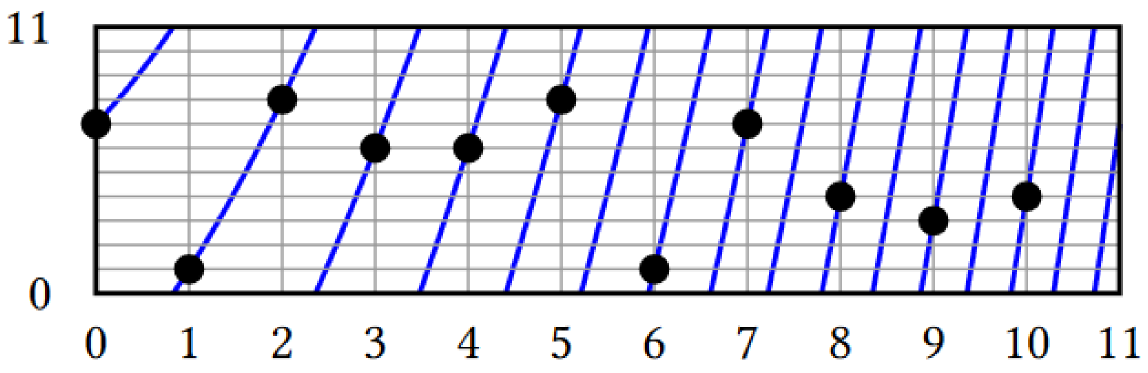
Inductive case ($k + 1$ case holds): $k \leq d$ points in $\mathcal{P}$. Let $x^* \in \mathbb{Z}_p \neq x_i$ for all i. Every polynomial must give some value when evaluated at x^*.

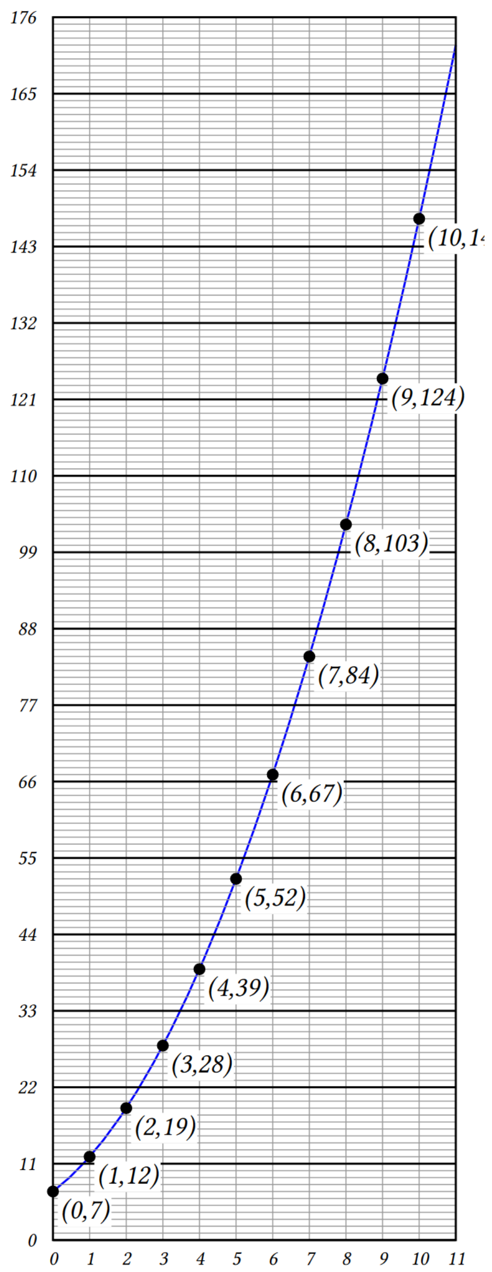
Compute [# of degree-d polynomials pass through points in $\mathcal{P}$] = $\sum_{y^* \in \mathbb{Z}_p}$[# of degree−$d$ polynomials pass through points in $\mathcal{P} \cup \{(x^*, y^*)\}$] = $\sum_{y^* \in \mathbb{Z}_p} p^{d+1-(k+1)} = p \cdot p^{d+1-k-1} = p^{d+1-k}$

An Example

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

- mod 11?
  - We care only about $\mathbb{Z}_{11}$ inputs to f
  - Just the 11 highlighted points alone (not the blue curve)





Shamir Secret Sharing

- Any $d+1$ points on a degree-d polynomial are enough to uniquely reconstruct the polynomial.
- 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.
- Now, any t shares can uniquely determine the polynomial f, and hence recover the secret $f(0)$.

Shamir Secret Sharing

- $\mathcal{M} = \mathbb{Z}_p$, where p is a prime. $n < p$, $t \le n$
- Share(m):
  - $f_1, \dots, f_{t-1} \leftarrow \mathbb{Z}_p$
  - $f(x) \coloneqq m + \sum_{j=1}^{t-1} f_j x^j$
  - Let $s_i = (i, f(i)\%p)$ for $i = 1$ to n. Return $(s_1, \dots, s_n)$.
- Reconstruct$(\{\, s_i \mid i \in U \,\})$:
  - $f(x) \coloneqq$ unique degree $-$ $(t-1)$ polynomial mod p passing through points $\{s_i \mid i \in U\}$
  - Return $f(0)$.

Shamir Secret Sharing - Security

Lemma Let p be a prime and define the following two libraries. These two libraries are interchangeable, i.e.,$\mathcal{L}_{\text{shamir-real}} \equiv \mathcal{L}_{\text{shamir-rand}}$.

$\mathcal{L}_{\text{shamir-real}}$
poly$(m, t, U \subseteq \{1, \dots, p\})$: If $\lvert U\rvert \geq t$: return err $f_1, \dots, f_{t-1} \leftarrow \mathbb{Z}_p$ $f(x) \coloneqq m + \sum_{j=1}^{t-1} f_j x^j$ For $i \in U$: $s_i \coloneqq (i, f(i)\%p)$ return $\{s_i \mid i \in U\}$

$\mathcal{L}_{\text{shamir-rand}}$
poly$(m, t, U \subseteq \{1, \dots, p\})$: If $\lvert U\rvert \geq t$: return err For $i \in U$: $y_i \leftarrow \mathbb{Z}_p$ $s_i \coloneqq (i, y_i)$ return $\{s_i \mid i \in U\}$

Shamir Secret Sharing - Security

Lemma Let p be a prime and define the following two libraries. These two libraries are interchangeable, i.e., $\mathcal{L}_{\text{shamir-real}} \equiv \mathcal{L}_{\text{shamir-rand}}$.

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

poly$(m, t, U \subseteq \{1, \ldots, p\})$:
    If $\lvert U\rvert \geq t$: return err
    $f_1, \ldots, f_{t-1} \leftarrow \mathbb{Z}_p$
    $f(x) := m + \sum_{j=1}^{t-1} f_j x^j$
    For $i \in U$:
        $s_i := (i, f(i)\%p)$
    return $\{s_i \mid i \in U\}$

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

poly$(m, t, U \subseteq \{1, \ldots, p\})$:
    If $\lvert U\rvert \geq t$: return err
    For $i \in U$:
    $y_i \leftarrow \mathbb{Z}_p$
    $s_i := (i, y_i)$
    return $\{s_i \mid i \in U\}$

Proof

Fix $m \in \mathbb{Z}_p$, fix set U with $|U| < t$.

For each $i \in U$, fix a value $y_i \in \mathbb{Z}_p$.

Consider the probability that **poly**(m, t, U) outputs $\{(i, y_i) \mid i \in U\}$ in each library.

- In $\mathcal{L}_{\text{shamir-real}}$, there are p^{t-1} such degree-$(t-1)$ polynomials (according to the previous corollary) such that $f(0) \equiv_p m$.
  - To be consistent with $(0, m) \cup \{(i, y_i) \mid i \in U\}$, there are $p^{t-(|U|+1)}$ such polynomials.
  - Happen with probability $\frac{p^{t-|U|-1}}{p^{t-1}} = p^{-|U|}$.

Shamir Secret Sharing - Security

Lemma Let p be a prime and define the following two libraries. These two libraries are interchangeable, i.e., $\mathcal{L}_{\text{shamir-real}} \equiv \mathcal{L}_{\text{shamir-rand}}$.

Proof

Fix $m \in \mathbb{Z}_p$, fix set U with $|U| < t$.

For each $i \in U$, fix a value $y_i \in \mathbb{Z}_p$.

Consider the probability that **poly**(m, t, U) outputs $\{(i, y_i) \mid i \in U\}$ in each library.

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

poly$(m, t, U \subseteq \{1, \dots, p\})$:
    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 $\{s_i \mid i \in U\}$

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

poly$(m, t, U \subseteq \{1, \dots, p\})$:
    If $\|U\| \geq t$: return err
    For $i \in U$:
    $y_i \leftarrow \mathbb{Z}_p$
    $s_i := (i, y_i)$
    return $\{s_i \mid i \in U\}$

- In $\mathcal{L}_{\text{shamir-rand}}$, $|U|$ output values are chosen uniformly in $\mathbb{Z}_p$, $p^{|U|}$ ways to choose them, but only one cause **poly**(m, t, U) to output specific choice of $\{(i, y_i) \mid i \in U\}$. 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.

Shamir Secret Sharing - Security

Theorem Shamir's secret-sharing scheme is secure.

proof

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

$\textsc{share}(m_L, m_R, U)$:
- 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 $\{s_i \mid i \in U\}$

$\textsc{share}(m_L, m_R, U)$:
- return $\textsc{poly}(m_L, t, U)$

$\diamond$

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

$\textsc{poly}(m, t, U)$:
- 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 $\{s_i \mid i \in U\}$

Shamir Secret Sharing - Security

Theorem Shamir's secret-sharing scheme is secure.

proof

$\underline{\textsc{share}(m_L, m_R, U)\text{:}}$

  return $\textsc{poly}(m_L, t, U)$

$\diamond$

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

$\underline{\textsc{poly}(m, t, U)\text{:}}$

  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 $\{s_i \mid i \in U\}$

$\Rightarrow$

$\underline{\textsc{share}(m_L, m_R, U)\text{:}}$

  return $\textsc{poly}(m_L, t, U)$

$\diamond$

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

$\underline{\textsc{poly}(m, t, U)\text{:}}$

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

Shamir Secret Sharing - Security

Theorem Shamir's secret-sharing scheme is secure.

proof

$\textsc{share}(m_L, m_R, U)$:
return $\textsc{poly}(m_L, t, U)$

$\diamond$

$\mathcal{L}_{\text{shamir-rand}}$
$\textsc{poly}(m, t, U)$:
if $\lvert U \rvert \geqslant t$: return err
for $i \in U$:
$y_i \leftarrow \mathbb{Z}_p$
$s_i := (i, y_i)$
return $\{s_i \mid i \in U\}$

$\Rightarrow$

$\textsc{share}(m_L, m_R, U)$:
return $\textsc{poly}(\boxed{m_R}, t, U)$

$\diamond$

$\mathcal{L}_{\text{shamir-rand}}$
$\textsc{poly}(m, t, U)$:
if $\lvert U \rvert \geqslant t$: return err
for $i \in U$:
$y_i \leftarrow \mathbb{Z}_p$
$s_i := (i, y_i)$
return $\{s_i \mid i \in U\}$

Shamir Secret Sharing - Security

Theorem Shamir's secret-sharing scheme is secure.

proof

$\textsc{share}(m_L, m_R, U)$:
  return $\textsc{poly}(m_R, t, U)$

$\diamond$

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

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

$\textsc{share}(m_L, m_R, U)$:
  return $\textsc{poly}(m_R, t, U)$

$\diamond$

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

$\textsc{poly}(m, t, U)$:
  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 $\{s_i \mid i \in U\}$

Shamir Secret Sharing - Security

Theorem Shamir's secret-sharing scheme is secure.

proof

SHARE(m_L, m_R, U):
  return POLY(m_R, 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 $\{s_i \mid i \in U\}$

$\Rightarrow$

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

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

Basing Cryptography on Intractable Computations

What Qualifies as a "Computationally Infeasible" Attack?

- Intuition

  It doesn't really matter whether attacks are impossible, only whether attacks are computationally infeasible.

- For a scheme with λ-bit keys
  - The most direct computation procedure would be for the enemy to try all 2^{λ} possible keys, one by one. Obviously this is easily made impractical for the enemy by simply choosing λ large enough.

  - We call λ the security parameter of the scheme. A scheme described as having n-bit security if the best known attack requires 2^n steps.
    - A scheme with λ-bit keys may have attack that cost only $2^{\lambda/2}$.

What Qualifies as a "Computationally Infeasible" Attack?

clock cycles	*approx cost*	*reference*
2^{50}	*\$3.50*	*cup of coffee*
2^{55}	*\$100*	*decent tickets to a Portland Trailblazers game*
2^{65}	*\$130,000*	*median home price in Oshkosh, WI*
2^{75}	*\$130 million*	*budget of one of the Harry Potter movies*
2^{85}	*\$140 billion*	*GDP of Hungary*
2^{92}	*\$20 trillion*	*GDP of the United States*
2^{99}	*\$2 quadrillion*	*all of human economic activity since 300,000 BC*[4]
2^{128}	*really a lot*	*a billion human civilizations' worth of effort*

Asymptotic Running Time

Definition A program runs in polynomial time if there exists a constant $c > 0$ such that for all sufficiently long input strings x, the program stops after no more than $O(|x|^c)$ steps.

- In crypto world, “polynomial-time” is a synonym for “efficient”.
  - Polynomial time is not a perfect match to what we mean when we informally talk about “efficient” algorithms.
    - Algorithms with running time $\Theta(n^{1000})$ are technically polynomial-time, while those with running time $\Theta(n^{\log\log\log n})$ aren’t.
- Closure property: repeating a polynomial-time process a polynomial number of times results in a polynomial-time process overall.

Potential Pitfall: Numerical Algorithms

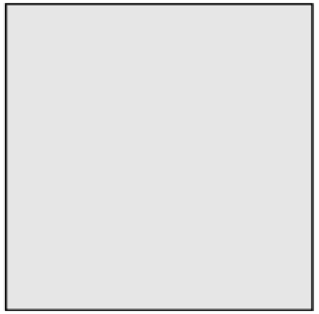
- Remember that representing the number N on a computer requires only $\sim \log_2 N$ bits. This means that $\log_2 N$, rather than N, is our security parameter.
  - The difference between running time $O(\log N)$ and $O(N)$ is the difference between writing down a number and counting to the number.

Efficient algorithm known:	**No known efficient algorithm:**
Computing GCDs	Factoring integers
Arithmetic mod N	Computing $\phi(N)$ given N
Inverses mod N	Discrete logarithm
Exponentiation mod N	Square roots mod composite N

What qualifies as a “Negligible” Success Probability?

- We don’t want to worry about attacks that are as expensive as a brute-force attack. (“Computationally Infeasible” Attack)
- We don’t want to worry about attacks whose success probability is as low as a blind-guess attack. (“Negligible” Success Probability)

probability	*equivalent*
2^{-10}	*full house in 5-card poker* 满堂红（三张相同牌加对子）
2^{-20}	*royal flush in 5-card poker* 皇家同花顺
2^{-28}	*you win this week’s Powerball jackpot*
2^{-40}	*royal flush in 2 consecutive poker games*
2^{-60}	*the next meteorite that hits Earth lands in this square* $\longrightarrow$



What qualifies as a "Negligible" Success Probability?

- For example, $1/2^\lambda$ approaches zero so fast that no polynomial can "rescue" it, i.e., $\lim_{\lambda \to \infty} \frac{p(\lambda)}{2^\lambda} = 0$ for polynomial p.
  - In other words, it approaches zero faster than 1 over any polynomial.

What qualifies as a “Negligible” Success Probability?

Definition A function f is negligible if, for every polynomial p, we have $\lim_{\lambda\to\infty} p(\lambda)f(\lambda) = 0$.

Claim If for every integer c, $\lim_{\lambda\to\infty} \lambda^c f(\lambda) = 0$, then f is negligible.

What qualifies as a "Negligible" Success Probability?

Claim If for every integer c, $\lim_{\lambda\to\infty} \lambda^c f(\lambda) = 0$, then f is negligible.

Proof

Suppose f has this property, and take arbitrary polynomial p, show that $\lim_{\lambda\to\infty} p(\lambda) f(\lambda) = 0$.

If d is the degree of p, then $\lim_{\lambda\to\infty} p(\lambda)/\lambda^{d+1} = 0$. Hence,

$$\lim_{\lambda\to\infty} p(\lambda) f(\lambda) = \lim_{\lambda\to\infty} [\frac{p(\lambda)}{\lambda^{d+1}} \left(\lambda^{d+1} \cdot f(\lambda)\right)]$$

$$= \left(\lim_{\lambda\to\infty} \frac{p(\lambda)}{\lambda^{d+1}}\right) \left(\lim_{\lambda\to\infty} \lambda^{d+1} \cdot f(\lambda)\right) = 0 \cdot 0 = 0$$

What qualifies as a "Negligible" Success Probability?

Definition If $f, g : \mathbb{N} \to \mathbb{R}$ are two functions, we write $f \approx g$ to mean that $|f(\lambda) - g(\lambda)|$ is a negligible function.

$$
\begin{aligned}
\Pr[X] \approx 0 \quad &\Leftrightarrow \quad \text{“event } X \text{ almost never happens”} \\
\Pr[Y] \approx 1 \quad &\Leftrightarrow \quad \text{“event } Y \text{ almost always happens”} \\
\Pr[A] \approx \Pr[B] \quad &\Leftrightarrow \quad \text{“events } A \text{ and } B \text{ happen with} \\
&\qquad\quad \text{essentially the same probability”}
\end{aligned}
$$

Indistinguishability

Definition Let $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ be two libraries with a common interface. We say that $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ are indistinguishable, and write $\mathcal{L}_{\text{left}} \approx \mathcal{L}_{\text{right}}$, if for all polynomial-time programs $\mathcal{A}$ that output a single bit, $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] \approx \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]$.

We call the quantity $|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]|$ the advantage or bias of $\mathcal{A}$ in distinguishing $\mathcal{L}_{\text{left}}$ from $\mathcal{L}_{\text{right}}$. Two libraries are therefore indistinguishable if all polynomial-time calling programs have negligible advantage in distinguishing them.

Indistinguishability

- A very simple example of two indistinguishable libraries:

$\mathcal{L}_{\text{left}}$
$\textsc{predict}(x)$:
$s \leftarrow \{0,1\}^{\lambda}$
return $x \stackrel{?}{=} s$

$\mathcal{L}_{\text{right}}$
$\textsc{predict}(x)$:
return `false`

- The $\mathcal{L}_{\text{left}}$ library tells the calling program whether its prediction was correct.
- The $\mathcal{L}_{\text{right}}$ library doesn't even bother sampling a string, it just always says "*sorry, your prediction was wrong.*"

Indistinguishability

- A very simple example of two indistinguishable libraries:

$\mathcal{L}_{\text{left}}$

PREDICT(x):
- $s \leftarrow \{0,1\}^\lambda$
- return $x \stackrel{?}{=} s$

$\mathcal{L}_{\text{right}}$

PREDICT(x):
- return false

$\mathcal{A}_{obvious}$

```
do q times:
  if PREDICT(0^λ) = true
    return 1
return 0
```

$\Pr[\mathcal{A}_{obvious} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1] = 0$

$\Pr[\mathcal{A}_{obvious} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1]$
$= 1 - \Pr[\textit{all } q \textit{ independent calls to } \text{PREDICT } \textit{return } \text{false}]$
$= 1 - \left(1 - \frac{1}{2^\lambda}\right)^q$

Then $\mathcal{L}_{\text{left}} \not\equiv \mathcal{L}_{\text{right}}$. These two libraries are not interchangeable.

What about indistinguishability?

Compute an upper bound first

Indistinguishability

- A very simple example of two indistinguishable libraries:

$\mathcal{L}_{\text{left}}$

$\text{PREDICT}(x)$:
$\quad s \leftarrow \{0,1\}^\lambda$
$\quad \text{return } x \stackrel{?}{=} s$

$\mathcal{L}_{\text{right}}$

$\text{PREDICT}(x)$:
$\quad \text{return false}$

$\mathcal{A}_{\text{obvious}}$

do q times:
$\quad$ if $\text{PREDICT}(0^\lambda) = \text{true}$
$\quad\quad$ return 1
return 0

$$\Pr[\mathcal{A}_{obvious} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1] = 0$$

$$\Pr[\mathcal{A}_{obvious} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1]$$
$$\leq \Pr[\textit{first call to PREDICT returns true}]$$
$$+ \Pr[\textit{second call to PREDICT returns true}] + \cdots = q\frac{1}{2^\lambda}$$

- This is an overestimate of some probabilities (e.g., if the first call to predict returns true, then the second call isn't made).
- $\mathcal{A}_{obvious}$ has advantage at most $q/2^\lambda$. Since $\mathcal{A}_{obvious}$ runs in polynomial time, it can only make a polynomial number q of queries to the library, so $q/2^\lambda$ is negligible.
- To show that the libraries are indistinguishable, we must show that every calling program's advantage is negligible. (prove later)

Other Properties

Lemma If $\mathcal{L}_1 \equiv \mathcal{L}_2$ then $\mathcal{L}_1 \approx \mathcal{L}_2$. If $\mathcal{L}_1 \approx \mathcal{L}_2 \approx \mathcal{L}_3$ then $\mathcal{L}_1 \approx \mathcal{L}_3$.

Lemma If $\mathcal{L}_{\text{left}} \approx \mathcal{L}_{\text{right}}$ then $\mathcal{L}^* \diamond \mathcal{L}_{\text{left}} \approx \mathcal{L}^* \diamond \mathcal{L}_{\text{right}}$ for any polynomial-time library $\mathcal{L}^*$.

Bad-Event Lemma

Lemma Let $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ be libraries that each define a variable named '*bad*' that is initialized to 0. If $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ have identical code, except for code blocks reachable only when $bad = 1$, then

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr\left[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1\right]\right| \leq \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}}\ sets\ bad = 1]$$

proof

Fix an arbitrary calling program $\mathcal{A}$. Define the following events

$\mathcal{B}_{\text{left}}$: the event that $\mathcal{A} \diamond \mathcal{L}_{\text{left}}$ sets bad to 1 at some point.

$\mathcal{B}_{\text{right}}$: the event that $\mathcal{A} \diamond \mathcal{L}_{\text{right}}$ sets bad to 1 at some point.

Bad-Event Lemma

Lemma Let $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ be libraries that each define a variable named '*bad*' that is initialized to 0. If $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ have identical code, except for code blocks reachable only when $bad = 1$, then

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]\right| \leq \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}}\ sets\ bad = 1]$$

proof

$\mathcal{B}_{\text{left}}$: the event that $\mathcal{A} \diamond \mathcal{L}_{\text{left}}$ sets bad to 1 at some point.

$\mathcal{B}_{\text{right}}$: the event that $\mathcal{A} \diamond \mathcal{L}_{\text{right}}$ sets bad to 1 at some point.

$$\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \mathcal{B}_{\text{left}}]\Pr[\mathcal{B}_{\text{left}}] + \Pr[\mathcal{A} \diamond \mathcal{L} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}]\Pr[\overline{\mathcal{B}_{\text{left}}}]$$

$$\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \mathcal{B}_{\text{right}}]\Pr[\mathcal{B}_{\text{right}}] + \Pr[\mathcal{A} \diamond \mathcal{L} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{right}}}]\Pr[\overline{\mathcal{B}_{\text{right}}}]$$

We have $\Pr[\mathcal{B}_{\text{left}}]=\Pr[\mathcal{B}_{\text{right}}]$, because $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ have identical code, except for code blocks reachable only when $bad = 1$. Let $p^* =_{\text{def}} \Pr[\mathcal{B}_{\text{left}}] = \Pr[\mathcal{B}_{\text{right}}]$.

$advantage_{\mathcal{A}} = \left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]\right| = |p^*(\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \mathcal{B}_{\text{left}}] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \mathcal{B}_{\text{right}}] + (1 - p^*)\left(\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}]\right)|$

Bad-Event Lemma

Lemma Let $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ be libraries that each define a variable named 'bad' that is initialized to 0. If $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ have identical code, except for code blocks reachable only when $bad = 1$, then

$$\left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]\right| \leq \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}}\ sets\ bad = 1]$$

proof

Let $p^* =_{\text{def}} \Pr[\mathcal{B}_{\text{left}}] = \Pr[\mathcal{B}_{\text{right}}]$.

$$advantage_{\mathcal{A}} = \left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]\right|$$

$$= |p^*(\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \mathcal{B}_{\text{left}}] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \mathcal{B}_{\text{right}}]$$

$$+\ (1 - p^*)\left(\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}]\right)|$$

We already have $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \overline{\mathcal{B}_{\text{left}}}]$:

$$advantage_{\mathcal{A}} = p^*|\left(\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1 \mid \mathcal{B}_{\text{left}}] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1 \mid \mathcal{B}_{\text{right}}]\right|$$

Hence, $advantage_{\mathcal{A}} \leq p^* = \Pr[\mathcal{B}_{\text{left}}] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \text{ sets bad} = 1]$

Return to the Example

- $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ are indistinguishable.

$\mathcal{L}_{\text{left}}$
$\underline{\textsc{predict}(x):}$
$s \leftarrow \{\texttt{0}, \texttt{1}\}^\lambda$
return $x \overset{?}{=} s$

$\mathcal{L}_{\text{right}}$
$\underline{\textsc{predict}(x):}$
return $\texttt{false}$

Return to the Example

- $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ are indistinguishable.

$\mathcal{L}_{\text{left}}$

```
PREDICT(x):
  s ← {0,1}^λ
  return x =? s
```

$\equiv$

$\mathcal{L}_{\text{hyb-L}}$

```
bad := 0

PREDICT(x):
  s ← {0,1}^λ
  if x = s:
    bad := 1
    return true
  return false
```

$\approx$

$\mathcal{L}_{\text{hyb-R}}$

```
bad := 0

PREDICT(x):
  s ← {0,1}^λ
  if x = s:
    bad := 1

  return false
```

$\equiv$

$\mathcal{L}_{\text{right}}$

```
PREDICT(x):
  return false
```

$\mathcal{L}_{\text{hyb-L}} \approx \mathcal{L}_{\text{hyb-R}}$: $\left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{hyb-L}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{hyb-R}} \Rightarrow 1]\right| \leq \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{hyb-L}}\ sets\ bad = 1]$
$\mathcal{A} \diamond \mathcal{L}_{\text{hyb-L}}\ sets\ bad = 1$ only if s is successfully predicted, which happens at most $q/2^{\lambda}$, which is negligible when $\mathcal{A}$ runs in polynomial time.

Birthday Probabilities

- Taking q independent, uniform samples from a set of N items. What is the probability that the same value gets chosen more than once? In other words, what is the probability that the following program outputs 1?

$\mathcal{B}(q, N)$

for $i := 1$ to q:
  $s_i \leftarrow \{1, \ldots, N\}$
  for $j := 1$ to $i - 1$:
    if $s_i = s_j$ then return 1
return 0

- $\text{BirthdayProb}(q, N) =_{\text{def}} \Pr[\mathcal{B}(q, N) \text{ outputs } 1]$

Birthday Probabilities

Lemma $\mathrm{BirthdayProb}(q, N) = 1 - \prod_{i=1}^{q-1}(1 - \frac{i}{N})$.

Proof

We instead compute the probability that $\mathcal{B}$ outputs 0. In order for $\mathcal{B}$ to output 0, it must avoid the early termination conditions in each iteration.

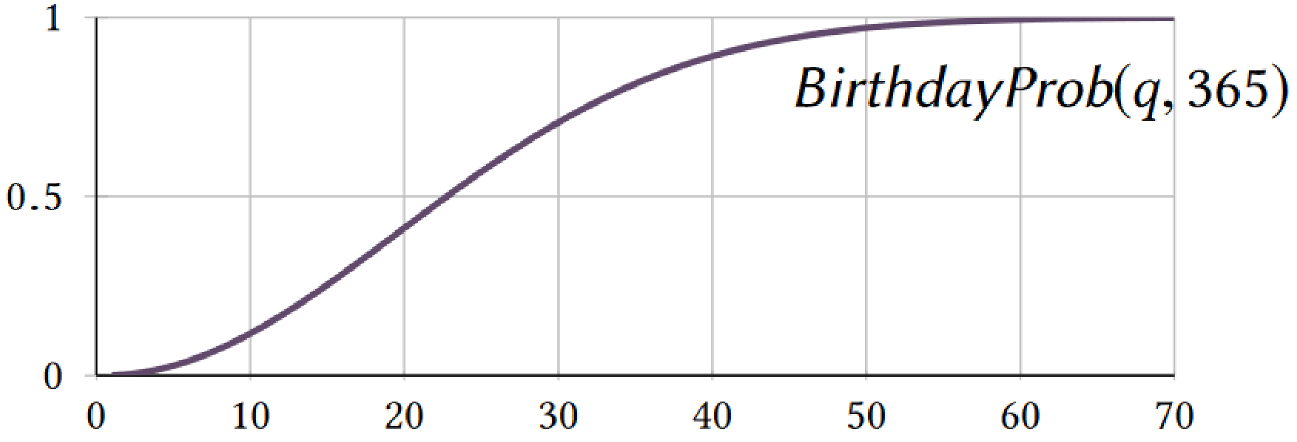
$$\begin{aligned}&\Pr[\mathcal{B}(q, N) \text{ outputs } 0]\\ &= \Pr[\mathcal{B}(q, N) \text{ doesn't terminate early in interation } i = 1] \times \cdots\\ &\times \Pr[\mathcal{B}(q, N) \text{ doesn't terminate early in interation } i = q]\end{aligned}$$

We have $\Pr[\mathcal{B}(q, N) \text{ doesn't terminate early in interation } i] = 1 - \frac{i-1}{N}$.

$$\begin{aligned}&\mathrm{BirthdayProb}(q, N) = \Pr[\mathcal{B}(q, N) \text{ outputs } 1] = 1 - \Pr[\mathcal{B}(q, N) \text{ outputs } 0]\\ &= 1 - \left(1 - \frac{1}{N}\right) \times \cdots \times \left(1 - \frac{q-1}{N}\right) = 1 - \prod_{i=1}^{q-1}(1 - \frac{i}{N})\end{aligned}$$

Birthday Probabilities

- Plotting $\mathrm{BirthdayProb}(q, 365)$



- With only $q = 23$ people, the probability already exceeds 50%.
- $q = 70$, the probability exceeds 99.9%.

Asymptotic Bounds on the Birthday Probability

We are most interested in the case where q is relatively small compared to N (e.g., when q is a polynomial function of λ but N is exponential).

Lemma if $q \leq \sqrt{2N}$, then $0.632\frac{q(q-1)}{2N} \leq \text{BirthdayProb}(q, N) \leq \frac{q(q-1)}{2N}$. This means that $\text{BirthdayProb}(q, N) = \Theta(\frac{q^2}{N})$