Advanced Cryptography

（Provable Security）

Yi LIU

"Real-vs-Random" Style of Security Definition

Consider: the attacker is a calling program to the following subroutines.

ctxt$(m \in \Sigma.\mathcal{M})$: $k \leftarrow \Sigma.\mathrm{KeyGen}$ $c := \Sigma.\mathrm{Enc}(k, m)$ return c	Vs.	**ctxt**$(m \in \Sigma.\mathcal{M})$: $c \leftarrow \Sigma.\mathcal{C}$ return c

No calling program should have any way of determining which of these two implementations is answering subroutine calls.

"an encryption scheme is a good one if, when you plug its KeyGen *and* Enc *algorithms into the template of the* **ctxt** *subroutine above, the two implementations of* **ctxt** *induce identical behavior in every calling program."*

“Real-vs-Random” Style of Security Definition

- One-time pad satisfies the new security property

ctxt(m): $k \leftarrow \{0,1\}^{\lambda}$ $c := k \oplus m$ return c	Vs.	**ctxt**(m): $c \leftarrow \{0,1\}^{\lambda}$ return c

"Left-vs-Right" Style of Security Definition

An intuitive idea

"an encryption scheme is a good one if encryptions of m_L look like encryptions of m_R to an attacker when each key is secret and used to encrypt only one plaintext, even when the attacker chooses m_L and m_R."

How to express these details?

Consider: the attacker is a calling program to the following subroutine.

eavesdrop$(m_L, m_R \in \Sigma.\mathcal{M})$: $\quad k \leftarrow \Sigma.\text{KeyGen}$ $\quad c := \Sigma.\text{Enc}(k, m_L)$ $\quad$return c	Vs.	**eavesdrop**$(m_L, m_R \in \Sigma.\mathcal{M})$: $\quad k \leftarrow \Sigma.\text{KeyGen}$ $\quad c := \Sigma.\text{Enc}(k, m_R)$ $\quad$return c

“Left-vs-Right” Style of Security Definition

Consider: the attacker is a calling program to the following subroutines.

eavesdrop$(m_L, m_R \in \Sigma.\mathcal{M})$: $k \leftarrow \Sigma.\text{KeyGen}$ $c := \Sigma.\text{Enc}(k, m_L)$ return c	Vs.	**eavesdrop**$(m_L, m_R \in \Sigma.\mathcal{M})$: $k \leftarrow \Sigma.\text{KeyGen}$ $c := \Sigma.\text{Enc}(k, m_R)$ return c

No calling program should have any way of determining which of these two implementations is answering subroutine calls.

“an encryption scheme is a good one if, when you plug its KeyGen *and* Enc *algorithms into the template of the* **eavesdrop** *subroutine above, the two implementations of* **eavesdrop** *induce identical behavior in every calling program.”*

"Left-vs-Right" Style of Security Definition

- One-time pad satisfies the new security property

eavesdrop(m_L, m_R): $k \leftarrow \{0,1\}^\lambda$ $c := k \oplus m_L$ return c	Vs.	**eavesdrop**(m_L, m_R): $k \leftarrow \{0,1\}^\lambda$ $c := k \oplus m_R$ return c

Two Styles

real-vs-random paradigm and *left-vs-right paradigm*

Q: Is one “correct” and the other one “incorrect?

Discuss later.

Formalisms for Security Definitions

- We've defined security in terms of a single, self-contained subroutine, and imagined the attacker as a program that calls this subroutine.
- We will need to generalize beyond a single subroutine, to a collection of subroutines that share common (private) state information.

Formalisms for Security Definitions

Definition (Libraries)

- A library $\mathcal{L}$ is a collection of subroutines and private/static variables.
- A library's interface consists of the names, argument types, and output type of all of its subroutines.
- If a program $\mathcal{A}$ includes calls to subroutines in the interface of $\mathcal{L}$, then we write $\mathcal{A} \diamond \mathcal{L}$ to denote the result of linking $\mathcal{A}$ to $\mathcal{L}$ in the natural way (answering those subroutine calls using the implementation specified in $\mathcal{L}$).
- We write $\mathcal{A} \diamond \mathcal{L} \Rightarrow z$ to denote the event that program $\mathcal{A} \diamond \mathcal{L}$ outputs the value z.

Formalisms for Security Definitions

Example

- A library $\mathcal{L}$
- A calling program $\mathcal{A}$

$\mathcal{A}$
$m \leftarrow \{0,1\}^\lambda$
$c := \textbf{ctxt}(m)$
return $m =_? c$

$\mathcal{L}$
$\textbf{ctxt}(m)$:
$\quad k \leftarrow \{0,1\}^\lambda$
$\quad c := k \oplus m$
$\quad$ return c

We have $\Pr[\mathcal{A} \diamond \mathcal{L} \Rightarrow true] = 1/2^\lambda$.

Formalisms for Security Definitions

$\mathcal{L}$

$s \leftarrow \{0,1\}^\lambda$
reset():
    $s \leftarrow \{0,1\}^\lambda$
guess($x \in \{0,1\}^\lambda$):
    return $x =_? s$

Example

- A library $\mathcal{L}$
  - Code outside of a subroutine (e.g., the first line) is run once at the initialization time.
  - Variables defined at initiation time (like s here) are available in all subroutine scopes (but not to the calling program).

Interchangeability

Definition Let $\mathcal{L}_{left}$ and $\mathcal{L}_{right}$ be two libraries that have the same interface. We say that $\mathcal{L}_{left}$ and $\mathcal{L}_{right}$ are interchangeable, and write $\mathcal{L}_{left} \equiv \mathcal{L}_{right}$, if for all programs $\mathcal{A}$ that output a Boolean value,

$$\Pr[\mathcal{A} \diamond \mathcal{L}_{left} \Rightarrow true] = \Pr[\mathcal{A} \diamond \mathcal{L}_{right} \Rightarrow true]$$

- We often refer to the calling program as a distinguisher.
- A Boolean output is enough for that task. Think of the output bit as the calling program's "guess" for which library the calling program thinks it is linked to.
- The distinction between "calling program outputs true" and "calling program outputs false" is not significant.

Interchangeability - Example

```
FOO(x):
  if x is even:
     return 0
  else if x is odd:
     return 1
  else:
     return -1
```

$\equiv$

```
FOO(x):
  if x is even:
     return 0
  else if x is odd:
     return 1
  else:
     return ∞
```

Their only difference happens in an un-reachable block of code.

Interchangeability - Example

$\underline{\textsc{foo}(x):}$ $\quad$ return $\textsc{bar}(x, \boxed{x})$ $\underline{\textsc{bar}(a, b):}$ $\quad k \leftarrow \{0, 1\}^\lambda$ $\quad$ return $k \oplus a$	$\equiv$	$\underline{\textsc{foo}(x):}$ $\quad$ return $\textsc{bar}(x, \boxed{0^\lambda})$ $\underline{\textsc{bar}(a, b):}$ $\quad k \leftarrow \{0, 1\}^\lambda$ $\quad$ return $k \oplus a$

Their only difference is the value they assign to a variable that is never actually used.

Interchangeability - Example

```
FOO(x, n):                    FOO(x, n):
  for i = 1 to λ:        ≡      for i = 1 to n:
    BAR(x, i)                     BAR(x, i)
                                for i = n + 1 to λ:
                                  BAR(x, i)
```

Their only difference is that one library unrolls a loop that occurs in the other library.

Interchangeability - Example

$\textsc{foo}(x)$:
$k \leftarrow \{0,1\}^\lambda$
$y \leftarrow \{0,1\}^\lambda$
return $k \oplus y \oplus x$

$\equiv$

$\textsc{foo}(x)$:
$k \leftarrow \{0,1\}^\lambda$
return $k \oplus \textsc{bar}(x)$

$\textsc{bar}(x)$:
$y \leftarrow \{0,1\}^\lambda$
return $y \oplus x$

Their only difference is that one library inlines a subroutine call that occurs in the other library.

Interchangeability - Example

$$\boxed{\begin{array}{l}\underline{\textsc{foo}():}\\ \quad x \leftarrow \{0,1\}^\lambda\\ \quad y \leftarrow \{0,1\}^\lambda\\ \quad \text{return } x\|y\end{array}} \equiv \boxed{\begin{array}{l}\underline{\textsc{foo}():}\\ \quad z \leftarrow \{0,1\}^{2\lambda}\\ \quad \text{return } z\end{array}}$$

The uniform distribution over strings acts independently on different characters in the string (“‖” refers to concatenation).

Interchangeability - Example

$k \leftarrow \{0,1\}^\lambda$ $\underline{\textsc{foo}(x)\text{:}}$ $\quad$ return $k \oplus x$	$\equiv$	$\underline{\textsc{foo}(x)\text{:}}$ $\quad$ if k not defined: $\quad\quad k \leftarrow \{0,1\}^\lambda$ $\quad$ return $k \oplus x$

Sampling a value "eagerly" (as soon as possible) vs. sampling a value "lazily" (at the last possible moment before the value is needed). We assume that k is static/global across many calls to FOO, *and initially undefined.*

Formal Restatements of Previous Concepts

- "real-vs-random" style
  - **Definition** An encryption scheme Σ has one-time uniform ciphertexts if:

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

$\equiv$

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

Formal Restatements of Previous Concepts

- "left-vs-right" style
  - **Definition** An encryption scheme Σ has one-time secrecy if:

$\mathcal{L}^{\Sigma}_{\text{ots-L}}$		$\mathcal{L}^{\Sigma}_{\text{ots-R}}$
eavesdrop$(m_L, m_R \in \Sigma.\mathcal{M})$: $k \leftarrow \Sigma.\text{KeyGen}$ $c := \Sigma.\text{Enc}(k, m_L)$ return c	$\equiv$	**eavesdrop**$(m_L, m_R \in \Sigma.\mathcal{M})$: $k \leftarrow \Sigma.\text{KeyGen}$ $c := \Sigma.\text{Enc}(k, m_R)$ return c

Formal Restatements of One-Time Pad

- “real-vs-random” style
  - **Claim** One-time pad satisfies the one-time uniform ciphertexts property. In other words:

$\mathcal{L}_{\text{otp-real}}$
ctxt$(m \in \{0,1\}^\lambda)$:
$\quad k \leftarrow \{0,1\}^\lambda$
$\quad$ return $k \oplus m$

$\equiv$

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

- Also satisfies the one-time secrecy property (“left-vs-right” style)

Kerckhoffs' Principle, Revisited

- The definition of interchangeability considers literally every calling program, so it must also consider calling programs like yours that "know" what algorithms are being used.

- Assume that the attacker knows every fact in the universe, except for:
    1. which of the two possible libraries it is linked to in any particular execution
    2. the random choices that the library will make during any particular execution (which are usually assigned to privately scoped variables within the library).

How to Demonstrate Insecurity with Attacks

- If the two libraries that you get (after calling in the specifics of a particular scheme) are interchangeable, then we say that the scheme satisfies the security property.

- If we want to show that some scheme is insecure, we have to demonstrate just one calling program that behaves differently in the presence of those two libraries.

Demonstrate Insecurity with Attacks

Insecure construction

- KeyGen: return $k \leftarrow \{0,1\}^{\lambda}$
- $\text{Enc}(k, m)$: return k & m
- Cannot satisfy the correctness…But pretend we haven't noticed that yet.

Claim This construction does not have one-time uniform ciphertexts.

Demonstrate Insecurity with Attacks

Insecure construction

- KeyGen: return $k \leftarrow \{0,1\}^\lambda$
- Enc(k, m): return k & m

Claim This construction does not have one-time uniform ciphertexts.

Proof

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

$\equiv$

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

Demonstrate Insecurity with Attacks

Insecure construction

- KeyGen: return $k \leftarrow \{0,1\}^\lambda$
- $\text{Enc}(k, m)$: return $k \;\&\; m$

Claim This construction does not have one-time uniform ciphertexts.

Proof

$\mathcal{A}$
$c := \textbf{ctxt}(0^\lambda)$ return $c =_? 0^\lambda$

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

$\equiv_?$

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

Demonstrate Insecurity with Attacks

Insecure construction

- KeyGen: return $k \leftarrow \{0,1\}^\lambda$
- Enc(k, m): return $k \,\&\, m$

Claim This construction does not have one-time uniform ciphertexts.

Proof

$\mathcal{A}$
$c := \mathbf{ctxt}(0^\lambda)$ return $c =_? 0^\lambda$

$\diamond$

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

$$\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots\$-real}} \Rightarrow \mathit{true}] = 1$$

Demonstrate Insecurity with Attacks

Insecure construction

- KeyGen: return $k \leftarrow \{0,1\}^\lambda$
- Enc(k, m): return k & m

Claim This construction does not have one-time uniform ciphertexts.

Proof

$\mathcal{A}$
$c \coloneqq \textbf{ctxt}(0^\lambda)$ return $c =_? 0^\lambda$

$\diamond$

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

$\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots\$-real}} \Rightarrow \mathit{true}] = 1$

$\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \Rightarrow \mathit{true}] = 1/2^\lambda$

Since these two probabilities are different, this shows that $\mathcal{L}^{\Sigma}_{\text{ots\$-real}} \not\equiv \mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$.
In other words, the scheme does not satisfy this uniform ciphertexts property.

Demonstrate Insecurity with Attacks

Insecure construction

- KeyGen: return $k \leftarrow \{0,1\}^{\lambda}$
- $\mathrm{Enc}(k, m)$: return k & m

Claim This construction does not satisfy one-time secrecy.

Proof

$\mathcal{L}^{\Sigma}_{\text{ots-L}}$
eavesdrop$(m_L, m_R \in \Sigma.\mathcal{M})$:
$\quad k \leftarrow \Sigma.\mathrm{KeyGen}$
$\quad c := \Sigma.\mathrm{Enc}(k, m_L)$
$\quad$ return c

$\equiv$

$\mathcal{L}^{\Sigma}_{\text{ots-R}}$
eavesdrop$(m_L, m_R \in \Sigma.\mathcal{M})$:
$\quad k \leftarrow \Sigma.\mathrm{KeyGen}$
$\quad c := \Sigma.\mathrm{Enc}(k, m_R)$
$\quad$ return c

Demonstrate Insecurity with Attacks

Insecure construction

- KeyGen: return $k \leftarrow \{0,1\}^\lambda$
- $\text{Enc}(k, m)$: return $k \mathbin{\&} m$

Claim This construction does not satisfy one-time secrecy.

Proof

$\mathcal{A}$
$c := \textbf{eavesdrop}(0^\lambda, 1^\lambda)$
return $c =_? 0^\lambda$

$\mathcal{L}^{\Sigma}_{\text{ots-L}}$
$\textbf{eavesdrop}(m_L, m_R)$:
$\quad k \leftarrow \{0,1\}^\lambda$
$\quad c := k \mathbin{\&} m_L$
$\quad$ return c

$\equiv$

$\mathcal{L}^{\Sigma}_{\text{ots-R}}$
$\textbf{eavesdrop}(m_L, m_R)$:
$\quad k \leftarrow \{0,1\}^\lambda$
$\quad c := k \mathbin{\&} m_R$
$\quad$ return c

Demonstrate Insecurity with Attacks

Insecure construction

- KeyGen: return $k \leftarrow \{0,1\}^{\lambda}$
- Enc(k, m): return $k \,\&\, m$

Claim This construction does not satisfy one-time secrecy.

Proof

$\mathcal{A}$
$c :=$ **eavesdrop**$(0^{\lambda}, 1^{\lambda})$ return $c =_? 0^{\lambda}$

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{ots-L}}$
eavesdrop(m_L, m_R): $k \leftarrow \{0,1\}^{\lambda}$ $c := k \,\&\, m_L$ return c

$\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots-L}} \Rightarrow \mathit{true}] = 1$

Demonstrate Insecurity with Attacks

Insecure construction

- KeyGen: return $k \leftarrow \{0,1\}^{\lambda}$
- $\text{Enc}(k, m)$: return $k \,\&\, m$

Claim This construction does not satisfy one-time secrecy.

Proof

$\mathcal{A}$
$c := \textbf{eavesdrop}(0^{\lambda}, 1^{\lambda})$ return $c =_? 0^{\lambda}$

$\diamond$

$\mathcal{L}^{\Sigma}_{\text{ots-R}}$
$\textbf{eavesdrop}(m_L, m_R)$: $k \leftarrow \{0,1\}^{\lambda}$ $c := k \,\&\, m_R$ return c

$\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots-L}} \Rightarrow \mathit{true}] = 1$

$\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots-R}} \Rightarrow \mathit{true}] = 1/2^{\lambda}$

Since these two probabilities are different, this shows that $\mathcal{L}^{\Sigma}_{\text{ots-L}} \not\equiv \mathcal{L}^{\Sigma}_{\text{ots-R}}$. In other words, the scheme does not have one-time secrecy.

How to Prove Security with The Hybrid Technique

- Chaining several components
  - $\mathcal{A} \diamond \mathcal{L}_1 \diamond \mathcal{L}_2$, $\mathcal{L}_1$ can make calls to subroutines in $\mathcal{L}_2$, but not vice-versa.
- We can interpret $\mathcal{A} \diamond \mathcal{L}_1 \diamond \mathcal{L}_2$ as:
  - $(\mathcal{A} \diamond \mathcal{L}_1) \diamond \mathcal{L}_2$: compound calling program linked to $\mathcal{L}_2$.
  - $\mathcal{A} \diamond (\mathcal{L}_1 \diamond \mathcal{L}_2)$: $\mathcal{A}$ linked to compound library.

How to Prove Security with The Hybrid Technique

- **Lemma** if $\mathcal{L}_{left} \equiv \mathcal{L}_{right}$, then for any library $\mathcal{L}^*$, we have $\mathcal{L}^* \diamond \mathcal{L}_{left} \equiv \mathcal{L}^* \diamond \mathcal{L}_{right}$.

Proof

Let $\mathcal{A}$ be an arbitrary calling program. We must show $\mathcal{A} \diamond (\mathcal{L}^* \diamond \mathcal{L}_{left})$ and $\mathcal{A} \diamond (\mathcal{L}^* \diamond \mathcal{L}_{right})$ have identical output distributions.

$$\Pr[\mathcal{A} \diamond (\mathcal{L}^* \diamond \mathcal{L}_{left}) \Rightarrow true] = \Pr[(\mathcal{A} \diamond \mathcal{L}^*) \diamond \mathcal{L}_{left} \Rightarrow true]$$
$$= \Pr[(\mathcal{A} \diamond \mathcal{L}^*) \diamond \mathcal{L}_{right} \Rightarrow true] = \Pr[\mathcal{A} \diamond (\mathcal{L}^* \diamond \mathcal{L}_{right}) \Rightarrow true]$$

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$((k_1, k_2), m)$: $c_1 := k_1 \oplus m$, $c_2 := k_2 \oplus c_1$, return c_2.
  - Dec$((k_1, k_2), c)$: $c_1 := k_2 \oplus c_2$, $m := k_1 \oplus c_1$, 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$(m \in \Sigma.\mathcal{M})$: $\quad (k_1, k_2) \leftarrow \{0,1\}^{2\lambda}$ $\quad c_1 := k_1 \oplus m$ $\quad c_2 := k_2 \oplus c_1$ $\quad$ 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:

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

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

An Example of Hybrid Proof

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

Proof we must show that:

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

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

An Example of Hybrid Proof

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

Proof we must show that:

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

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

An Example of Hybrid Proof

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

Proof we must show that:

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

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

Summary of the Hybrid Technique

- Proving security means showing that two particular libraries, say $\mathcal{L}_{left}$ and $\mathcal{L}_{right}$, are interchangeable.
- Often $\mathcal{L}_{left}$ and $\mathcal{L}_{right}$ are significantly different, The idea is to break up the large "gap" between $\mathcal{L}_{left}$ and $\mathcal{L}_{right}$ 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.

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

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

$\overset{?}{\equiv}$

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

```
ctxt(m):
  c ← {0,1}^λ
  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

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

$\underline{\textsc{ctxt}(m):}$
$\quad \boxed{k \leftarrow \{0,1\}^{\lambda}}$
$\quad c_1 \coloneqq k \oplus m$
$\quad c_2 \coloneqq \boxed{k \oplus c_1}$
$\quad$ return c_2

$\overset{?}{\equiv}$

$\underline{\textsc{ctxt}(m):}$
$\quad c_1 \coloneqq k \oplus m$ // ??
$\quad c_2 \coloneqq \boxed{\textsc{ctxt}'(c_1)}$
$\quad$ return c_2

$\diamond$

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

$\underline{\textsc{ctxt}'(m'):}$
$\quad k \leftarrow \{0,1\}^{\lambda}$
$\quad$ return $k \oplus m'$

(the last two boxes together: $\mathcal{L}_{\text{hyb}}$)

$\mathcal{L}_{\text{otp-real}}$ 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 ← Σ.KeyGen
  c ← Σ.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 ← Σ.KeyGen
  c ← Σ.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

$$\begin{array}{|l|}\hline \textsc{eavesdrop}(m_L, m_R): \\ \hline \quad c := \textsc{ctxt}(m_L) \\ \quad \text{return } c \\ \hline \end{array} \diamond \begin{array}{|l|}\hline \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \\ \hline \textsc{ctxt}(m): \\ \quad c \leftarrow \Sigma.\mathcal{C} \\ \quad \text{return } c \\ \hline \end{array} \quad \Longrightarrow \quad \begin{array}{|l|}\hline \textsc{eavesdrop}(m_L, m_R): \\ \hline \quad c := \textsc{ctxt}(m_R) \\ \quad \text{return } c \\ \hline \end{array} \diamond \begin{array}{|l|}\hline \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \\ \hline \textsc{ctxt}(m): \\ \quad c \leftarrow \Sigma.\mathcal{C} \\ \quad \text{return } c \\ \hline \end{array}$$

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

$\Longrightarrow$

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

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

$\diamond$

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

$\Longrightarrow$

$\mathcal{L}^{\Sigma}_{\text{ots-R}}$
$\textsc{eavesdrop}(m_L, m_R)$:
$k \leftarrow \Sigma.\mathsf{KeyGen}$
$c \leftarrow \Sigma.\mathsf{Enc}(k, m_R)$
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 $k \leftarrow \{0,1\}^\lambda$.

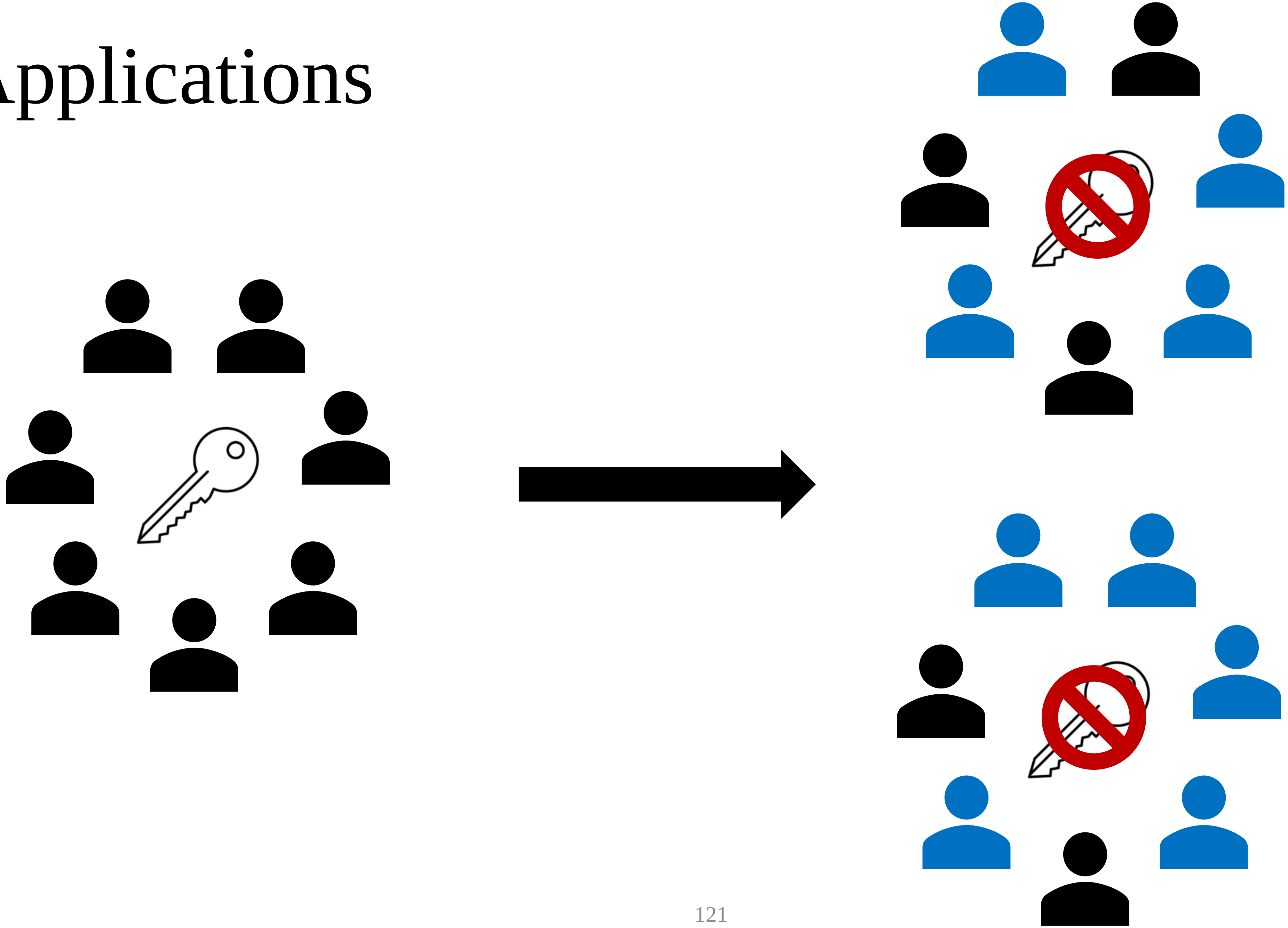
Enc$(k, m \in \{0,1\}^\lambda)$: $c' \coloneqq k \oplus m$, $c \coloneqq c' || 00$, return c.

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

- 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

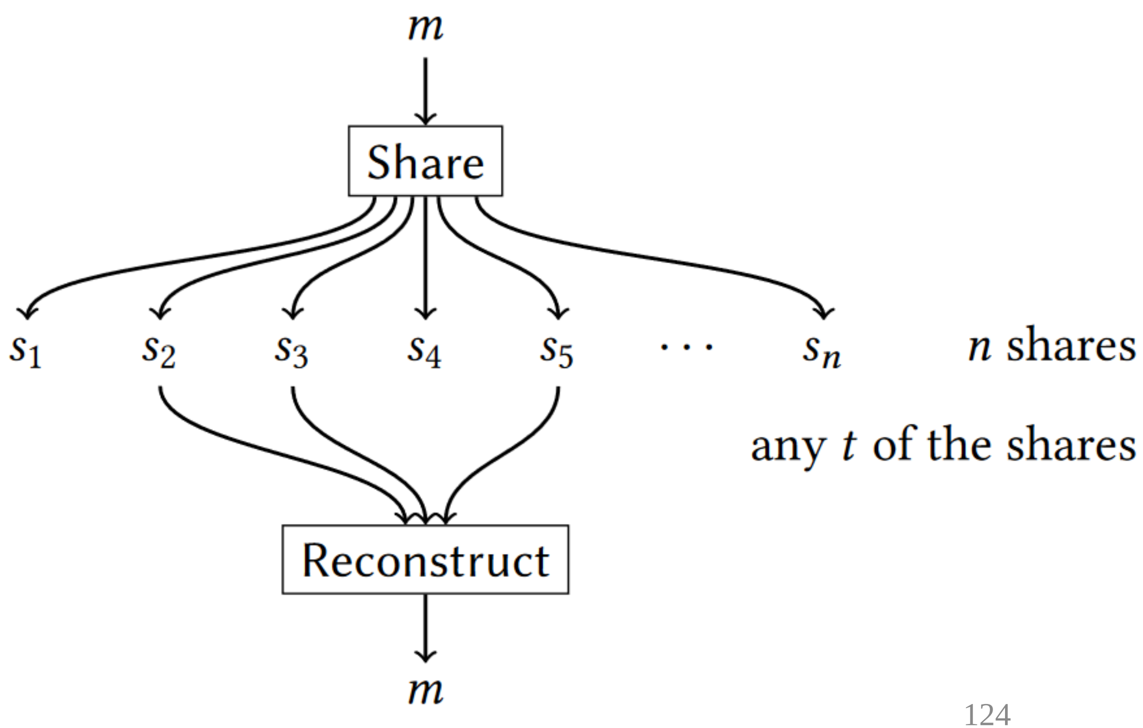
Definition A t-out-of-n threshold secret-sharing scheme (TSSS) consists of the following algorithms:

- 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:

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

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

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

share$(m_L, m_R \in \Sigma.\mathcal{M}, U)$:
    If $\lvert U \rvert \geq \Sigma.t$: return err
    $s \leftarrow \Sigma.\text{Share}(m_R)$
    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 $(s_1, \dots, s_5)$.
- Reconstruct$(s_1, \dots, s_5)$: return $s_1 || \cdots || s_5$.

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

An Construction

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

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

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 $(s_1, \dots, s_5)$.
- Reconstruct$(s_1, \dots, s_5)$: return $s_1 || \cdots || s_5$.
- But this construction is insecure.

$\mathcal{A}$

$s_1 := \textsc{share}(0^{500}, 1^{500}, \{1\})$

return $s_1 \stackrel{?}{=} 0^{100}$

$\diamond$

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

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

  if $|U| \geqslant t$: return 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.

$$\boxed{\begin{array}{l} \mathcal{A} \\ \hline s_1 := \text{SHARE}(\mathtt{0}^{500}, \mathtt{1}^{500}, \{1\}) \\ \text{return } s_1 \stackrel{?}{=} \mathtt{0}^{100} \end{array}} \diamond \boxed{\begin{array}{l} \mathcal{L}_{\text{tsss-R}} \\ \hline \underline{\text{SHARE}(m_L, m_R, U):} \\ \quad \text{if } |U| \geqslant t\text{: return } \mathtt{err} \\ \quad s \leftarrow \text{Share}(m_R) \\ \quad \text{return } \{s_i \mid i \in U\} \end{array}}$$

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

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

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

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

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

if $|U| \geqslant 2$: return err

$s_1 \leftarrow \{0,1\}^\ell$

$s_2 \coloneqq s_1 \oplus m_L$

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

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
  $s_1 \leftarrow \{0,1\}^\ell$
  $s_2 \coloneqq s_1 \oplus m_L$
  return $\{s_i \mid i \in U\}$

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

A Simple 2-out-of-2 Scheme

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

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

$\Rightarrow$

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

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

$$
\begin{array}{l}
\underline{\textsc{share}(m_L, m_R, U):} \\
\quad \text{if } |U| \geqslant 2\text{: return } \texttt{err} \\
\quad \text{if } U = \{1\}: \\
\qquad s_1 \leftarrow \{\texttt{0},\texttt{1}\}^\ell \\
\qquad s_2 := s_1 \oplus \boxed{m_R} \\
\qquad \text{return } \{s_1\} \\
\quad \text{elsif } U = \{2\}: \\
\qquad s_1 \leftarrow \{\texttt{0},\texttt{1}\}^\ell \\
\qquad s_2 := s_1 \oplus m_L \\
\qquad \text{return } \{s_2\} \\
\quad \text{else return } \emptyset
\end{array}
\Longrightarrow
\begin{array}{l}
\underline{\textsc{share}(m_L, m_R, U):} \\
\quad \text{if } |U| \geqslant 2\text{: return } \texttt{err} \\
\quad \text{if } U = \{1\}: \\
\qquad s_1 \leftarrow \{\texttt{0},\texttt{1}\}^\ell \\
\qquad s_2 := s_1 \oplus m_R \\
\qquad \text{return } \{s_1\} \\
\quad \text{elsif } U = \{2\}: \\
\qquad \boxed{s_2 \leftarrow \textsc{eavesdrop}(m_L, m_R)} \\
\qquad \text{return } \{s_2\} \\
\quad \text{else return } \emptyset
\end{array}
\diamond
\boxed{
\begin{array}{l}
\mathcal{L}^{\text{OTP}}_{\text{ots-L}} \\
\underline{\textsc{eavesdrop}(m_L, m_R):} \\
\quad k \leftarrow \{\texttt{0},\texttt{1}\}^\ell \\
\quad c := k \oplus m_L \\
\quad \text{return } c
\end{array}
}
$$

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.

Because s_2 is never used in this branch.

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 $\emptyset$

$\diamond$

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

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

$\Rightarrow$

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

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

$\underline{\text{SHARE}(m_L, m_R, U):}$
  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 $U = \{2\}$:
    $s_2 \leftarrow \text{EAVESDROP}(m_L, m_R)$
    return $\{s_2\}$
  else return $\emptyset$

$\diamond$

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

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

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

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

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

Rewrite the Construction

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

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 $\{(x_1, y_1), \dots, (x_{d+1}, y_{d+1})\} \subseteq \mathbb{R}^2$ 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

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

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

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

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 $\{(x_1, y_1), \dots, (x_{d+1}, y_{d+1})\} \subseteq \mathbb{R}^2$ 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 $\{(x_1, y_1), \dots, (x_{d+1}, y_{d+1})\} \subseteq \mathbb{R}^2$ 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 $\mathcal{P} = \{(x_1, y_1), \dots, (x_k, y_k)\} \subseteq \left(\mathbb{Z}_p\right)^2$ 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

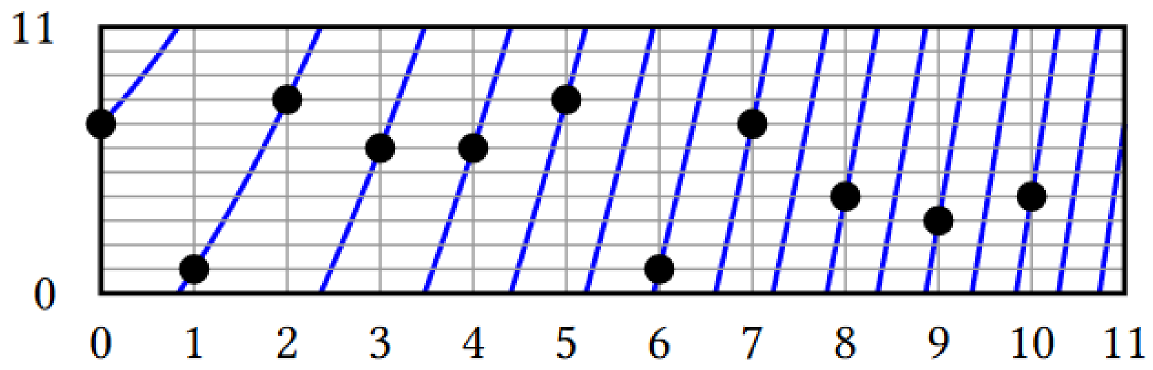
Inductive case ($k + 1$ case holds): $k \leq d$ points in $\mathcal{P}$. Let $x^* \in \mathbb{Z}_p \neq x_i$ for all i. Every polynomial must give some value when evaluated at x^*.

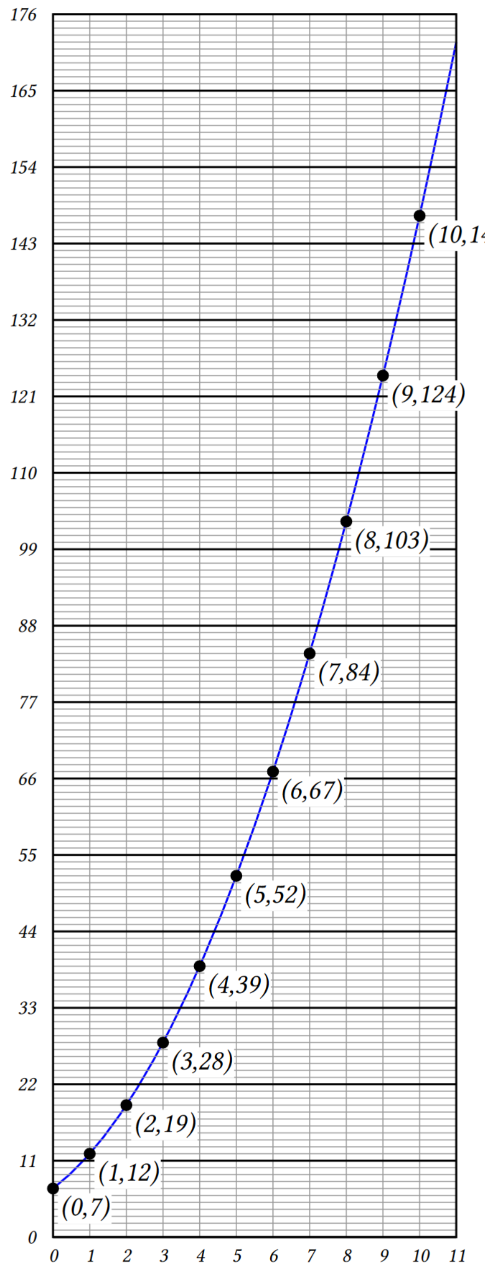
Compute [# of degree $-$ d polynomials pass through points in $\mathcal{P}$] $=$ $\sum_{y^* \in \mathbb{Z}_p}$[# of degree $-$ d polynomials pass through points in $\mathcal{P} \cup \{(x^*, y^*)\}$] $=$ $\sum_{y^* \in \mathbb{Z}_p} p^{d+1-(k+1)} = p \cdot p^{d+1-k-1} = p^{d+1-k}$

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 $f(0)$.