数据隐私方法伦理和实践

Methodology, Ethics and Practice of Data Privacy

安全基础

Basics of Security

张兰
中国科学技术大学 计算机学院
2020春季

“

Human ingenuity cannot concoct a cypher which human ingenuity cannot resolve?

“

The core concept that allows us to actually prove things about security is the security definition.

— Mike Rosulek

1. Abstract Security Definition

Abstract View

- **Convenience**: "any encryption scheme, when combined with this other thing **in this specific way**, results in a system with **security property X**, as long as the encryption scheme **satisfies property Y**.

- **Modularity**: you might be able to swap encryption scheme A for some encryption scheme B, as long as scheme B satisfies all of the security requirements.

Provable Security

- A good abstraction for encryption
- How to write a security definition
- Formalisms for security definitions
- How to prove security with the hybrid technique
- How to demonstrate insecurity with attacks

[1]http://web.engr.oregonstate.edu/~rosulekm/crypto/

Syntax & Correctness

» Definition: Encryption syntax

*A **symmetric-key encryption (SKE) scheme** consists of the following algorithms:*

- KeyGen: *a randomized algorithm that outputs a **key** $k \in \mathcal{K}$.*
- Enc: *a (possibly randomized) algorithm that takes a key $k \in \mathcal{K}$ and **plaintext** $m \in \mathcal{M}$ as input, and outputs a **ciphertext** $c \in \mathcal{C}$.*
- Dec: *a deterministic algorithm that takes a key $k \in \mathcal{K}$ and ciphertext $c \in \mathcal{C}$ as input, and outputs a plaintext $m \in \mathcal{M}$.*

*We call $\mathcal{K}$ the **key space**, $\mathcal{M}$ the **message space**, and $\mathcal{C}$ the **ciphertext space** of the scheme. Sometimes we refer to the entire scheme (all algorithms) by a single variable Σ. When we do so, we write Σ.KeyGen, Σ.Enc, Σ.Dec, $\Sigma.\mathcal{K}$, $\Sigma.\mathcal{M}$, and $\Sigma.\mathcal{C}$ to refer to its components.*

Syntax & Correctness

Definition: Correctness

An encryption scheme Σ *satisfies* ***correctness*** *if for all* $k \in \Sigma.\mathcal{K}$ *and all* $m \in \Sigma.\mathcal{M}$,

$$\Pr\left[\Sigma.\mathrm{Dec}(k, \Sigma.\mathrm{Enc}(k, m)) = m\right] = 1.$$

- Enc is allowed to be a randomized algorithm.
- Decrypting a ciphertext, using the same key that was used for encryption, **always** results in the original plaintext.
- Not involve any adversarial behavior, so correct but not necessarily secure.

How to Write a Security Definition

- A specific property that one-time pad satisfies:
- **Attempt 1**
  - For $\boldsymbol{m} \in \{\mathbf{0}, \mathbf{1}\}^{\lambda}$, the output of the following subroutine is uniformly distributed over $\{\mathbf{0}, \mathbf{1}\}^{\lambda}$

$$
\boxed{
\begin{array}{l}
\underline{\textsc{eavesdrop}(m \in \{\texttt{0}, \texttt{1}\}^{\lambda}):} \\
\quad k \leftarrow \{\texttt{0}, \texttt{1}\}^{\lambda} \\
\quad c := k \oplus m \\
\quad \text{return } c
\end{array}
}
.
$$

How to Write a Security Definition

- **We need a general-purpose definition.**

$$\begin{array}{l} \underline{\textsc{eavesdrop}(m \in \Sigma.\mathcal{M}):} \\ \quad k \leftarrow \Sigma.\text{KeyGen} \\ \quad c \leftarrow \Sigma.\text{Enc}(k, m) \\ \quad \text{return } c \end{array}$$

- The output of this subroutine is uniformly distributed over what set?
  - plaintext space, key space, or ciphertext space?

How to Write a Security Definition

- **Attempt 2**
  - Σ is "secure" if, for all $m \epsilon \Sigma.M$, the output of the following subroutine is uniformly distributed over $\Sigma.C$:

$$
\boxed{
\begin{array}{l}
\underline{\textsc{eavesdrop}(m \in \Sigma.\mathcal{M}):} \\
\quad k \leftarrow \Sigma.\mathsf{KeyGen} \\
\quad c \leftarrow \Sigma.\mathsf{Enc}(k, m) \\
\quad \text{return } c
\end{array}
}
$$

How to Write a Security Definition

- **Adversaries as Distinguishers**
  - **A game:** an adversary can send me any input, and I will either run the left implementation or the right implementation. The adversary need to guess which implementation I'm using.
- **Attempt 3**
  - Σ is "secure" if the following two implementations of an eavesdrop subroutine have the same input-output behavior.

$$
\boxed{\begin{array}{l}
\underline{\textsc{eavesdrop}(m \in \Sigma.\mathcal{M}):} \\
\quad k \leftarrow \Sigma.\text{KeyGen} \\
\quad c \leftarrow \Sigma.\text{Enc}(k, m) \\
\quad \text{return } c
\end{array}}
\;;\qquad
\boxed{\begin{array}{l}
\underline{\textsc{eavesdrop}(m \in \Sigma.\mathcal{M}):} \\
\quad c \leftarrow \Sigma.C \\
\quad \text{return } c
\end{array}}
$$

How to Write a Security Definition

- **Define "identical input-output behavior"**
- **Attempt 4**
  - Σ is "secure" if, for all calling programs A, connecting A with either the left or right version of eavesdrop does not change the output probability of A.

No calling program can tell them apart!

$$\begin{array}{|l|}\hline \underline{\textsc{eavesdrop}(m \in \Sigma.\mathcal{M}):} \\ \quad k \leftarrow \Sigma.\text{KeyGen} \\ \quad c \leftarrow \Sigma.\text{Enc}(k, m) \\ \quad \text{return } c \\ \hline \end{array} \; ; \qquad \begin{array}{|l|}\hline \underline{\textsc{eavesdrop}(m \in \Sigma.\mathcal{M}):} \\ \quad c \leftarrow \Sigma.\mathcal{C} \\ \quad \text{return } c \\ \hline \end{array}$$

- **Pros/cons of this security definition?**

How to Write a Security Definition

- Let us consider some (possibly strange) encryption schemes and seeing whether they satisfy the definition
- **Construction: Doubled OTP**

$$\mathcal{K} = \{0,1\}^\lambda \quad \mathcal{M} = \{0,1\}^\lambda \quad \mathcal{C} = \{0,1\}^{2\lambda}$$

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

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

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

- Intuitively, this new scheme is just as **secure** as original one-time pad.
- However, this doubled OTP **does not satisfy** the security definition attempt #4.

How to Write a Security Definition

- Attempt #4 requires that the following two subroutines implementations have the same input-output behavior:

$$\boxed{\begin{array}{l}\underline{\textsc{eavesdrop}(m):}\\ \quad k \leftarrow \{\texttt{0},\texttt{1}\}^\lambda \\ \quad c' := k \oplus m \\ \quad \text{return } c' \| c'\end{array}}\ ; \qquad \boxed{\begin{array}{l}\underline{\textsc{eavesdrop}(m):}\\ \quad c \leftarrow \{\texttt{0},\texttt{1}\}^{2\lambda} \\ \quad \text{return } c\end{array}}\ .$$

$$\boxed{\begin{array}{l}\mathcal{A}\text{:}\\ c := \textsc{eavesdrop}(\texttt{0}^\lambda) \\ L := \text{first half of } c \\ R := \text{second half of } c \\ \text{return } L \stackrel{?}{=} R\end{array}}$$

- **Left:** $\Pr[\mathcal{A} \text{ outputs } \texttt{true}] = 1$

- **Right**: $\Pr[\mathcal{A} \text{ outputs } \texttt{true}] = 1/2^\lambda < 1$

How to Write a Security Definition

- **Attempt #4 was slightly too strong.**
  - It demands that Eavesdrop(m) was uniform.
  - The most important factor is Eavesdrop(m) and Eavesdrop(m') are the same distribution for all m and m'.

$$\begin{array}{|l|}\hline \textsc{Eavesdrop}(m \in \Sigma.\mathcal{M}): \\ \hline \quad k \leftarrow \Sigma.\text{KeyGen} \\ \quad c \leftarrow \Sigma.\text{Enc}(k, m) \\ \quad \text{return } c \\ \hline \end{array} ; \quad \begin{array}{|l|}\hline \textsc{Eavesdrop}(m \in \Sigma.\mathcal{M}): \\ \hline \quad c \leftarrow \Sigma.\mathcal{C} \\ \quad \text{return } c \\ \hline \end{array}$$

How to Write a Security Definition

Attempt 5 (chosen-plaintext attack):

- Σ is "secure" if, for **all calling programs A (all strategies for choosing m_L and m_R**), connecting A with either the left or right version of eavesdrop does not change the output probability of A.

$$\boxed{\begin{array}{l} \textsc{eavesdrop}(m_L, m_R \in \Sigma.\mathcal{M}): \\ \hline \quad k \leftarrow \Sigma.\text{KeyGen} \\ \quad c \leftarrow \Sigma.\text{Enc}(k, m_L) \\ \quad \text{return } c \end{array}} \; ; \qquad \boxed{\begin{array}{l} \textsc{eavesdrop}(m_L, m_R \in \Sigma.\mathcal{M}): \\ \hline \quad k \leftarrow \Sigma.\text{KeyGen} \\ \quad c \leftarrow \Sigma.\text{Enc}(k, m_R) \\ \quad \text{return } c \end{array}} \; .$$

How to Write a Security Definition

- **Attempt 5 (chosen-plaintext attack):**
  - The calling program chooses two plaintext m_L, m_R and only one is encrypted.
  - Seeing a ciphertext leaks no information about the choice of plaintext,
  - even if you already knew some partial information about the choice of plaintext,
  - even if you knew that it was one of only two options,
  - even if you got to choose those two options!

2. Provable Security Fundamentals

Libraries & Interfaces

- **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.
- A program $\mathcal{A}$ includes calls to subroutines in the interface of $\mathcal{L}$.
- $\mathcal{A} \diamond \mathcal{L} \Rightarrow z$: the result of linking $\mathcal{A}$ to $\mathcal{L}$
- The only thing a calling program can do with a library is to call its subroutines (on any arguments of its choice) and receive the output of subroutines.

Libraries & Interfaces

» **Example:** here are two libraries $\mathcal{L}_1, \mathcal{L}_2$, and calling program $\mathcal{A}$.

$\mathcal{L}_1$

$\textsc{eavesdrop}(m)$:

$k \leftarrow \{\texttt{0},\texttt{1}\}^\lambda$
$c' := k \oplus m$
return $c' \| c'$

$\mathcal{L}_2$

$\textsc{eavesdrop}(m)$:

$c \leftarrow \{\texttt{0},\texttt{1}\}^{2\lambda}$
return c

$\mathcal{A}$:

$c := \textsc{eavesdrop}(\texttt{0}^\lambda)$
$L :=$ first half of c
$R :=$ second half of c
return $L \stackrel{?}{=} R$

we argued that: $\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow \mathtt{true}] = 1,$

$$\Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow \mathtt{true}] = 1/2^\lambda$$

Libraries & Interfaces

- **Example:** here is a simple library that picks a string s uniformly and allows the calling program to guess **s** :

$$\mathcal{L}$$

$s \leftarrow \{0,1\}^\lambda$

$\underline{\text{RESET}():}$

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

$\underline{\text{GUESS}(x \in \{0,1\}^\lambda):}$

$\quad \text{return } x \stackrel{?}{=} s$

- **Our convention** is that code outside of a subroutine (like the first line here) is run once at initialization time.

Interchangeability

- **Whether two libraries have the same input-output behavior.**

- **Definition: Interchangeable.** Let $\mathcal{L}_1, \mathcal{L}_2$ be two libraries with a common interface. We say that $\mathcal{L}_1, \mathcal{L}_2$ are interchangeable, and write $\mathcal{L}_1 \equiv \mathcal{L}_2$, if for all programs $\mathcal{A}$ that output a single bit, $\Pr[\mathcal{A} \diamond \mathcal{L}_1 \Rightarrow 1] = \Pr[\mathcal{A} \diamond \mathcal{L}_2 \Rightarrow 1]$.

- **Note** the definition says that the two libraries have the same effect on all calling program.
- **Note** a calling program $\mathcal{A}$'s only goal is to distinguish between these particular libraries.

Interchangeability

- **Example:** The following two libraries are interchangeable.
  - In the uniform distribution on $\{0,1\}^{n+m}$, each of the individual bits is distributed independently of the others.

$$\begin{array}{|l|}\hline \underline{\text{SAMPLE}():} \\ \quad x \leftarrow \{0,1\}^n \\ \quad y \leftarrow \{0,1\}^m \\ \quad \text{return } x \| y \\ \hline \end{array} \qquad \begin{array}{|l|}\hline \underline{\text{SAMPLE}():} \\ \quad z \leftarrow \{0,1\}^{n+m} \\ \quad \text{return } z \\ \hline \end{array}$$

Interchangeability

» **Example:** the following two libraries are interchangeable. The library on the left samples s “eagerly” — as soon as it can. The library on the right samples s “lazily” — only at the last possible moment.

```
s ← {0,1}^n

GET():
  return s
```

```
GET():
  if s not defined:
    s ← {0,1}^n
  return s
```

Security Definitions, Using New Terminology

- Specifically about one-time pad can be written in terms of interchangeable libraries:
- **Claim: OTP rule.** The following two libraries are interchangeable:

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

$\textsc{eavesdrop}(m \in \{0,1\}^\lambda)$:

$k \leftarrow \{0,1\}^\lambda$

return $k \oplus m$

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

$\textsc{eavesdrop}(m \in \{0,1\}^\lambda)$:

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

return c

Security Definitions, Using New Terminology

- **Definition:** Let Σ be an encryption scheme. We say that Σ has **one-time uniform ciphertexts** if $\mathcal{L}^{\Sigma}_{\text{ots\$-real}} \equiv \mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$ where:

$$\begin{array}{l} \hline \mathcal{L}^{\Sigma}_{\text{ots\$-real}} \\ \hline \underline{\textsc{ctxt}(m \in \Sigma.\mathcal{M}):} \\ \quad k \leftarrow \Sigma.\text{KeyGen} \\ \quad c \leftarrow \Sigma.\text{Enc}(k, m) \\ \quad \text{return } c \\ \hline \end{array} \qquad \begin{array}{l} \hline \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \\ \hline \underline{\textsc{ctxt}(m \in \Sigma.\mathcal{M}):} \\ \quad c \leftarrow \Sigma.\mathcal{C} \\ \quad \text{return } c \\ \hline \end{array}$$

- we will use the "\$" symbol to denote something random (or pseudorandom).

Security Definitions, Using New Terminology

» **Definition: One-time secrecy.** Let Σ be an encryption scheme. We say that Σ has **one-time secrecy** if $\mathcal{L}^{\Sigma}_{\text{ots-L}} \equiv \mathcal{L}^{\Sigma}_{\text{ots-R}}$, where:

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

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

Security Definitions

- One-time uniform ciphertexts, which states that ciphertexts should be uniformly distributed in $\Sigma.\mathrm{C}$.

- One-time secrecy, which states that all m result in the same ciphertext distribution (but that distribution need not be uniform).

Security Definitions

- The libraries capture the attacker's view of things.
- **Don't** interpret one-time secrecy to mean "I'm not allowed to choose what to encrypt, I have to ask the adversary to choose for me."
- **Do** think "If I encrypt only one plaintext per key, then I am safe to encrypt things even if the attacker sees the resulting ciphertext and even if she has some influence or partial information on what I'm encrypting.

Kerckhoffs' Principle

- **Kerckhoffs' Principle** says to assume that the adversary has full knowledge of the algorithms, and only lacks knowledge about the choice of keys.

- **Kerckhoffs' Principle, in our terminology:**
  - Assume that the distinguisher knows every fact in the universe, except for:
  - **1.** which of the two possible libraries it is linked to,
  - **2.** the outcomes of random choices made by the library.

3. How to Prove Security

Chaining Several Components

» $\mathcal{A} \diamond \mathcal{L}_1 \diamond \mathcal{L}_2$

- $(\mathcal{A} \diamond \mathcal{L}_1) \diamond \mathcal{L}_2$: a **compound calling program** linked to $\mathcal{L}_2$. After all, $\mathcal{A} \diamond \mathcal{L}_1$ is a program that makes calls to the interface of $\mathcal{L}_2$.
- or: $\mathcal{A} \diamond (\mathcal{L}_1 \diamond \mathcal{L}_2)$: $\mathcal{A}$ linked to a **compound library**. After all, $\mathcal{A}$ is a program that makes calls to the interface of $(\mathcal{L}_1 \diamond \mathcal{L}_2)$.

» Lemma: Chaining

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

$$\begin{aligned}
\Pr[\mathcal{A} \diamond (\mathcal{L}^* \diamond \mathcal{L}_{\text{left}}) \Rightarrow 1] &= \Pr[(\mathcal{A} \diamond \mathcal{L}^*) \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] && \text{(change of perspective)} \\
&= \Pr[(\mathcal{A} \diamond \mathcal{L}^*) \diamond \mathcal{L}_{\text{right}} \Rightarrow 1] && \text{(since } \mathcal{L}_{\text{left}} \equiv \mathcal{L}_{\text{right}}\text{)} \\
&= \Pr[\mathcal{A} \diamond (\mathcal{L}^* \diamond \mathcal{L}_{\text{right}}) \Rightarrow 1]. && \text{(change of perspective)}
\end{aligned}$$

One-Time Secrecy of One-Time Pad

- We have already proved that OTP satisfied the one-time uniform ciphertexts definition.

- Theroem: Let Σ be an encryption scheme. If Σ has one-time uniform ciphertext, then Σ also has one-time secrecy.

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

- Interpret: if all plaintexts m result in a uniform distribution of ciphertexts, then all m result in the same distribution of ciphertexts.

How to Prove Security with the Hybrid Technique

- **Proof:** to prove $\mathcal{L}^{\Sigma}_{\text{ots-L}} \equiv \mathcal{L}^{\Sigma}_{\text{ots-R}}$

We show that

$$\mathcal{L}^{\Sigma}_{\text{ots-L}} \equiv \mathcal{L}_{\text{hyb-1}} \equiv \mathcal{L}_{\text{hyb-2}} \equiv \mathcal{L}_{\text{hyb-3}} \equiv \mathcal{L}_{\text{hyb-4}} \equiv \mathcal{L}^{\Sigma}_{\text{ots-R}}$$

We are allowed to use the fact that

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

This proof technique is called the hybrid technique.

How to Prove Security with the Hybrid Technique

- **Proof:**
  - 1. start point

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

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

How to Prove Security with the Hybrid Technique

- **Proof:**
  - 2. compound library with $\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$

$$\mathcal{L}_{\text{hyb-1}}: \quad \boxed{\begin{array}{l} \underline{\textsc{eavesdrop}(m_L, m_R \in \Sigma.\mathcal{M}):} \\ \quad c := \textsc{ctxt}(m_L) \\ \quad \text{return } c \end{array}} \diamond \boxed{\begin{array}{l} \qquad \mathcal{L}^{\Sigma}_{\text{ots\$-real}} \\ \underline{\textsc{ctxt}(m \in \Sigma.\mathcal{M}):} \\ \quad k \leftarrow \Sigma.\text{KeyGen} \\ \quad c \leftarrow \Sigma.\text{Enc}(k, m) \\ \quad \text{return } c \end{array}}$$

How to Prove Security with the Hybrid Technique

- **Proof:**
  - 3. replace $\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$ with $\mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$
  - Chaining lemma

$\mathcal{L}_{\text{hyb-2}}$:

$\textsc{eavesdrop}(m_L, m_R \in \Sigma.\mathcal{M})$:
$c := \textsc{ctxt}(m_L)$
return c

$\diamond$

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

How to Prove Security with the Hybrid Technique

- **Proof:**
  - **4.** The argument to ctxt has been changed from m_L to m_R. This has no effect on the library's behavior since ctxt does not actually use its argument in these hybrids!

$$\mathcal{L}_{\text{hyb-3}}: \quad \boxed{\begin{array}{l} \underline{\text{EAVESDROP}(m_L, m_R \in \Sigma.\mathcal{M}):} \\ \quad c := \text{CTXT}(\,m_R\,) \\ \quad \text{return } c \end{array}} \diamond \boxed{\begin{array}{c} \mathcal{L}^{\Sigma}_{\text{ots\$-rand}} \\ \hline \begin{array}{l} \underline{\text{CTXT}(m \in \Sigma.\mathcal{M}):} \\ \quad c \leftarrow \Sigma.\mathcal{C} \\ \quad \text{return } c \end{array} \end{array}}$$

How to Prove Security with the Hybrid Technique

» **Proof:**

- **5. Chaining lemma**

$$\mathcal{L}_{\text{hyb-4}}: \quad \boxed{\begin{array}{l} \underline{\text{EAVESDROP}(m_L, m_R \in \Sigma.\mathcal{M}):} \\ \quad c := \text{CTXT}(m_R) \\ \quad \text{return } c \end{array}} \diamond \boxed{\begin{array}{c} \mathcal{L}^{\Sigma}_{\text{ots\$-real}} \\ \hline \begin{array}{l} \underline{\text{CTXT}(m \in \Sigma.\mathcal{M}):} \\ \quad k \leftarrow \Sigma.\text{KeyGen} \\ \quad c \leftarrow \Sigma.\text{Enc}(k, m) \\ \quad \text{return } c \end{array} \end{array}}$$

How to Prove Security with the Hybrid Technique

- **Proof:**
  - **6. result**

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

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

Summary of the Hybrid Technique

- Proving security means showing that two particular libraries, say L_{left} and L_{right} are interchangeable.
- Often L_{left} and L_{right} are significantly different, making them hard to compare directly. To make the comparison more manageable, we can show a sequence of (interchangeable) hybrid libraries. The idea is to break up the large "gap" into smaller ones that are easier to justify.
- With each modification you should justify why it doesn't affect the calling program.

Does this scheme have one-time secrecy?

$$\mathcal{K} = \left\{ \begin{array}{c} \textit{permutations} \\ \textit{of}\ \{1, \ldots, \lambda\} \end{array} \right\}$$

$$\mathcal{M} = \{\texttt{0}, \texttt{1}\}^\lambda$$

$$\mathcal{C} = \{\texttt{0}, \texttt{1}\}^\lambda$$

<u>KeyGen:</u>

$k \leftarrow \mathcal{K}$

return k

<u>$\text{Enc}(k, m)$:</u>

for $i := 1$ to λ:

$c_{k(i)} := m_i$

return $c_1 \cdots c_\lambda$

<u>$\text{Dec}(k, c)$:</u>

for $i := 1$ to λ:

$m_i := c_{k(i)}$

return $m_1 \cdots m_\lambda$

How to Demonstrate Insecurity with Attacks

How to Demonstrate Insecurity with Attacks

- To show that a scheme is insecure, we just have to show that the two relevant libraries are **not interchangeable**.

- Attack: we have to **find just one** calling program that behaves differently in the presence of the two libraries!

How to Demonstrate Insecurity with Attacks

- **Construction:**

$$\mathcal{K} = \left\{ \begin{array}{c} \textit{permutations} \\ \textit{of } \{1, \ldots, \lambda\} \end{array} \right\}$$

$$\mathcal{M} = \{\texttt{0}, \texttt{1}\}^\lambda$$

$$\mathcal{C} = \{\texttt{0}, \texttt{1}\}^\lambda$$

$\underline{\text{KeyGen:}}$
$k \leftarrow \mathcal{K}$
return k

$\underline{\text{Enc}(k, m)\text{:}}$
for $i := 1$ to λ:
$c_{k(i)} := m_i$
return $c_1 \cdots c_\lambda$

$\underline{\text{Dec}(k, c)\text{:}}$
for $i := 1$ to λ:
$m_i := c_{k(i)}$
return $m_1 \cdots m_\lambda$

- This scheme encrypts a plaintext by simply rearranging its bits according to the secret permutation k.
- **It** does **not** have one-time secrecy.

How to Demonstrate Insecurity with Attacks

- **Proof :**
  - To construct a program $\mathcal{A}$ that $\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{ost-L} \Rightarrow 1] = \Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{ost-R} \Rightarrow 1]$ are different.

How to Demonstrate Insecurity with Attacks

- **Proof :**
  - To construct a program $\mathcal{A}$ that $\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{ost-L} \Rightarrow 1] = \Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{ost-R} \Rightarrow 1]$ are different.
  - One obervation: the ciphertext preserves (leaks) the number of 0s and 1s in the plaintext.
  - We must specify how m_L and m_R are chosen, e.g., with different numbers of 0s and 1s.

How to Demonstrate Insecurity with Attacks

Proof :

- We define the following distinguisher:

$$\begin{array}{|l|}\hline \hfill \mathcal{A} \hfill \\ \hline c \leftarrow \textsc{eavesdrop}(\texttt{0}^\lambda, \texttt{1}^\lambda) \\ \text{return } c \stackrel{?}{=} \texttt{0}^\lambda \\ \hline \end{array} \diamond \begin{array}{|l|}\hline \hfill \mathcal{L}^{\Sigma}_{\text{ots-L}} \hfill \\ \hline \underline{\textsc{eavesdrop}(m_L, m_R):} \\ \quad k \leftarrow \{\text{permutations of } \{1, \ldots, \lambda\}\} \\ \quad \text{for } i := 1 \text{ to } \lambda: \\ \qquad c_{k(i)} := (\,m_L\,)_i \\ \quad \text{return } c_1 \cdots c_\lambda \\ \hline \end{array}.$$

We can see that m_L takes on the value $\texttt{0}^\lambda$, so each bit of m_L is $\texttt{0}$, and each bit of c is $\texttt{0}$. Hence, the final output of $\mathcal{A}$ is always 1 (true):

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

How to Demonstrate Insecurity with Attacks

Proof (Continued):

$$\boxed{\begin{array}{l} \mathcal{A} \\ \hline c \leftarrow \textsc{eavesdrop}(\mathtt{0}^\lambda, \mathtt{1}^\lambda) \\ \text{return } c \stackrel{?}{=} \mathtt{0}^\lambda \end{array}} \diamond \boxed{\begin{array}{l} \mathcal{L}^{\Sigma}_{\text{ots-R}} \\ \hline \underline{\textsc{eavesdrop}(m_L, m_R):} \\ \quad k \leftarrow \{\text{permutations of } \{1, \ldots, \lambda\}\} \\ \quad \text{for } i := 1 \text{ to } \lambda: \\ \qquad c_{k(i)} := (\, m_R \,)_i \\ \quad \text{return } c_1 \cdots c_\lambda \end{array}}\,.$$

We can see that each bit of m_R, and hence each bit of c, is $\mathtt{1}$. So $\mathcal{A}$ will always output 0 (false), giving:

$$\Pr[\mathcal{A} \diamond \mathcal{L}^{\Sigma}_{\text{ots-R}} \Rightarrow 1] = 0.$$

The two probabilities are different, demonstrating that $\mathcal{A}$ behaves differently (in fact, as differently as possible) when linked to the two libraries. We conclude that Construction 2.11 does **not** satisfy the definition of one-time secrecy. ■

“

This is no security, for a practical key should not be too long. But this does not consider how easy or difficult it is for the enemy to make the computation determining the key. If this computation, although possible in principle, were suffiently long at best then the process could still be secure in a practical sense.

--Nash

Computational Security

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

“Computationally Infeasible” Attack

- Schemes like one-time pad cannot be broken, even by an adversary that performs a bruteforce attack.
- However, all future schemes that we will see can indeed be broken by such an attack.
- Nash points out that, for a scheme with λ-bit keys is easily made impractical for the enemy by simply choosing λ large enough.
- λ is the security parameter, which makes the difficulty of a bruteforce attack grow exponentially fast.

“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*

“Computationally Infeasible” Attack

In 2017, the first collision in the SHA-1 hash function was found. The attack involved evaluating the SHA-1 function **2^{63} times** on a cluster of GPUs.

The monetary cost of the attack:
Had the researchers performed their attack on Amazon’s Web Services platform, it would have cost $560,000 at normal pricing.

“Computationally Infeasible” Attack

But, what is the line between “feasible” attacks and “infeasible” ones.

Nash: how does the cost of a computation scale as the security parameter λ goes to infinity?

Asymptotic Running Time
This is at best exponential and at worst probably a relatively small power of λ, $\mathrm{a} \cdot \lambda^2$ or $a \cdot \lambda^3$, as in substitution ciphers.

“Computationally Infeasible” Attack

- 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.
- **Closure property**: repeating a polynomial-time process a polynomial number of times results in a polynomial-time process overall.
- Our goal will be to ensure that no polynomial-time attack can successfully break security.
- We will not worry about attacks like brute-force that require exponential time.

"Computationally Infeasible" Attack

» Potential Pitfall: Numerical Algorithms

- Representing the number N on a computer requires only $\log_2 N$ bits. $\log_2 N$, rather than N, is our security parameter.

» some numerical operations

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

"efficient" means polynomial-time.

Those operations do have known polynomial-time algorithms on quantum computers

“

Is it enough to consider the running time of an attack?

Even a simple guess still has a nonzero chance of breaking security!

“Negligible” Success Probability

- We don’t want to worry about attacks that are as expensive as a brute-force attack.
- We don’t want to worry about attacks whose success probability is as low as a blind-guess attack.

probability	*equivalent*
2^{-10}	*full house in 5-card poker*
2^{-20}	*royal flush in 5-card poker*
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* $\longrightarrow$

- Where to draw the line between “reasonable” and “unreasonable” success probability for an attack?

"Negligible" Success Probability

- Consider how fast a success probability approaches zero as the security parameter grows.
- In a scheme with λ-bit keys, a blind-guessing attack succeeds with probability $\mathrm{f}(\lambda) = 1/2^{\lambda}$, an adversary who makes λ^c guesses, we have:

$$\lim_{\lambda\to\infty} \frac{\lambda^c}{2^\lambda} = 0$$

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

It is supposed to hold against all polynomial-time adversaries.

“Negligible” Success Probability

» Definition:

If $f, g : N \rightarrow \mathrm{R}$ are two functions, we write $\boldsymbol{f} \approx \boldsymbol{g}$ to mean that $|f(\lambda) = g(\lambda)|$ is negligible function.

$\Pr[X] \approx 0 \Leftrightarrow$ “event X almost never happens”
$\Pr[Y] \approx 1 \Leftrightarrow$ “event Y almost always happens”
$\Pr[\mathrm{A}] \approx \Pr[B] \Leftrightarrow$ “events A and B happen with essentially the same probability”.

Transitive applied a polynomial number of times:

If $\Pr[\mathrm{X}] \approx \Pr[Y], \Pr[\mathrm{Y}] \approx \Pr[Z]$, then $\Pr[\mathrm{X}] \approx \Pr[Z]$

“

Interchangeable libraries require that two libraries have exactly the same effect on every calling program.

In practice, we only consider polynomial-time calling programs; we don’t require the libraries to have exactly the same effect on the calling program, only that the difference in effects is negligible.

Indistinguishability

- Indistinguishability:

Let L_{left} and L_{right} be two libraries with a common interface. We say that L_{left} and L_{right} are **indistinguishable**, and write $L_{left} \approxeq L_{right}$, if for all polynomial-time programs A that output a single bit,

$$\Pr[A \diamond L_{left} \Rightarrow 1] \approx \Pr[A \diamond L_{right} \Rightarrow 1].$$

- $\Pr[A \diamond L_{left} \Rightarrow 1] - \Pr[A \diamond L_{right} \Rightarrow 1]$ is the **advantage** or bias of A in distinguishing L_{left} and L_{right}.

Two libraries are indistinguishable if all polynomial-time calling programs have **negligible advantage** in distinguishing them.

Indistinguishability

» An example:

- the calling program tries to predict which string will be chosen when uniformly sampling from $\{0,1\}^\lambda$.
- The left library tells whether its prediction was correct.
- The right library always returns false.

L_{left}	L_{right}
Predict(x): $s \to \{0,1\}^\lambda$ Return true if $x = s$, Return false otherwise	Predict(x): return false

How to distinguish two libraries?

Indistinguishability

The calling program A calls predict q times and outputs 1 if it ever received true as a response.

The calling program A
do q times: if predict$(0^\lambda) =$ true return 1 return 0

$\Pr[A \diamond L_{right} \Rightarrow 1] = 0$

$\Pr\left[A \diamond L_{left} \Rightarrow 1\right] = 1 - \left(1 - \frac{1}{2^\lambda}\right)^q \leq \frac{q}{2^\lambda}$

$|\Pr\left[A \diamond L_{left} \Rightarrow 1\right] \leq$ *Pr[first call to predict returns true}+Pr[second call to predict returns true}+...=*$\frac{q}{2^\lambda}$

$|\Pr\left[A \diamond L_{left} \Rightarrow 1\right] - \Pr[A \diamond L_{right} \Rightarrow 1]| \leq \frac{q}{2^\lambda}$

$\frac{q}{2^\lambda}$ are negligible when A is polynomial

Indistinguishability

- Other properties:

$$\textit{If } \mathcal{L}_1 \equiv \mathcal{L}_2 \textit{ then } \mathcal{L}_1 \approx \mathcal{L}_2. \textit{ Also, if } \mathcal{L}_1 \approx \mathcal{L}_2 \approx \mathcal{L}_3 \textit{ then } \mathcal{L}_1 \approx \mathcal{L}_3.$$

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

- We can prove indistinguishability using the hybrid technique.

Indistinguishability

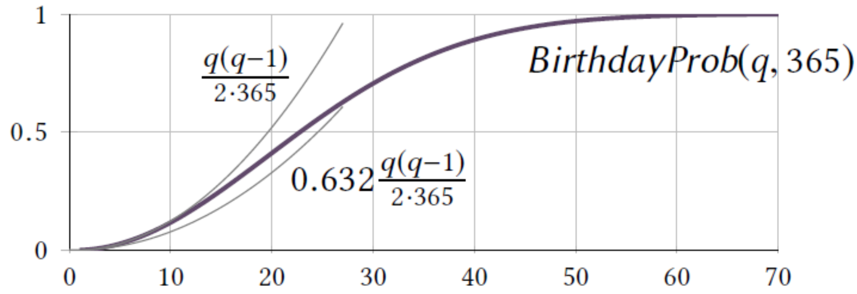
- Both libraries provide a SAMP subroutine that samples a random element of $\{0,1\}^\lambda$.
  - What is the distinguishing strategy?
  - Are the libraries interchangeable?
  - Are they indistinguishable?

$\mathcal{L}_{\text{samp-L}}$
$\underline{\text{SAMP}():}$
$\quad r \leftarrow \{0,1\}^\lambda$
$\quad$ return r

$\mathcal{L}_{\text{samp-R}}$
$R := \emptyset$
$\underline{\text{SAMP}():}$
$\quad r \leftarrow \{0,1\}^\lambda \setminus R$
$\quad R := R \cup \{r\}$
$\quad$ return r

Indistinguishability

- **Birthday probabilities (birthday paradox)**: If q people are in a room, what is the probability that two of them have the same birthday (if we assume that each person's birthday is uniformly chosen from among the possible days in a year)?
- $BirthdayProb(q, N) = 1 - \prod_{i=1}^{q-1}(1 - \frac{i}{N})$



Indistinguishability

- Best possible distinguishing strategy: call samp many times, If you ever see a repeated output, then you must certainly be linked to $\mathcal{L}_{samp-L}$.
- The advantage is exactly $BirthdayProb(q, 2^\lambda) = \Theta(q^2/2^\lambda)$.
- They are not interchangeable.
- They are indistinguishable.

$\mathcal{L}_{\text{samp-L}}$
$\underline{\text{SAMP}():}$ $\quad r \leftarrow \{0,1\}^\lambda$ $\quad \text{return } r$

$\mathcal{L}_{\text{samp-R}}$
$R := \emptyset$ $\underline{\text{SAMP}():}$ $\quad r \leftarrow \{0,1\}^\lambda \setminus R$ $\quad R := R \cup \{r\}$ $\quad \text{return } r$

THANKS!

Any questions?

You can find me at:

» zhanglan@ustc.edu.cn