

# Advanced Cryptography

(Provable Security)

## Introduction

Yi LIU

# About Me

Yi LIU (刘逸)

- Lecturer, College of Cyber Security, Jinan University
- Ph.D., The University of Hong Kong
- Research Interests: Multi-Party Computation
- Email: `liuyi@jnu.edu.cn`, `i@liuyi.pro`
- Homepage: <https://liuyi.pro>

# Course Information

- Time: 14 : 00 – 17 : 30, Thursday, Week 3-10
- Credit: 2
- QQ group: 1065158836



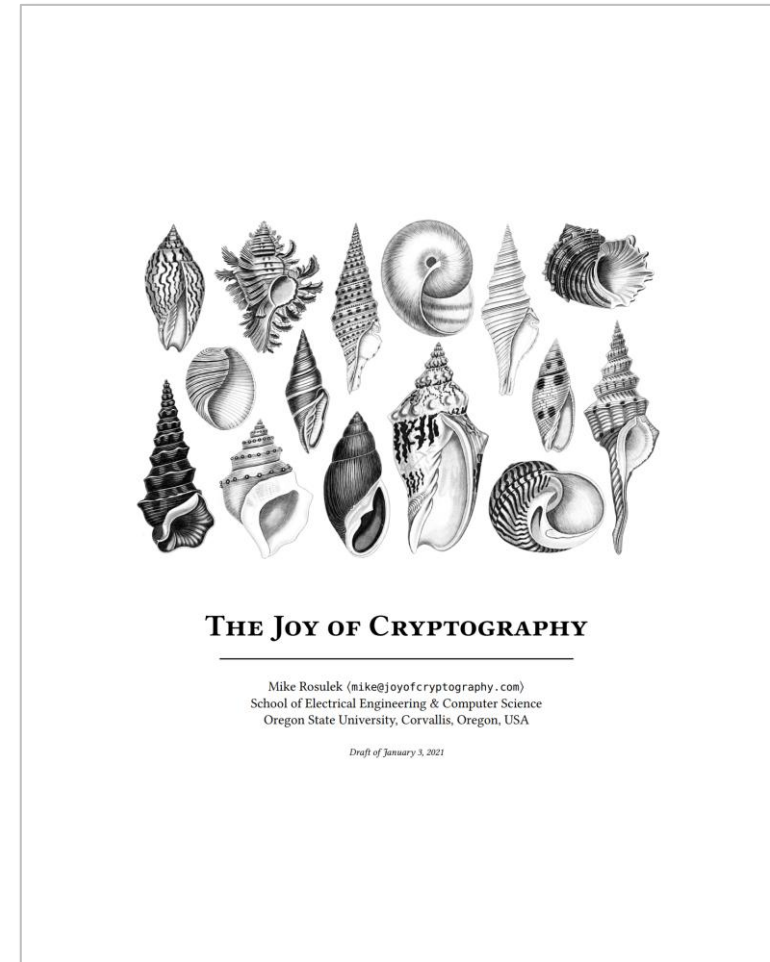
扫一扫二维码, 加入群聊

# Main Textbook

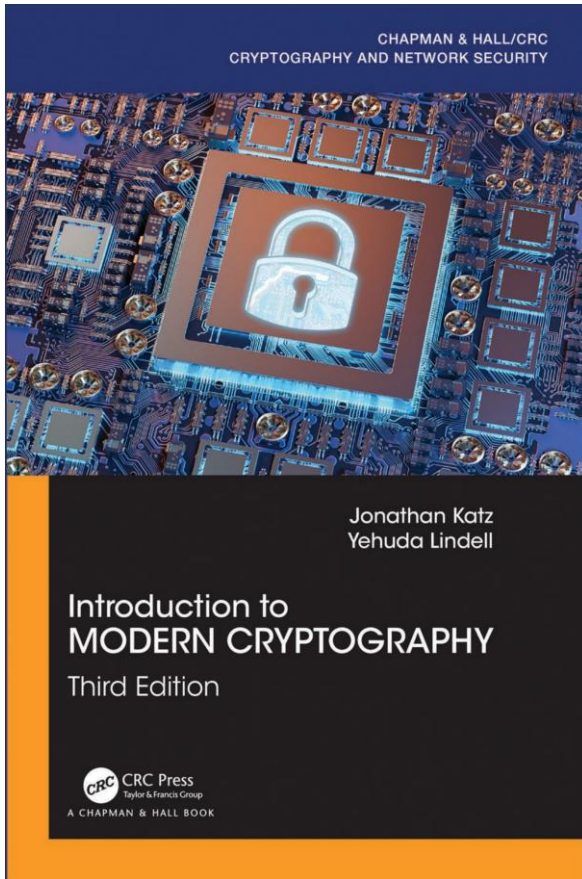
*The Joy of Cryptography*

by Mike Rosulek

Link: <https://joyofcryptography.com>



# Other Reading Resources



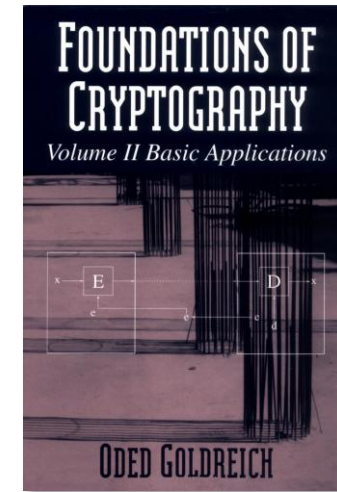
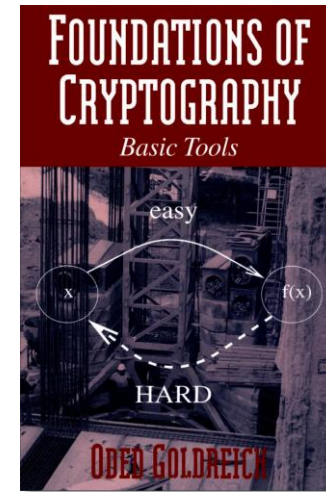
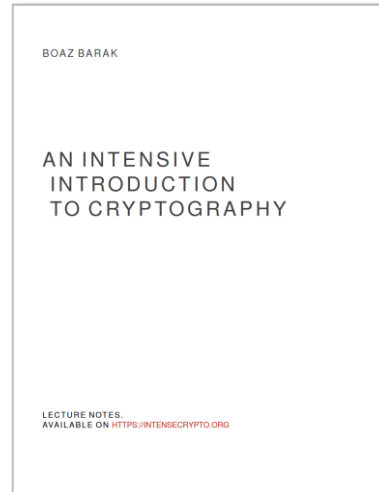
---

## A Graduate Course in Applied Cryptography

---

Dan Boneh and Victor Shoup

---



# What you will learn

- Foundations and principles of the science
- Basic primitives and components
- Definitions and proofs of security
- Will help you understand security proofs in crypto papers

# Marking Scheme (Tentative)

- Assignment (40%)
  - **MUST** be written via LaTeX
- Final exam (60%)

# Important Messages to Students

- Main ideas will be covered in class but some details might be skipped. You are responsible for **all materials** in the textbook & slides, even if they are not taught in class.
- Homework assignments should be worked on and written up **individually**, though collaboration on homework with other students are **encouraged** (acknowledge this).
- Any unintellectual behavior and cheating on exams, homework assignments will be dealt with **severely**.

# Important Messages to Students

- Please ask questions in class
  - If you're having trouble understanding something, then at least 50% of the class is also having trouble: they'll **be happy** if you ask for more explanation.
- If you find mistakes or just think that something's confusing
  - Please email me.

# Prerequisites of This Course

- Required:

1. Ability to read and write mathematical proofs and definitions.
2. Familiarity with algorithms – proving correctness and analyzing running time (**O notation**).
3. Familiarity with **basic probability theory**.
4. Familiarity with **discrete mathematics**.

- Helpful:

1. Complexity: P, NP, BPP, etc.
2. Number theory: modular arithmetic, prime numbers, etc.
3. Probabilistic algorithms: primality testing, etc.

# More About This Course

- This course is **NOT EASY**
  - Emphasis on proofs
  - Counter-intuitive concepts
  - Extensive use of quantifiers/probability
  - Need to acquire “crypto-intuition”

# Other Things You Should Know

- Where to find paper?
  - 中国密码学会推荐学术刊物和国际会议目录
    - <https://www.cacrnet.org.cn/site/content/1290.html>

编号	简称	会议名称	推荐级别
1	Crypto	International Cryptology Conference	A
2	Eurocrypt	European Cryptology Conference	A
3	Asiacrypt	Annual International Conference on the Theory and Application of Cryptology and Information Security	A
4	CCS	ACM Conference on Computer and Communications Security	A
5	IEEE S&P	IEEE Symposium on Security and Privacy	A
6	TCC	Theory of Cryptography Conference	B
7	PKC	International Workshop on Practice and Theory in Public Key Cryptography	B
8	FSE	Fast Software Encryption	B
9	CT-RSA	RSA Conference, Cryptographers' Track	B
10	CHES	Workshop on Cryptographic Hardware and Embedded Systems	B
11	ESORICS	European Symposium on Research in Computer Security	B
12	AsiaCCS	ACM Asia Conference on Computer and Communications Security	B
13	FC	Financial Cryptography and Data Security	B
14	PQCrypto	International Workshop on Post-Quantum Cryptography	B

编号	简称	期刊名称	推荐级别
1		Journal of Cryptology	A
2	TISSEC	ACM Transactions on Information and System Security	A
3		IEEE Transactions on Information Theory	A
4	TDSC	IEEE Transactions on Dependable and Secure Computing	A
5	TIFS	IEEE Transactions on Information Forensics and Security	A
6	JCR	密码学报	B
7	DCC	Designs, Codes and Cryptography	B
8	IJIS	International Journal of Information Security	B
9		IET Information Security	B
10	FFA	Finite Fields and Their Applications	B
11		Discrete Mathematics	B
12		Science China/中国科学	B
13		Discrete Applied Mathematics	B
14	IPL	Information Processing Letters	B
15	JOC	Journal of Complexity	B
16		Information Sciences	B

# Other Things You Should Know

- Where to find paper?
  - 中国密码学会推荐学术刊物和国际会议目录
    - <https://www.cacrnet.org.cn/site/content/1290.html>
- Cryptology ePrint Archive
  - <https://eprint.iacr.org/>

# Review of Concepts & Notation

# Logs & Exponents

- $(x^a)(x^b) = x^{a+b}$
- $(x^a)^b = x^{ab}$
- $\log_x(ab) = \log_x a + \log_x b$
- $a \log_x b = \log_x(b^a)$

# Modular Arithmetic

- Integers
  - $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$
  - Natural numbers:  $\mathbb{N} = \{0, 1, 2, \dots\}$
- Division & Modulo
  - For  $x, n \in \mathbb{Z}$ , we say that  $n$  **divides**  $x$ , and write  $n \mid x$ , if there exists an integer  $k$ , such that  $x = kn$ .
  - Let  $n$  be a positive integer and  $a$  be any integer,  $a \% n$  means  $a \bmod n$ 
    - If  $r = a \% n$ , then  $n \mid (a - r)$
    - $a \% n$  is always a **nonnegative** number

# Modular Arithmetic

- For positive  $n$ , Let  $\mathbb{Z}_n = \{0, \dots, n - 1\}$ , i.e.,  $\mathbb{Z}$  is the set of integers modulo  $n$ .
- We say that integers  $a$  and  $b$  are congruent modulo  $n$ , and write  $a \equiv_n b$ , if  $n \mid (a - b)$ . In other words,  $a \equiv_n b$  if and only if  $a \% n = b \% n$ .
  - E.g.,  $99 \equiv_{10} 19$

# Greatest Common Divisor (GCD)

- If  $d \mid x$  and  $d \mid y$ , then  $d$  is a common divisor of  $x$  and  $y$ .
- The **largest** possible such  $d$  is called the **greatest common divisor (GCD)**, denoted  $\text{gcd}(x, y)$ .
- If  $\text{gcd}(x, y) = 1$ , then we say that  $x$  and  $y$  are **relatively prime**.
- The oldest “algorithm” is the Euclid’s algorithm for computing GCD.

```
gcd(x, y): // Euclid's algorithm  
    if  $y = 0$  then return  $x$   
    else return  $\text{gcd}(y, x \% y)$ 
```

# Strings

- We write  $\{0,1\}^n$  to denote the **set** of  $n$ -bit binary string, and  $\{0,1\}^*$  to denote the set of all (**finite-length**) binary strings.
- We write  $0^n$  and  $1^n$  to denote strings of  $n$  zeros and  $n$  ones, respectively.
- If  $x$  is a **binary** string,  $|x|$  is the length (in bits) of  $x$ .
- When  $x, y \in \{0, 1\}^n$ ,  $x \oplus y$  is the bitwise exclusive-or (xor) of the two strings.

# Functions

Let  $X$  and  $Y$  be finite sets. A function  $f: X \rightarrow Y$  is:

- **Injective (1-to-1)**: if  $x \neq x' \Rightarrow f(x) \neq f(x')$ .
  - We must have  $|Y| \geq |X|$ .
- **Surjective (onto)**: for all  $y \in Y$ , there exists an element  $x \in X$  with  $f(x) = y$ .
  - We must have  $|Y| \leq |X|$ .
- **Bijective (1-to-1 correspondence)**: if  $f$  is both injective and surjective.
  - We must have  $|X| = |Y|$ .

# Notation in Pseudocode

- $x \leftarrow \mathcal{D}$ : sample  $x$  according to the distribution  $\mathcal{D}$ .
- $x \leftarrow X$ : if  $X$  is a finite set, then  $x$  is sampled from the **uniform distribution** over  $X$ .
- $x \leftarrow \mathcal{A}(y)$  : run a **probabilistic** algorithm  $\mathcal{A}$  on input  $y$  and assign the output to  $x$ .
- $x := y$ : take the value of expression  $y$  and assign it to variable  $x$ .

# Asymptotics

Let  $f: \mathbb{N} \rightarrow \mathbb{N}$  be a function.

- $f(n)$  is  $O(g(n)) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$

$$\Leftrightarrow \exists c > 0, \text{ for all but finitely many } n, f(n) \leq c \cdot g(n).$$

- $f(n)$  is  $\Omega(g(n)) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0$

$$\Leftrightarrow \exists c > 0, \text{ for all but finitely many } n, f(n) \geq c \cdot g(n).$$

- $f(n)$  is  $\Theta(g(n)) \Leftrightarrow f(n)$  is  $O(g(n))$  and  $f(n)$  is  $\Omega(g(n))$ .

$$\Leftrightarrow 0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

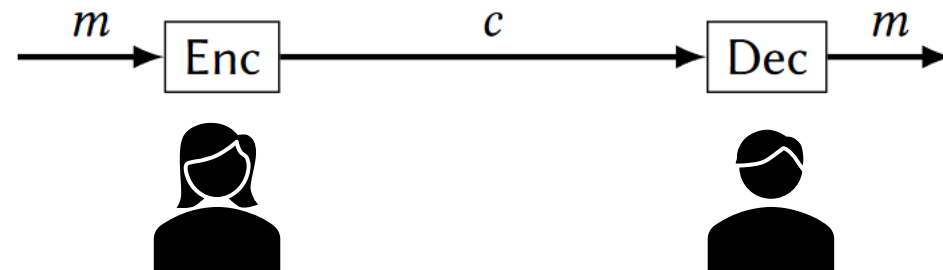
$$\Leftrightarrow \exists c_1, c_2 > 0, \text{ for all but finitely many } n, c_1 g(n) \leq f(n) \leq c_2 g(n)$$

# Kerckhoffs' Principle & Perfect Secrecy

# Encryption Scheme

Alice wants to send Bob a secret message.

- Algorithms
  - Key generation (KeyGen)
  - Encryption (Enc)
  - Decryption (Dec)



# Kerckhoffs' Principle (1883)

- Design your system to be secure even if the attacker has **complete knowledge of all its algorithms**.
- In other words, system should be **secure** even if **algorithms are known**, as long as key is secret.

**Q:** Why is it **OK** to assume that the algorithm is known?

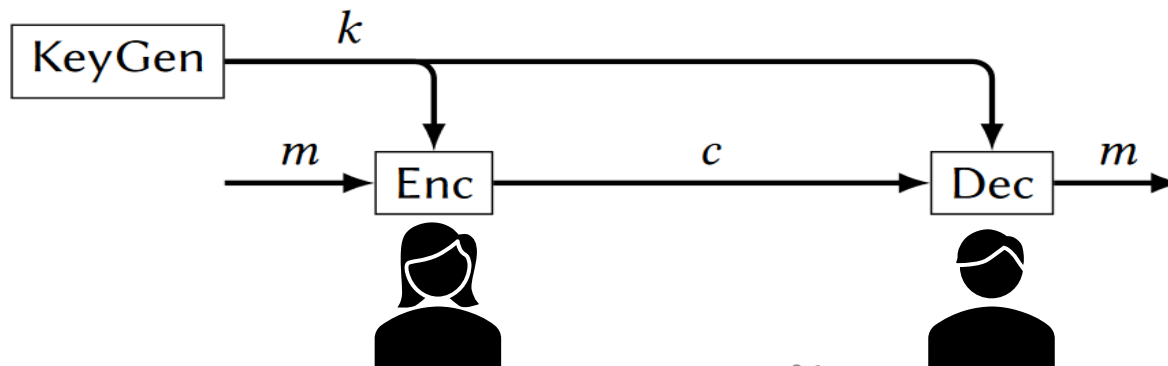
**A:** Because we can always choose a **fresh** key.

# Encryption Scheme

Alice wants to send Bob a secret message.

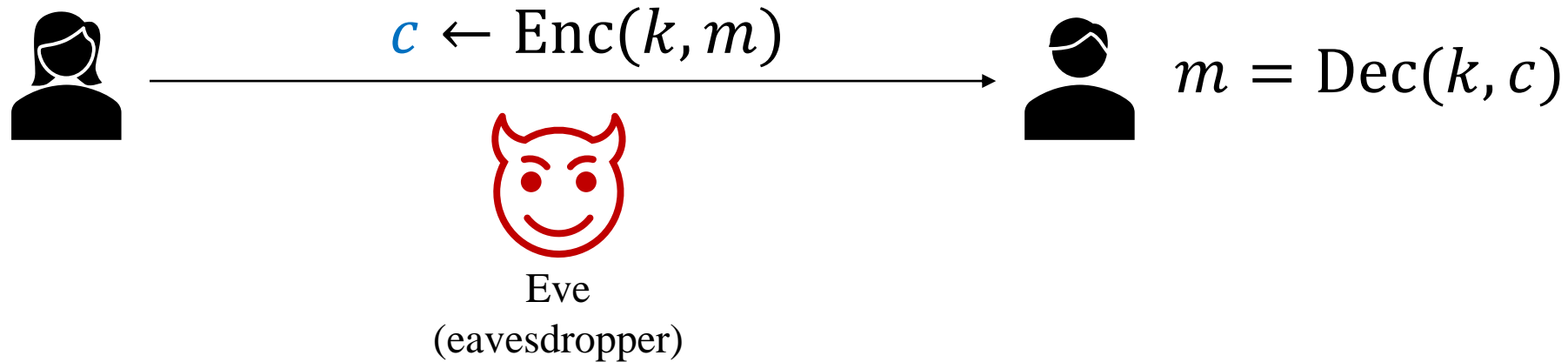
Symmetric-key encryption scheme with message space  $\mathcal{M}$ , ciphertext space  $\mathcal{C}$ , key space  $\mathcal{K}$  and algorithms:

- Key generation (KeyGen): generate  $k \in \mathcal{K}$
- Encryption (Enc):  $c \leftarrow \text{Enc}(k, m) \in \mathcal{C}$ , where  $m \in \mathcal{M}$
- Decryption (Dec):  $m := \text{Dec}(k, c) \in \mathcal{M}$



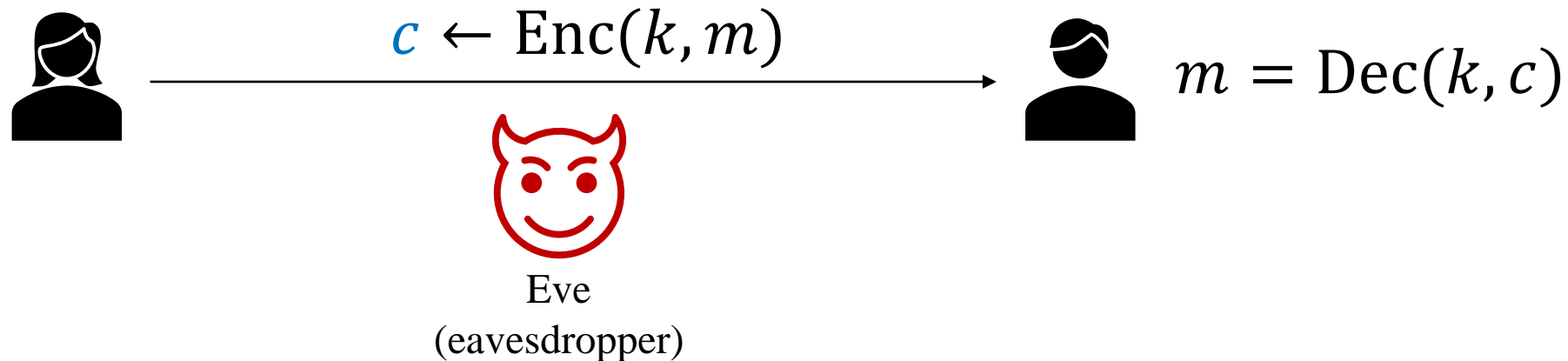
# Encryption Scheme

Alice wants to send Bob a secret message.



# Secure Encryption?

Alice wants to send Bob a secret message.



**Definition** (**Security of encryption**). An encryption scheme  $(\text{KeyGen}, \text{Enc}, \text{Dec})$  is  $\lambda$ -secure if no matter what method Eve employs, the probability that she can recover the true key  $k$  from the ciphertext  $c$  is at most  $2^{-\lambda}$ .

# Secure Encryption?

**Definition** ([Security of encryption](#)). An encryption scheme [\(KeyGen, Enc, Dec\)](#) is  $\lambda$ -[secure](#) if no matter what method Eve employs, the probability that she can recover the true key  $k$  from the ciphertext  $c$  is at most  $2^{-\lambda}$ .

**Q:** Is it a reasonable way to capture the real life phenomenon we are discussing?

It is too **weak**!

Consider: the secret key is chosen at random in  $\{0,1\}^n$  but our encryption scheme is simply [Enc](#)( $k, x$ ) =  $x$  and [Dec](#)( $k, y$ ) =  $y$ .

# Secure Encryption?

Consider: the secret key is chosen at random in  $\{0,1\}^n$  but our encryption scheme is simply  $\text{Enc}(k, x) = x$  and  $\text{Dec}(k, y) = y$ .

**Lemma** Let  $(\text{KeyGen}, \text{Enc}, \text{Dec})$  be the encryption scheme above. For every function  $Eve: \{0,1\}^* \rightarrow \{0,1\}^\lambda$  and for every  $x \in \{0,1\}^\ell$ , the probability that  $Eve(\text{Enc}(k, x)) = k$  is exactly  $2^{-\lambda}$ .

*Proof*

$Eva(\text{Enc}(k, x)) = eva(x)$  which is some fixed value  $k' \in \{0,1\}^\lambda$  independent of  $k$ . Hence the probability that  $k = k'$  is  $2^{-\lambda}$ .

**Problem:** Could be hard to learn key, but **easy** to learn message.

# Secure Encryption?

**Definition** (**Security of encryption**). An encryption scheme  $(\text{KeyGen}, \text{Enc}, \text{Dec})$  is  $\lambda$ -**secure** if for **every message**  $m$  no matter what method *Eve* employs, the probability that she can recover  $m$  from the ciphertext  $c$  is at most  $2^{-\lambda}$ .

Too **strong**, for “every message”, it is **impossible** to achieve!

- Consider:  $\text{Eve}(\text{Enc}(k, m)) = 0^\ell$  for all  $m$ .
- If  $m = 0^\ell$ , *Eve* can recover  $m$  with probability 1.

# Secure Encryption?

**Definition** (**Security of encryption**). An encryption scheme  $(\text{KeyGen}, \text{Enc}, \text{Dec})$  is  $\lambda$ -**secure** if  $m$  is chosen at random from  $\{0, 1\}^\ell$ , the probability that she can recover  $m$  from the ciphertext  $c$  is at most  $2^{-\lambda}$ .

Too **weak**!

- Consider: an encryption that **hides** the last  $\ell/2$  bits of the message, but completely **reveals** the first  $\ell/2$  bits. The probability of guessing a random message is  $2^{-\ell/2}$ , and so it would be  $\ell/2$ -**secure**.

# Secure Encryption?

## Perfect secrecy (informal)

- “Regardless of any **prior** information the attacker has about the plaintext, the ciphertext should leak **no additional information** about the plaintext”
- Attacker’s information about the plaintext = attacker-known **distribution of  $M$**  (random variable of the encrypted message)
- **Perfect secrecy** means that observing the ciphertext should **not** change the attacker’s knowledge about the distribution of  $M$

# Perfect Secrecy

**Definition** An encryption scheme  $(\text{Gen}, \text{Enc}, \text{Dec})$  with message space  $\mathcal{M}$  and ciphertext space  $\mathcal{C}$  is **perfectly secure** if **for every distribution over  $\mathcal{M}$ , every  $m \in \mathcal{M}$ , and every  $c \in \mathcal{C}$  with  $\Pr[C = c] > 0$ , it holds that**

$$\Pr[M = m \mid C = c] = \Pr[M = m]$$

**Key point:** The ciphertext  $c$  reveals **zero additional information** about the plaintext  $m$ .

# Perfect Secrecy

Another *equivalent* definition

**Definition** An encryption scheme  $(\text{Gen}, \text{Enc}, \text{Dec})$  with message space  $\mathcal{M}$  and ciphertext space  $\mathcal{C}$  is **perfectly secure** if **for every two distinct plaintexts  $\{m, m'\} \in \mathcal{M}$** , and every  $c \in \mathcal{C}$ , we have

$$\Pr[\text{Enc}(K, m) = c] = \Pr[\text{Enc}(K, m') = c]$$

(where the probabilities are over choice of  $K$  and any randomness of  $\text{Enc}$ )

# Perfect Secrecy

**Definition** An encryption scheme (Gen, Enc, Dec) with message space  $\mathcal{M}$  and ciphertext space  $\mathcal{C}$  is perfectly secure if for every distribution over  $\mathcal{M}$ , every  $m \in \mathcal{M}$ , and every  $c \in \mathcal{C}$  with  $\Pr[C = c] > 0$ , it holds that

$$\Pr[M = m \mid C = c] = \Pr[M = m]$$

**Definition** An encryption scheme (Gen, Enc, Dec) with message space  $\mathcal{M}$  and ciphertext space  $\mathcal{C}$  is perfectly secure if for every two distinct plaintexts  $\{m, m'\} \in \mathcal{M}$ , and every  $c \in \mathcal{C}$ , we have

$$\Pr[\text{Enc}(K, m) = c] = \Pr[\text{Enc}(K, m') = c]$$

(where the probabilities are over choice of  $K$  and any randomness of Enc)

*Proof*

$\Rightarrow$ :

$$\Pr[C = c \mid M = m] = \Pr[\text{Enc}(K, M) = c \mid M = m] = \Pr[\text{Enc}(K, m) = c \mid M = m] = \Pr[\text{Enc}(K, m) = c]$$

Then for any  $c \in \mathcal{C}$  with  $\Pr[C = c] > 0$ , we have

$$\Pr[M = m \mid C = c] \cdot \Pr[C = c] = \Pr[C = c \mid M = m] \cdot \Pr[M = m]$$

Take the uniform distribution over  $\mathcal{M}$ . If the scheme is perfectly secure, then  $\Pr[M = m \mid C = c] = \Pr[M = m]$ , and thus  $\Pr[C = c] = \Pr[C = c \mid M = m]$

Since  $m$  and  $c$  were arbitrary, we have that for every  $m, m' \in \mathcal{M}$  and every  $c \in \mathcal{C}$

$$\Pr[\text{Enc}(K, m) = c] = \Pr[C = c \mid M = m] = \Pr[C = c] = \Pr[C = c \mid M = m'] = \Pr[\text{Enc}(K, m') = c]$$

# Perfect Secrecy

**Definition** An encryption scheme (Gen, Enc, Dec) with message space  $\mathcal{M}$  and ciphertext space  $\mathcal{C}$  is perfectly secure if for every distribution over  $\mathcal{M}$ , every  $m \in \mathcal{M}$ , and every  $c \in \mathcal{C}$  with  $\Pr[C = c] > 0$ , it holds that

$$\Pr[M = m \mid C = c] = \Pr[M = m]$$

**Definition** An encryption scheme (Gen, Enc, Dec) with message space  $\mathcal{M}$  and ciphertext space  $\mathcal{C}$  is perfectly secure if for every two distinct plaintexts  $\{m, m'\} \in \mathcal{M}$ , and every  $c \in \mathcal{C}$ , we have

$$\Pr[\text{Enc}(K, m) = c] = \Pr[\text{Enc}(K, m') = c]$$

(where the probabilities are over choice of  $K$  and any randomness of Enc)

*Proof*

$\Leftarrow$ :

If  $\Pr[M = m] = 0$ , then we trivially have  $\Pr[M = m \mid C = c] = \Pr[M = m] = 0$

If  $\Pr[M = m] > 0$ :

For  $c \in \mathcal{C}$ , define  $p_c = \Pr[\text{Enc}(K, m) = c]$ .

Since  $\Pr[C = c \mid M = m] = \Pr[\text{Enc}(K, m) = c]$  and  $\Pr[\text{Enc}(K, m) = c] = \Pr[\text{Enc}(K, m') = c]$ , we have

$$\Pr[C = c \mid M = m'] = p_c$$

for every  $m' \in \mathcal{M}$ .

So,  $\Pr[C = c] = \sum_{m' \in \mathcal{M}} \Pr[C = c \mid M = m'] \cdot \Pr[M = m'] = \sum_{m' \in \mathcal{M}} p_c \cdot \Pr[M = m'] = p_c = \Pr[C = c \mid M = m]$

Based on the fact that  $\Pr[M = m \mid C = c] \cdot \Pr[C = c] = \Pr[C = c \mid M = m] \cdot \Pr[M = m]$ , we have

$$\Pr[M = m \mid C = c] = \Pr[M = m]$$

# Achieving Perfect Secrecy

The *XOR* operation:  $a \oplus b = a + b \bmod 2$

- $a \oplus 0 = a$
- $a \oplus a = 0$
- $a \oplus b = b \oplus a$  (**Commutativity**)
- $a \oplus (b \oplus c) = (a \oplus b) \oplus c$  (**Associativity**)

# Achieving Perfect Secrecy

**Theorem** (Vernam 1917, Shannon 1949): There is a perfectly secret valid encryption scheme **(Gen, Enc, Dec)** with  $\ell(\lambda) = \lambda$ .

## One-Time Pad

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

# One-Time Pad - Correctness

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

$$\begin{aligned} \text{Dec}(k, \text{Enc}(k, m)) &= \text{Dec}(k, k \oplus m) \\ &= k \oplus (k \oplus m) = (k \oplus k) \oplus m = 0^\lambda \oplus m = m \end{aligned}$$

# One-Time Pad - Security

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

**Theorem** One-time pad is perfectly secure.

*Proof*

$$\begin{aligned}\Pr[C = c \mid M = m] &= \Pr[K \oplus m = c \mid M = m] \\ &= \Pr[K = m \oplus c \mid M = m] = 2^{-\lambda}\end{aligned}$$

$$\begin{aligned}\Pr[C = c] &= \sum_{m \in \mathcal{M}} \Pr[C = c \mid M = m] \cdot \Pr[M = m] \\ &= 2^{-\lambda} \cdot \sum_{m \in \mathcal{M}} \Pr[M = m] = 2^{-\lambda}\end{aligned}$$

# One-Time Pad - Security

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

**Theorem** One-time pad is perfectly secure.

*Proof*

$$\begin{aligned}\Pr[C = c \mid M = m] &= 2^{-\lambda} \\ \Pr[C = c] &= 2^{-\lambda}\end{aligned}$$

Using Bayes' Theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} = \frac{2^{-\lambda} \cdot \Pr[M = m]}{2^{-\lambda}} = \Pr[M = m]$$

# Another Perspective

- From Eve's perspective

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

- Our goal is to say something like “the output of **eavesdrop** doesn't reveal the input  $m$ .”
- Since  $c = k \oplus m \Leftrightarrow k = m \oplus c$ ,  $\Pr[\mathbf{eavesdrop}(m) = c] = \Pr[k = m \oplus c]$ .  
For every  $m$  and  $c$ , the probability that  $\mathbf{eavesdrop}(m) = c$  is  $2^{-\lambda}$  (since  $k \in \{0,1\}^\lambda$  and only one value of  $k$  makes  $k = m \oplus c$  true).
  - This means that the output of  $\mathbf{eavesdrop}(m)$ , for any  $m$ , follows the uniform distribution.

# Another Perspective

- From Eve's perspective

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

- Our goal is to say something like “the output of **eavesdrop** doesn't reveal the input  $m$ .”
- If an attacker sees a single ciphertext, encrypted with **one-time pad**, where the key is chosen uniformly and kept secret from the attacker, then the ciphertext appears **uniformly distributed**.

# Key Generation

- When describing algorithms, we assume access to **uniformly distributed bits/bytes**. Where do these actually come from?
- Precise details depend on the system
  - Linux or unix: /dev/random or /dev/urandom
  - **Do not** use rand() or java.util.Random
  - Use crypto libraries instead

# Generating Randomness

**Q:** How do we get random bits in actual systems?

**A:** Two steps:

- Continually collect a “pool” of **high-entropy** (“unpredictable”) data
  - Must ultimately come from some physical process (keystroke/mouse movements, delays between network events, hard-disk access times, hardware random-number generation (Intel), etc.)
- When random bits are requested, process this data to generate a sequence of uniform, independent bits/bytes
  - Need to eliminate both bias and dependencies

**Q:** From a sequence of **biased** coin flips, where heads with prob.  $p$  and tails with prob.  $1 - p$ , how can we “smooth” this kind of data?

# Generating Randomness

**Q:** From a sequence of **biased** coin flips, where heads with prob.  $p$  and tails with prob.  $1 - p$ , how can we “smooth” this kind of data?

**A:** **von Neumann technique** for eliminating bias:

- Collect two bits per output bit
  - $01 \rightarrow 0$ ,  $10 \rightarrow 1$ ,  $00, 11 \rightarrow \text{skip}$
  - Note that this assumes **independence** (as well as constant bias)

# One-Time Pad

**Q:** Is this the end of cryptography?

**No.** We need more:

- Use the **same** key for **many** plaintexts
- Use  **$n$ -bit** key for  **$2n$ -bit** plaintexts.

**Theorem** (**Limitations of perfect secrecy**) If **(Gen, Enc, Dec)** is a **perfectly secret** encryption scheme with message space  $\mathcal{M}$  and key space  $\mathcal{K}$ , then  $|\mathcal{K}| \geq |\mathcal{M}|$ .

# Limitations of Perfect Secrecy

**Theorem** (Limitations of perfect secrecy) If  $(\text{Gen}, \text{Enc}, \text{Dec})$  is a perfectly secret encryption scheme with message space  $\mathcal{M}$  and key space  $\mathcal{K}$ , then  $|\mathcal{K}| \geq |\mathcal{M}|$ .

*Proof*

Assume  $|\mathcal{K}| < |\mathcal{M}|$ . Let  $c \in \mathcal{C}$  be a ciphertext that occurs with nonzero probability. Let

$$\mathcal{M}(c) = \{ m \mid m = \text{Dec}(k, c) \text{ for some } k \in \mathcal{K} \}$$

Clearly,  $|\mathcal{M}(c)| \leq |\mathcal{K}|$ . If  $|\mathcal{K}| < |\mathcal{M}|$ , there is some  $m' \in \mathcal{M}$ , s.t.,  $m' \notin \mathcal{M}(c)$ .

But then

$$\Pr[ M = m' \mid C = c ] \neq \Pr[M = m']$$

# Limitations of Perfect Secrecy

**Theorem** (Limitations of perfect secrecy) If  $(\text{Gen}, \text{Enc}, \text{Dec})$  is a perfectly secret encryption scheme with message space  $\mathcal{M}$  and key space  $\mathcal{K}$ , then  $|\mathcal{K}| \geq |\mathcal{M}|$ .

- The key is **as long as** the message
- **Only secure** if **each** key is used to encrypt a **single** message

**Q:** What about if using the same key twice?

Say  $c_1 = k \oplus m_1$  and  $c_2 = k \oplus m_2$ .

$$c_1 \oplus c_2 = (k \oplus m_1) \oplus (k \oplus m_2) = m_1 \oplus m_2$$

$m_1 \oplus m_2$  **leaks** information about  $m_1, m_2$ .

# Limitations of Perfect Secrecy

**Q:** length of key = length of plaintext is impractical. What people do in practice?

**A:** In real life, people are using encryption with keys shorter than the message size to encrypt all kinds of sensitive information.

If the algorithm you use to break the encryption scheme runs in time  $2^\lambda$ , it seems **OK** since the message may be expired by then ...

- Computational Security – [Modern Cryptography](#)

# Principles of Modern Cryptography

- Principle 1: Formal Definitions
- Principle 2: Precise Assumptions
- Principle 3: Proofs of Security

# Importance of Definitions

- Definitions are **essential** for the **design**, **analysis**, and **usage of crypto**
- **Security guarantee/goal**
  - What we want to achieve and/or what we want to prevent the attacker from achieving
- **Threat model**
  - What (real-world) capabilities the attacker is assumed to have

# Importance of Definitions - Design

- Developing a precise **definition** forces the designer to think about what they really want
  - What is **essential** and (sometimes more important) what is not
  - Often reveals subtleties of the problem

*If you don't understand what you want to achieve, how can you possibly know when (or if) you have achieved it?*

# Importance of Definitions - Analysis

- Definitions enable meaningful analysis, evaluation, and comparison of schemes
  - Does a scheme satisfy the definition?
  - What definition does it satisfy?

*One scheme may be less efficient than another, yet satisfy a stronger security definition.*

# Importance of Definitions - Usage

- Definitions allow to understand the security guarantees provided by a scheme
- Enable schemes to be used as **components** of a larger system (modularity)
- Enables one scheme to be **substituted** for another if they satisfy the same definition

# Assumptions

- With few exceptions, cryptography currently requires **computational assumptions**
  - At least until we prove  $P \neq NP$  (even that would not be enough)
- Principle: any such assumptions should be made **explicit**

# Importance of Clear Assumptions

- Allow researchers to (attempt to) **validate** assumptions by studying them
- Allow meaningful **comparison** between schemes based on different assumptions
  - Useful to understand **minimal** assumptions needed
- Practical implications if assumptions are **wrong**
- Enable proofs of security

# Proofs of Security

- Provide a **rigorous proof** that a construction satisfies a given **definition** under certain specified **assumptions**
- Proofs are **crucial** in crypto, where there is a malicious attacker trying to “break” the scheme

# Limitations?

- Provably secure schemes can be broken!
  - If the definition does not correspond to the real-world threat model
  - If the assumption is invalid
  - If the implementation is flawed

# The Basic of Provable Security

# Provable Security

*“Human ingenuity cannot concoct a cipher which human ingenuity cannot resolve.”*

by Edgar Allan Poe

- This was an accurate assessment of the cryptography that existed in 1841.
  - cat-and-mouse game
- Modern 21st-century cryptography, however, is **different**.
  - The code-makers can win against the code-breakers.
  - **Prove** the security!

# How to Write a Security Definition

- A security definition should give guarantees about what can happen to a system **in the presence of an attacker**.
- But, **not** all important properties of a system refer to an attacker.
  - We don't reference any attacker when we say that the Enc algorithm takes two arguments (a key and a plaintext), or that the KeyGen algorithm takes no arguments. We call these properties the **syntax** of the scheme.
  - Even if there is no attacker, it's still important that decryption is an inverse of encryption. This is not a security property of the encryption scheme. Instead, we call it a **correctness** property.
- Security definitions always consider the **attacker's view** of the system.

# 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  $m \in \mathcal{M}$ .

# Encryption - Correctness

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

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

A scheme defined by  $\text{Enc}(k, m) = m$  and  $\text{Dec}(k, c) = c$

- **does satisfy** the correctness property
- **would not satisfy** any reasonable security property

# “Real-vs-Random” Style of Security Definition

An intuitive idea

*“an encryption scheme is a good one if its ciphertexts look like **random junk** to an attacker.”*

- **Key** is kept secret from the attacker.
  - Consider the scheme where the key is used to encrypt **one** plaintext
  - Will consider one-key-many-plaintexts case later
- **Plaintext** is chosen by the attacker
  - A “**pessimistic**” choice: give much power to the attacker
  - If a scheme is secure **when the attacker chooses the plaintext**, then it **should be secure in more realistic scenarios** where the attacker has some uncertainty about the plaintexts.

# “Real-vs-Random” Style of Security Definition

An intuitive idea

*“an encryption scheme is a good one if its ciphertexts look like **random junk** to an attacker . . . when each key is **secret** and used to encrypt **only one** plaintext, even when the attacker chooses the plaintexts.”*

How to express these details?

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

```
ctxt( $m \in \Sigma.\mathcal{M}$ ):  
   $k \leftarrow \Sigma.\text{KeyGen}$   
   $c := \Sigma.\text{Enc}(k, m)$   
  return  $c$ 
```

Vs.

```
ctxt( $m \in \Sigma.\mathcal{M}$ ):  
   $c \leftarrow \Sigma.\mathcal{C}$   
  return  $c$ 
```

# “Real-vs-Random” Style of Security Definition

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

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

$k \leftarrow \Sigma.\text{KeyGen}$

$c := \Sigma.\text{Enc}(k, m_L)$

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  $c$

# “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  $\mathcal{A}$

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

$\mathcal{L}$
<b>ctxt</b> ( $m$ ): $k \leftarrow \{0,1\}^\lambda$ $c := k \oplus m$ return $c$

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

# Formalisms for Security Definitions

## Example

- A library  $\mathcal{L}$

$\mathcal{L}$
$s \leftarrow \{0,1\}^\lambda$ <b>reset():</b> $s \leftarrow \{0,1\}^\lambda$ <b>guess</b> ( $x \in \{0,1\}^\lambda$ ): return $x =_? s$

- 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**).

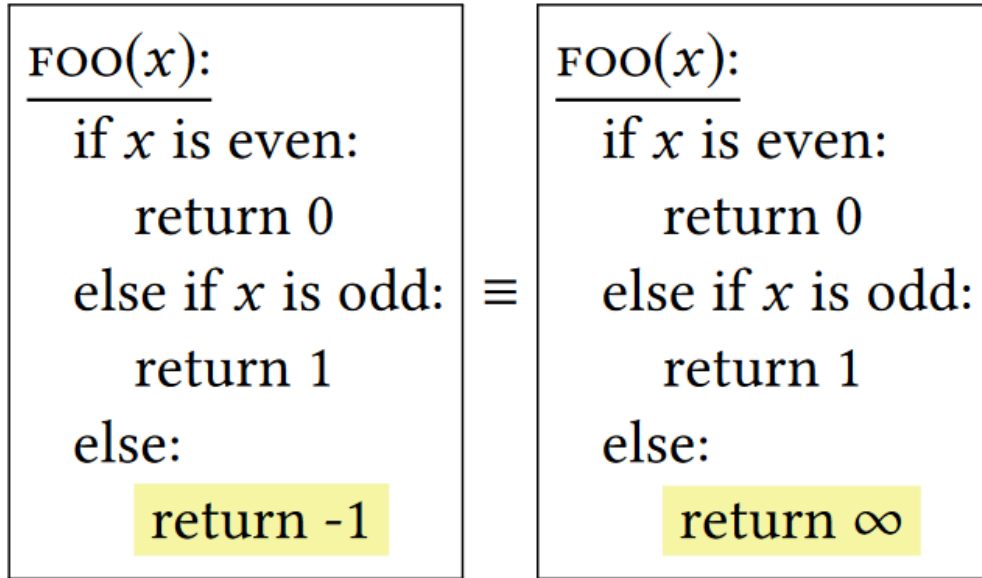
# Interchangeability

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

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

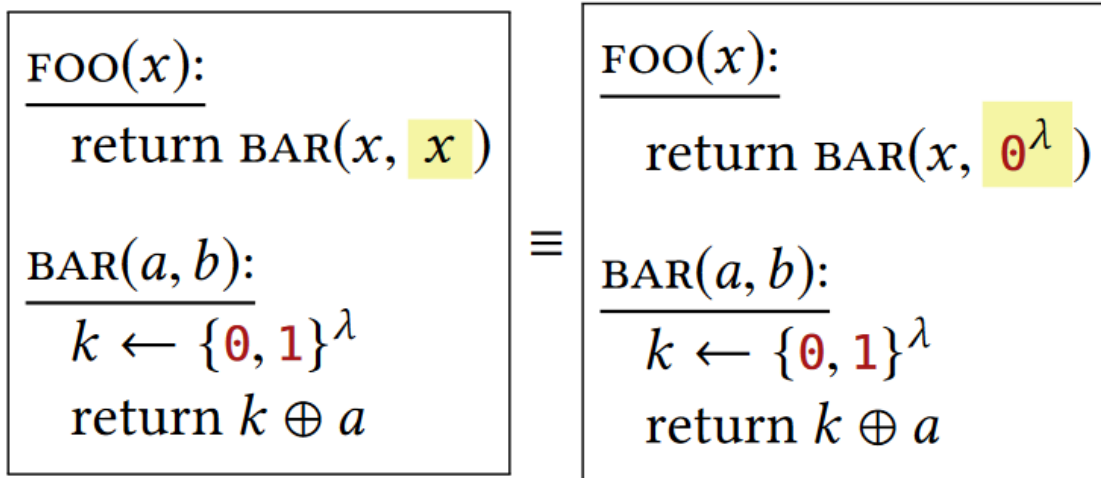
- We often refer to the calling program as a **distinguisher**.
- A Boolean output is enough for that task. Think of the output bit as the calling program's “**guess**” for which library the calling program thinks it is linked to.
- The distinction between “calling program **outputs true**” and “calling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

# Interchangeability - Example



*Their only difference happens in an unreachable block of code.*

# Interchangeability - Example



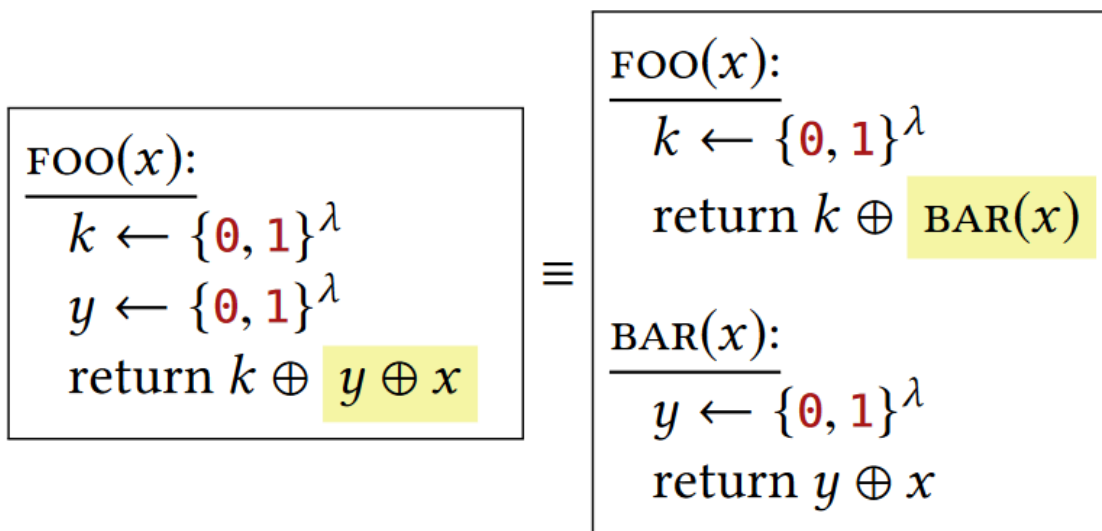
*Their only difference is the value they assign to a variable that is never actually used.*

# Interchangeability - Example

$$\boxed{\begin{array}{c} \text{FOO}(x, n): \\ \hline \text{for } i = 1 \text{ to } \lambda: \\ \quad \text{BAR}(x, i) \end{array}} \equiv \boxed{\begin{array}{c} \text{FOO}(x, n): \\ \hline \text{for } i = 1 \text{ to } n: \\ \quad \text{BAR}(x, i) \\ \text{for } i = n + 1 \text{ to } \lambda: \\ \quad \text{BAR}(x, i) \end{array}}$$

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

# Interchangeability - Example



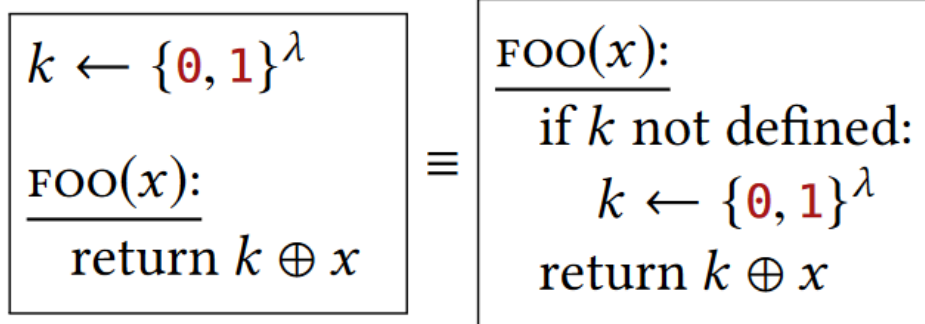
*Their only difference is that one library inlines a subroutine call that occurs in the other library.*

# Interchangeability - Example

$$\boxed{\begin{array}{l} \text{FOO():} \\ \hline x \leftarrow \{\text{0}, \text{1}\}^\lambda \\ y \leftarrow \{\text{0}, \text{1}\}^\lambda \\ \text{return } x \parallel y \end{array}} \equiv \boxed{\begin{array}{l} \text{FOO():} \\ \hline z \leftarrow \{\text{0}, \text{1}\}^{2\lambda} \\ \text{return } z \end{array}}$$

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

# Interchangeability - Example



*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  $\text{FOO}$ , and initially undefined.*

# Formal Restatements of Previous Concepts

- “real-vs-random” style
  - **Definition** An encryption scheme  $\Sigma$  has **one-time uniform ciphertexts** if:

$\mathcal{L}_{\text{ots}\$-\text{real}}^\Sigma$		$\mathcal{L}_{\text{ots}\$-\text{rand}}^\Sigma$
<b>ctxt</b> ( $m \in \Sigma.\mathcal{M}$ ): $k \leftarrow \Sigma.\text{KeyGen}$ $c := \Sigma.\text{Enc}(k, m)$ return $c$	$\equiv$	<b>ctxt</b> ( $m \in \Sigma.\mathcal{M}$ ): $c \leftarrow \Sigma.\mathcal{C}$ return $c$

# Formal Restatements of Previous Concepts

- “left-vs-right” style
  - **Definition** An encryption scheme  $\Sigma$  has **one-time secrecy** if:

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

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

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

 $\equiv$ 

$\mathcal{L}_{\text{ots\$-rand}}^\Sigma$
<b>ctxt</b> ( $m \in \Sigma. \mathcal{M}$ ): $c \leftarrow \Sigma. \mathcal{C}$ 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*

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

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

$\equiv?$

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

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

$\diamond$

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

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

# 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*

<div style="border: 1px solid black; padding: 10px; text-align: center;"> <math>\mathcal{A}</math> </div> <div style="border: 1px solid black; padding: 10px;"> <math>c := \text{ctxt}(0^\lambda)</math>  return <math>c \stackrel{?}{=} 0^\lambda</math> </div>	$\diamond$	<div style="border: 1px solid black; padding: 10px;"> <div style="background-color: #f0f0f0; padding: 5px; text-align: center; color: red;"> <math>\mathcal{L}_{\text{ots\\$-rand}}^\Sigma</math> </div> <div> <b>ctxt</b>(<math>m \in \Sigma.\mathcal{M}</math>):  <math>c \leftarrow \{0,1\}^\lambda</math>  return <math>c</math> </div> </div>	$\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{ots\$-real}}^\Sigma \Rightarrow \text{true}] = 1$
			$\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{ots\$-rand}}^\Sigma \Rightarrow \text{true}] = 1/2^\lambda$

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

# Demonstrate Insecurity with Attacks

Insecure construction

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

**Claim** This construction **does not** satisfy one-time secrecy.

*Proof*

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

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

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

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

◇

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

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

# Demonstrate Insecurity with Attacks

Insecure construction

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

**Claim** This construction **does not** satisfy one-time secrecy.

*Proof*

<div><math>\mathcal{A}</math></div> <div> <math>c := \text{eavesdrop}(0^\lambda, 1^\lambda)</math>  return <math>c =_? 0^\lambda</math> </div>	◊	<div><math>\mathcal{L}_{\text{ots-R}}^\Sigma</math></div> <div> <b>eavesdrop</b>(<math>m_L, m_R</math>):  <math>k \leftarrow \{0,1\}^\lambda</math>  <math>c := k \&amp; m_R</math>  return <math>c</math> </div>	$\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{ots-L}}^\Sigma \Rightarrow \text{true}] = 1$  $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{ots-R}}^\Sigma \Rightarrow \text{true}] = 1/2^\lambda$
--	---	---	--

Since these two probabilities are different, this shows that  $\mathcal{L}_{\text{ots-L}}^\Sigma \not\equiv \mathcal{L}_{\text{ots-R}}^\Sigma$ . 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  $\mathcal{L}_{\text{left}} \equiv \mathcal{L}_{\text{right}}$ , then for any library  $\mathcal{L}^*$ , we have  $\mathcal{L}^* \diamond \mathcal{L}_{\text{left}} \equiv \mathcal{L}^* \diamond \mathcal{L}_{\text{right}}$ .

*Proof*

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

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