Department of
Computer Science and Engineering
计算机科学与工程系

CSE5014 CRYPTOGRAPHY AND NETWORK SECURITY

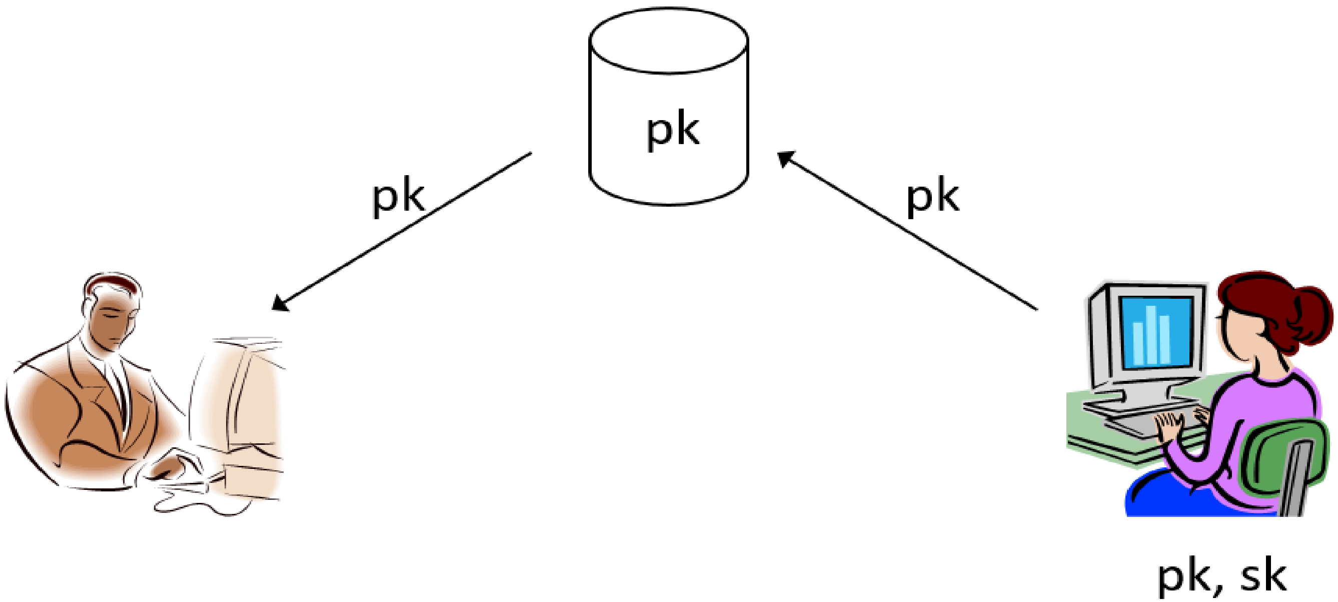
Dr. Qi Wang

Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

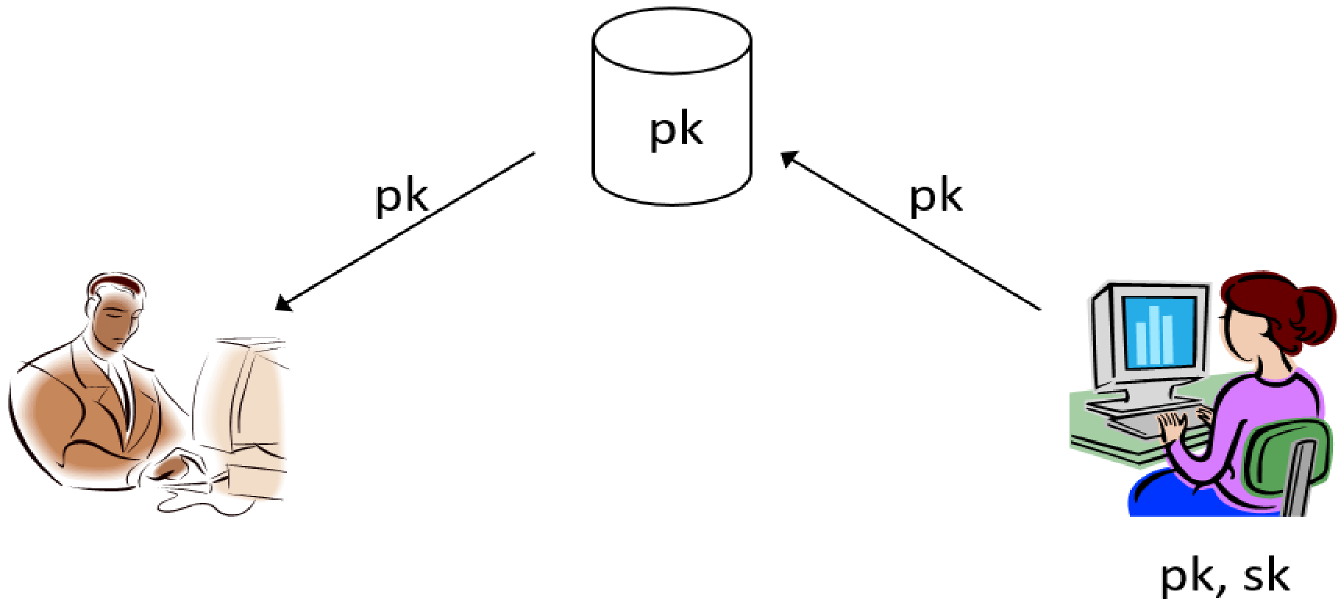
The public-key setting

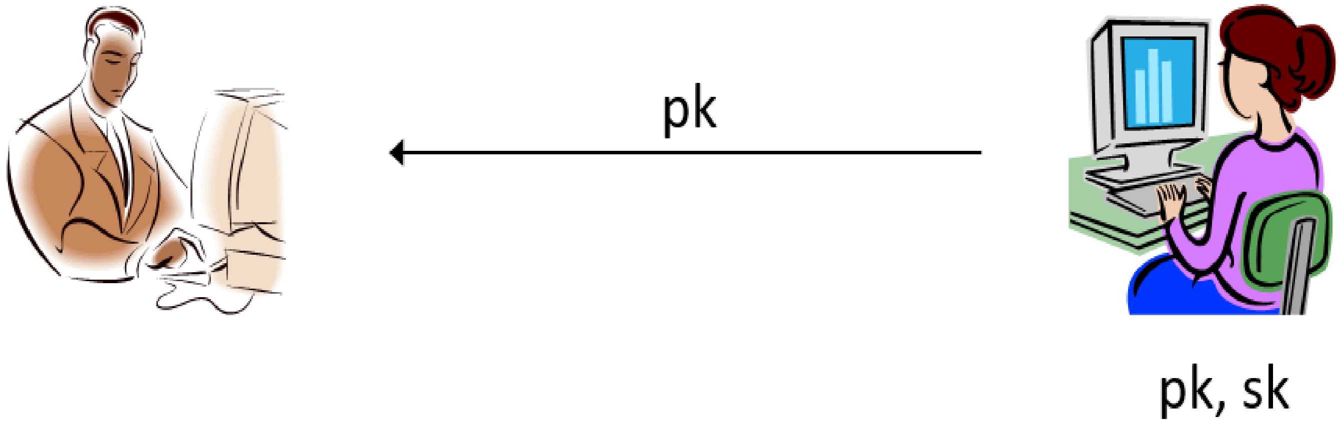
- A party generates a *pair* of keys: a *public* key *pk* and a *private* key *sk*
  - Public key is widely disseminated
  - Private key is kept secret, and shared with no one
- Private key used by the party who generated it; public key used by everyone else
  - Also called *asymmetric cryptography*
- Security must hold even if the attacker knows *pk*

Public-key distribution I, II



Public-key distribution I, II





Public-key distribution

- Previous figures (implicitly) assume parties are able to obtain correct cipies of each others' public keys
  - I.e., the attacker is *passive* during key distribution

Public-key distribution

- Previous figures (implicitly) assume parties are able to obtain correct cipies of each others' public keys
  - I.e., the attacker is *passive* during key distribution
- We will revisit this assumption later

Public-key distribution

- Previous figures (implicitly) assume parties are able to obtain correct cipies of each others' public keys
  - I.e., the attacker is *passive* during key distribution
- We will revisit this assumption later

	Private-key setting	Public-key setting
Secrecy	Private-key encryption	Public-key encryption
Integrity	Message authentication codes	Digital signature schemes

Addressing drawbacks of private-key crypto

- Key distribution
  - Public keys can be distributed over *public* (but *authenticated*) channels!

Addressing drawbacks of private-key crypto

- Key distribution
  - Public keys can be distributed over *public* (but *authenticated*) channels!
- Key management in large systems of N users
  - Each user stores 1 private key and $N - 1$ public keys; only N keys overall
  - Public keys can be stored in a central directory

Addressing drawbacks of private-key crypto

- Key distribution
  - Public keys can be distributed over *public* (but *authenticated*) channels!
- Key management in large systems of N users
  - Each user stores 1 private key and $N - 1$ public keys; only N keys overall
  - Public keys can be stored in a central directory
- Applicability in "open systems"
  - Even parties who have no prior relationship can find each others' public keys and use them

Why study private-key crypto?

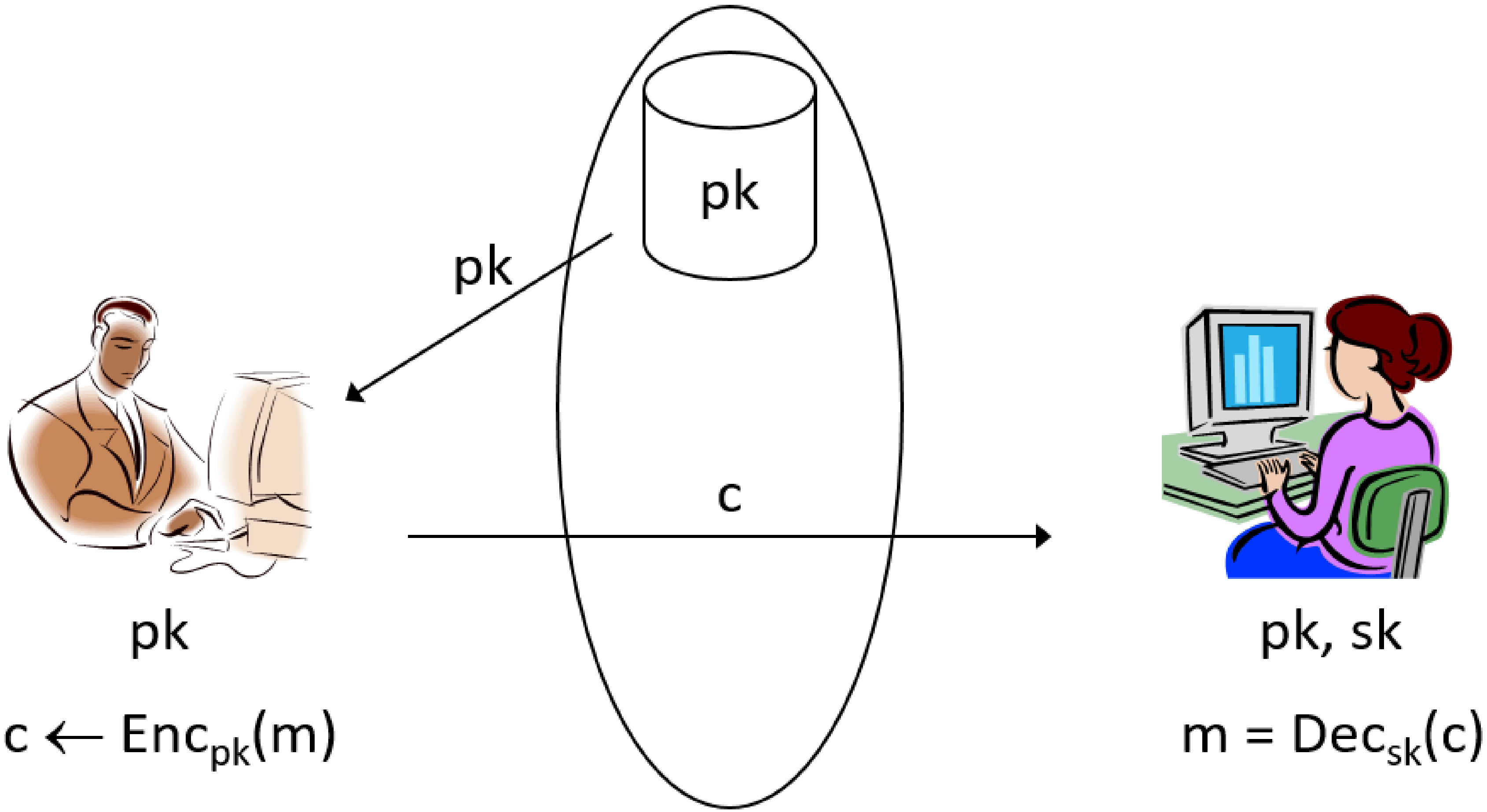
- Private-key cryptography is more suitable for certain applications
  - E.g., disk encryption

Why study private-key crypto?

- Private-key cryptography is more suitable for certain applications
  - E.g., disk encryption

- Public-key crypto is roughly $2 - 3$ orders of magnitude *slower* than private-key crypto
  - If private-key crypto is an option, use it!
  - Private-key crypto is used for *efficiency* even in the public-key setting

Public-key encryption



Public-key encryption

- **Theorem 12.2** A *public-key encryption* scheme is composed of three PPT algorithms:
  - *Gen*: *key-generation algorithm* that on input 1^n outputs pk, sk
  - *Enc*: *encryption algorithm* that on input pk and a message m outputs a ciphertext c
  - *Dec*: *decryption algorithm* that on input sk and a ciphertext c outputs a message m or an error $\perp$

- **Theorem 12.2** A *public-key encryption* scheme is composed of three PPT algorithms:
  - *Gen*: *key-generation algorithm* that on input 1^n outputs pk, sk
  - *Enc*: *encryption algorithm* that on input pk and a message m outputs a ciphertext c
  - *Dec*: *decryption algorithm* that on input sk and a ciphertext c outputs a message m or an error $\perp$

  For all m and pk, sk output by Gen,

$$Dec_{sk}(Enc_{pk}(m)) = m$$

CPA-security

- Fix a public-key encryption scheme Π and an adversary A
- Define experiment $PubK\text{-}CPA_{A,\Pi}(n)$:
  – Run $Gen(1^n)$ to get keys pk, sk
  – Give pk to A, who outputs m_0, m_1 of same length
  – Choose uniform $b \in \{0, 1\}$ and compute the ciphertext $c \leftarrow Enc_{pk}(m_b)$; give c to A
  – A outputs a guess b', and the experiment evaluates to 1 if $b' = b$

CPA-security

- Fix a public-key encryption scheme Π and an adversary A
- Define experiment $PubK\text{-}CPA_{A,\Pi}(n)$:
  - Run $Gen(1^n)$ to get keys pk, sk
  - Give pk to A, who outputs m_0, m_1 of same length
  - Choose uniform $b \in \{0, 1\}$ and compute the ciphertext $c \leftarrow Enc_{pk}(m_b)$; give c to A
  - A outputs a guess b', and the experiment evaluates to 1 if $b' = b$
- **Theorem 12.3** Public-key encryption scheme Π is *CPA-secure* if for all PPT adversaries A:

$$\Pr[PubK\text{-}CPA_{A,\Pi}(n) = 1] \leq 1/2 + negl(n)$$

Notes on the definition

- No encryption oracle?!

Notes on the definition

- No encryption oracle?!
  - Encryption oracle redundant in public-key setting

Notes on the definition

- No encryption oracle?!
  - Encryption oracle redundant in public-key setting

$\Rightarrow$ No *perfectly secret* public-key encryption

$\Rightarrow$ No *deterministic* public-key encryption can be CPA-secure

$\Rightarrow$ CPA-security implies security for encryption multiple messages as in the private-key case

Perfectly secret public-key encryption

- **Definition 12.4** A public-key encryption scheme is *perfectly secret* if for all public keys pk, all messages m_0, m_1, all ciphertexts c, and all algorithms A, we have:
  $\Pr[A(pk, c) = 0 | c \leftarrow Enc_{pk}(m_0)] = \Pr[A(pk, c) = 0 | c \leftarrow Enc_{pk}(m_1)]$

Perfectly secret public-key encryption

- **Definition 12.4** A public-key encryption scheme is *perfectly secret* if for all public keys pk, all messages m_0, m_1, all ciphertexts c, and all algorithms A, we have:

$$\Pr[A(pk, c) = 0 | c \leftarrow Enc_{pk}(m_0)] = \Pr[A(pk, c) = 0 | c \leftarrow Enc_{pk}(m_1)]$$

Theorem 12.5 No public-key encryption scheme is *prefectly secret*.

Proof.

Recall: plain RSA

- Choose random, equal-length primes p, q
- Compute modulus $N = pq$
- Choose e, d such that $e \cdot d = 1 \bmod \phi(N)$

Recall: plain RSA

- Choose random, equal-length primes p, q
- Compute modulus $N = pq$
- Choose e, d such that $e \cdot d = 1 \bmod \phi(N)$

- The e^{th} root of x modulo N is $x^d \bmod N$

$$(x^d)^e = x^{de} = x \bmod N$$

Recall: plain RSA

- Choose random, equal-length primes p, q
- Compute modulus $N = pq$
- Choose e, d such that $e \cdot d = 1 \bmod \phi(N)$

- The e^{th} root of x modulo N is $x^d \bmod N$

$$(x^d)^e = x^{de} = x \bmod N$$

- *RSA assumption*: given N, e only, it is *hard* to compute the e^{th} root of a uniform $c \in \mathbb{Z}_N^*$

Security

- The scheme is *deterministic*
  - *Cannot* be *CPA-secure*!

Security

- The scheme is *deterministic*
  - *Cannot* be *CPA-secure*!

- *RSA assumption* only refers to hardness of computing the e^{th} roots of *uniform* c
  - c is not uniform unless m is
  - Easy to recover "small" m from c

Security

- The scheme is *deterministic*
  - *Cannot* be *CPA-secure*!

- *RSA assumption* only refers to hardness of computing the e^{th} roots of *uniform* c
  - c is not uniform unless m is
  - Easy to recover "small" m from c

- *RSA assumption* only refers to hardness of computing the e^{th} roots *in its entirety*
  - Partial information about the e^{th} root may be leaked

Security

- The scheme is *deterministic*
  - *Cannot* be *CPA-secure*!
- *RSA assumption* only refers to hardness of computing the e^{th} roots of *uniform* c
  - c is not uniform unless m is
  - Easy to recover "small" m from c
- *RSA assumption* only refers to hardness of computing the e^{th} roots *in its entirety*
  - Partial information about the e^{th} root may be leaked
- Plain RSA should *never* be used!

PKCS #1 v1.5

- Standard issued by RSA labs in 1993
- Idea: add *random padding*
  - To encrypt m, choose random r
  - $c = [(r|m)^e \bmod N]$

PKCS #1 v1.5

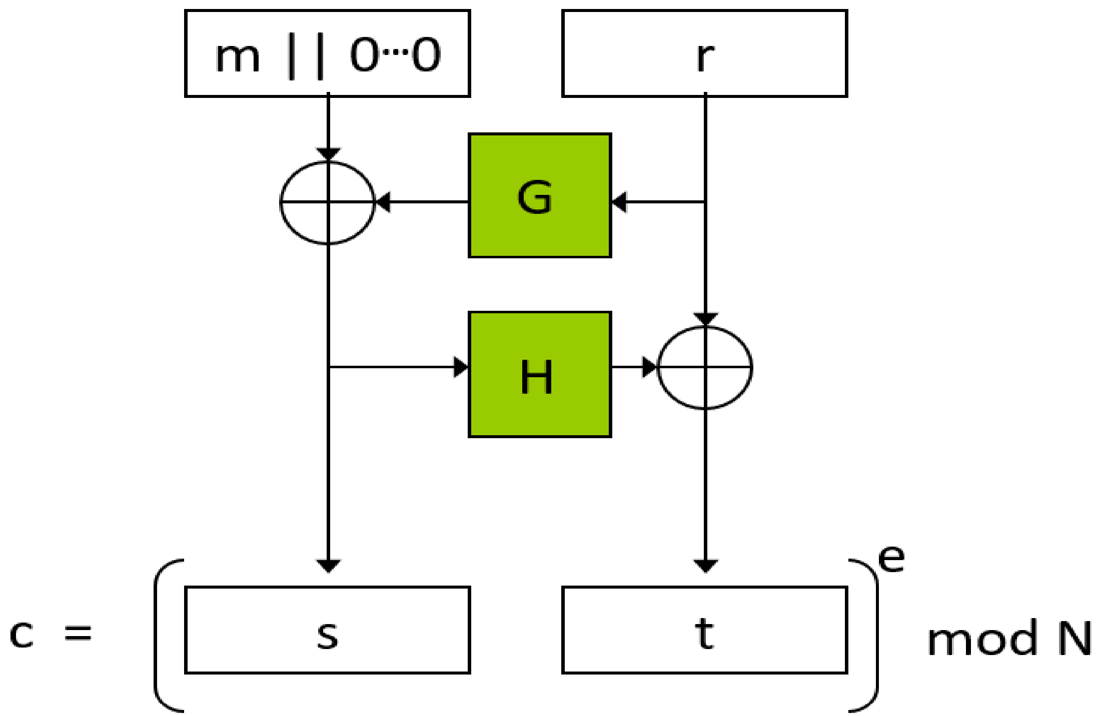
- Standard issued by RSA labs in 1993
- Idea: add *random padding*
  - To encrypt m, choose random r
  - $c = [(r|m)^e \bmod N]$
- Issues:
  - No proof of *CPA-security* (unless m is very short)
  - *Chosen-plaintext attacks* known if r is too short
  - *Chosen-ciphertext attacks* possible

PKCS #1 v2.0

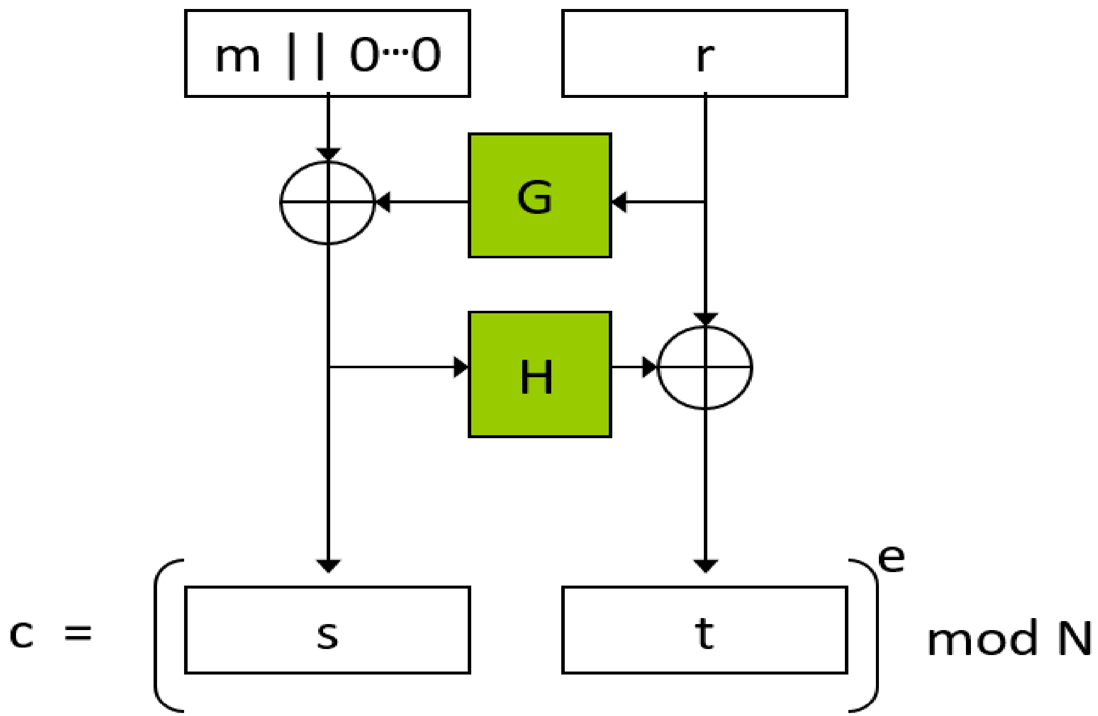
- *Optimal asymmetric encryption padding*
  - (OAEP) applied to message first

- This padding introduces *redundancy*, so that not every $c \in \mathbb{Z}_N^*$ is a valid ciphertext
  - Need to check for proper format upon decryption
  - Return error if not properly formatted

OAEP



OAEP



- RSA-OAEP can be proven *CCA-secure* under the *RSA assumption*, if G and H are modeled as *random oracles*

Chosen-ciphertext attacks

pk

$c \leftarrow \mathsf{Enc}_{pk}(m)$

c

pk, sk

c'

m'

Chosen-ciphertext attacks

- *Chosen-ciphertext attacks* are arguably even a greater concern in the public-key setting
  - Attacker might be a legitimate sender
  - Easier for attacker to obtain full decryptions of ciphertexts of its choice

Chosen-ciphertext attacks

- *Chosen-ciphertext attacks* are arguably even a greater concern in the public-key setting
  - Attacker might be a legitimate sender
  - Easier for attacker to obtain full decryptions of ciphertexts of its choice

- Related concern: *malleability*
  - I.e., given a ciphertext c that is the encryption of an unknown message m, might be possible to produce ciphertext c' that decrypts to a related message m'
  - This is also undesirable in the public-key setting

Diffie-Hellman key exchange

$$\xrightarrow{\quad G,\ q,\ g,\ h_1 \quad}$$

$$\xleftarrow{\quad h_2 \quad}$$

$(G, q, g) \leftarrow \mathcal{G}(1^n)$

$x \leftarrow \mathbb{Z}_q$

$h_1 = g^x$

$y \leftarrow \mathbb{Z}_q$

$h_2 = g^y$

$k = (h_1)^y$

El Gamal encryption

$$G, q, g, h_1 \longrightarrow$$

$$h_2 \longleftarrow$$

$$c_2 = k \cdot m \longleftarrow$$

$$(G, q, g) \leftarrow \mathcal{G}(1^n)$$
$$x \leftarrow \mathbb{Z}_q$$
$$h_1 = g^x$$

$$k = (h_2)^x$$
$$m = c_2/k$$

$$y \leftarrow \mathbb{Z}_q$$
$$h_2 = g^y$$

$$k = (h_1)^y$$

El Gamal encryption

Public key

G, q, g, h_1 $\longrightarrow$

$h_2, h_1^y \cdot m$ $\longleftarrow$

$(G, q, g) \leftarrow \mathcal{G}(1^n)$
$x \leftarrow \mathbb{Z}_q$
$h_1 = g^x$

$k = (h_2)^x$
$m = c_2/k$

$y \leftarrow \mathbb{Z}_q$
$h_2 = g^y$

$k = (h_1)^y$

El Gamal encryption

- $Gen(1^n)$
  – Run $\mathcal{G}(1^n)$ to obtain G, q, g. Choose uniform $x \in \mathbb{Z}_q$. The *public key* is (G, q, g, g^x) and the *private key* is x

El Gamal encryption

- *Gen*(1^n)
  - Run $\mathcal{G}(1^n)$ to obtain G, q, g. Choose uniform $x \in \mathbb{Z}_q$. The *public key* is (G, q, g, g^x) and the *private key* is x
- $Enc_{pk}(m)$, where $pk = (G, q, g, h)$ and $m \in G$
  - Choose uniform $y \in \mathbb{Z}_q$. The ciphertext is g^y, $h^y \cdot m$

El Gamal encryption

- $Gen(1^n)$
  - Run $\mathcal{G}(1^n)$ to obtain G, q, g. Choose uniform $x \in \mathbb{Z}_q$. The *public key* is (G, q, g, g^x) and the *private key* is x
- $Enc_{pk}(m)$, where $pk = (G, q, g, h)$ and $m \in G$
  - Choose uniform $y \in \mathbb{Z}_q$. The ciphertext is $g^y, h^y \cdot m$
- $Dec_{sk}(c_1, c_2)$
  - Output c_2 / c_1^x

Security

- If the *DDH assumption* is hard for $\mathcal{G}$, then the El Gamal encryption scheme is *CPA-secure*
  – Follows from security of Diffie-Hellman key exchange, or can be proved directly

Security

- If the *DDH assumption* is hard for $\mathcal{G}$, then the El Gamal encryption scheme is *CPA-secure*
  – Follows from security of Diffie-Hellman key exchange, or can be proved directly
- *Dlog assumption* alone is not enough here

In practice

- Parameters G, q, g are standardized and shared

In practice

- Parameters G, q, g are standardized and shared
- Inconvenient to treat message as group element
  - Use *key derivation* to derive a key k instead, and use k to encrypt the message
  - I.e., ciphertext is $g^y, Enc'_k(m)$, where $k = H(h^y)$

In practice

- Parameters G, q, g are standardized and shared
- Inconvenient to treat message as group element
  - Use *key derivation* to derive a key k instead, and use k to encrypt the message
  - I.e., ciphertext is $g^y, Enc'_k(m)$, where $k = H(h^y)$
- El Gamal encryption is not secure against *chosen-ciphertext attacks*
  - Follows from the fact that it is *malleable*

Chosen-ciphertext attacks

- El Gamal encryption is not secure against *chosen-ciphertest attacks*
  - Follows from the fact that it is *malleable*

Chosen-ciphertext attacks

- El Gamal encryption is not secure against *chosen-ciphertest attacks*
  - Follows from the fact that it is *malleable*

- Given ciphertext c_1, c_2, transform it to obtain the ciphertext $c_1, c_2' = c_1, \alpha \cdot c_2$ for arbitrary α
  - Since $c_1, c_2 = g^y, h^y \cdot m$,
    we have $c_1, c_2' = g^y, h^y \cdot (\alpha m)$

Chosen-ciphertext attacks

- El Gamal encryption is not secure against *chosen-ciphertest attacks*
  – Follows from the fact that it is *malleable*

- Given ciphertext c_1, c_2, transform it to obtain the ciphertext $c_1, c_2' = c_1, \alpha \cdot c_2$ for arbitrary α
  – Since $\quad c_1, c_2 = g^y, h^y \cdot m,$
  we have $\quad c_1, c_2' = g^y, h^y \cdot (\alpha m)$
  – I.e., encryption of m becomes an encryption of αm!

Chosen-ciphertext attacks security

- Use *key derivation* coupled with *CCA-secure* private-key encryption scheme
  - I.e., ciphertext is

$$g^y, Enc'_k(m),$$

where $k = H(h^y)$ and Enc' is a CCA-secure scheme

- Can be proved *CCA-secure* under appropriate assumptions, if H is modeled as a random oracle.

CPA-secure public key encryption

- Constructing *CCA-secure* public key encryption is more challenging than the private key case.

CPA-secure public key encryption

- Constructing *CCA-secure* public key encryption is more challenging than the private key case.

- **Construction 13.1**: Construct an encryption scheme as follows:
  - Let $H : \{0,1\}^* \to \{0,1\}^n$ be a random oracle and $\{(f, f^{-1})\}$ be a collection of *trapdoor permutations*. The public key of the scheme will be $f(\cdot)$ while the private key is $f^{-1}(\cdot)$.
  - To encrypt $x \in \{0,1\}^n$, choose $r \leftarrow_R \{0,1\}^n$ and compute $f(r), H(r) \oplus x$.
  - To decrypt y, z, compute $r = f^{-1}(y)$ and let $x = H(r) \oplus z$.

CPA-secure public key encryption

- Constructing *CCA-secure* public key encryption is more challenging than the private key case.

- **Construction 13.1**: Construct an encryption scheme as follows:
  - Let $H : \{0,1\}^* \to \{0,1\}^n$ be a random oracle and $\{(f, f^{-1})\}$ be a collection of *trapdoor permutations*. The public key of the scheme will be $f(\cdot)$ while the private key is $f^{-1}(\cdot)$.
  - To encrypt $x \in \{0,1\}^n$, choose $r \leftarrow_R \{0,1\}^n$ and compute $f(r), H(r) \oplus x$.
  - To decrypt y, z, compute $r = f^{-1}(y)$ and let $x = H(r) \oplus z$.

- **Theorem 13.2** The above scheme is *CPA-secure* in the random oracle model.

Recall

- **Construction 13.1**: Construct an encryption scheme as follows:
  - Let $H : \{0,1\}^* \to \{0,1\}^n$ be a random oracle and $\{(f, f^{-1})\}$ be a collection of *trapdoor permutations*. The public key of the scheme will be $f(\cdot)$ while the private key is $f^{-1}(\cdot)$.
  - To encrypt $x \in \{0,1\}^n$, choose $r \leftarrow_R \{0,1\}^n$ and compute $f(r), H(r) \oplus x$.
  - To decrypt y, z, compute $r = f^{-1}(y)$ and let $x = H(r) \oplus z$.

Recall

- **Construction 13.1**: Construct an encryption scheme as follows:
  - Let $H : \{0,1\}^* \to \{0,1\}^n$ be a random oracle and $\{(f, f^{-1})\}$ be a collection of *trapdoor permutations*. The public key of the scheme will be $f(\cdot)$ while the private key is $f^{-1}(\cdot)$.
  - To encrypt $x \in \{0,1\}^n$, choose $r \leftarrow_R \{0,1\}^n$ and compute $f(r), H(r) \oplus x$.
  - To decrypt y, z, compute $r = f^{-1}(y)$ and let $x = H(r) \oplus z$.

- **Theorem 5.1** (CPA security from PRFs)
  Suppose that F is a length-preserving, keyed PRF, then the following is a *CPA-secure encryption scheme*:

$$Enc_k(m) = \langle r, F_k(r) \oplus m \rangle$$
$$Dec_k(c_1, c_2) = c_2 \oplus F_k(c_1)$$

- **Theorem 13.2** The above scheme is *CPA-secure* in the random oracle model.

  Proof. For public key encryption, CPA security means that an adversary A that gets as input the encryption key $f(\cdot)$ cannot distinguish $Enc(x_1)$ and $Enc(x_2)$ for every x_1, x_2, since encryption is public. In the random oracle model, A has access to the random oracle $H(\cdot)$.

CPA-secure public key encryption

- **Theorem 13.2** The above scheme is *CPA-secure* in the random oracle model.

  Proof. For public key encryption, CPA security means that an adversary A that gets as input the encryption key $f(\cdot)$ cannot distinguish $Enc(x_1)$ and $Enc(x_2)$ for every x_1, x_2, since encryption is public. In the random oracle model, A has access to the random oracle $H(\cdot)$.

  Denote the ciphertext A gets as y^*, z^*, where $y^* = f(r^*)$ and $z^* = H(r^*) \oplus x^*$.

CPA-secure public key encryption

- **Theorem 13.2** The above scheme is *CPA-secure* in the random oracle model.

Proof. For public key encryption, CPA security means that an adversary A that gets as input the encryption key $f(\cdot)$ cannot distinguish $Enc(x_1)$ and $Enc(x_2)$ for every x_1, x_2, since encryption is public. In the random oracle model, A has access to the random oracle $H(\cdot)$.

Denote the ciphertext A gets as y^*, z^*, where $y^* = f(r^*)$ and $z^* = H(r^*) \oplus x^*$.

Claim 13.2.1 The probability that A queries r^* of its oracle $H(\cdot)$ is *negl.*.

CPA Secure Public Key Encryption

- **Claim 13.2.1** The probability that A queries r^* of its oracle $H(\cdot)$ is *negl.*.

  Proof. Consider the following experiment: Instead of giving $z^* = H(r^*) \oplus x^*$, we give A the string $z^* = u \oplus x^*$ where u is a uniform element.

- **Claim 13.2.1** The probability that A queries r^* of its oracle $H(\cdot)$ is *negl.*.

  Proof. Consider the following experiment: Instead of giving $z^* = H(r^*) \oplus x^*$, we give A the string $z^* = u \oplus x^*$ where u is a uniform element.

  The only way A could tell apart the two cases is if he queries r^* to H and sees a different answer from u. But then we already "lost". The probability that A queries r^* in the experiment is the same as the probability that it queries r^* in the actual attack.

- **Claim 13.2.1** The probability that A queries r^* of its oracle $H(\cdot)$ is *negl.*.

  Proof. Consider the following experiment: Instead of giving $z^* = H(r^*) \oplus x^*$, we give A the string $z^* = u \oplus x^*$ where u is a uniform element.

  The only way A could tell apart the two cases is if he queries r^* to H and sees a different answer from u. But then we already "lost". The probability that A queries r^* in the experiment is the same as the probability that it queries r^* in the actual attack.

  However, in this experiment, the only infomation A gets about r^* is $f(r^*)$. Thus, if it queries $H(\cdot)$ the value r^*, then it inverted the *trapdoor permutation*, which is almost impossible!

CPA Secure Public Key Encryption

- **Theorem 13.2** The above scheme is *CPA-secure* in the random oracle model.

  Claim 13.2.1 The probability that A queries r^* of its oracle $H(\cdot)$ is *negl.*.

- **Theorem 13.2** The above scheme is *CPA-secure* in the random oracle model.

  Claim 13.2.1 The probability that A queries r^* of its oracle $H(\cdot)$ is *negl.*.

  Proof cont'. Claim 13.2.1 means that we can ignore the probability that A queried r^* and hence we can assume that $z^* = u \oplus x^*$, where u is chosen independently at random.

- **Theorem 13.2** The above scheme is *CPA-secure* in the random oracle model.

  Claim 13.2.1 The probability that A queries r^* of its oracle $H(\cdot)$ is *negl.*.

  Proof cont'. Claim 13.2.1 means that we can ignore the probability that A queried r^* and hence we can assume that $z^* = u \oplus x^*$, where u is chosen independently at random.

  However, A gets no information about x^* and will not be able to guess if it is equal to x_1 or x_2 with probability greater than $1/2$.

CCA Secure Public Key Encryption

- **Construction 13.3**: Construct an encryption scheme (using two independent random oracles) as follows:
  - Let $H,\ H' : \{0,1\}^* \to \{0,1\}^n$ be two independent random oracles and $\{(f, f^{-1})\}$ be a collection of *trapdoor permutations*. The public key of the scheme will be $f(\cdot)$ while the private key is $f^{-1}(\cdot)$.
  - To encrypt $x \in \{0,1\}^n$, choose $r \leftarrow_R \{0,1\}^n$ and compute $f(r), H(r) \oplus x, H'(x,r)$.
  - To decrypt y, z, w, compute $r = f^{-1}(y)$ and let $x = H(r) \oplus z$. Then, check that $w = H'(x,r)$: if so then return x, otherwise return $\perp$.

CCA Secure Public Key Encryption

- **Construction 13.3**: Construct an encryption scheme (using two independent random oracles) as follows:
  - Let $H,\ H' : \{0,1\}^* \to \{0,1\}^n$ be two independent random oracles and $\{(f, f^{-1})\}$ be a collection of *trapdoor permutations*. The public key of the scheme will be $f(\cdot)$ while the private key is $f^{-1}(\cdot)$.
  - To encrypt $x \in \{0,1\}^n$, choose $r \leftarrow_R \{0,1\}^n$ and compute $f(r), H(r) \oplus x, H'(x, r)$.
  - To decrypt y, z, w, compute $r = f^{-1}(y)$ and let $x = H(r) \oplus z$. Then, check that $w = H'(x, r)$: if so then return x, otherwise return $\perp$.

Theorem 13.4 The above scheme is *CCA-secure* in the random oracle model.

CCA Secure Public Key Encryption

- **Theorem 13.4** The above scheme is *CCA-secure* in the random oracle model.

  Proof. Let A be the algorithm in a CCA attack against the scheme. Denote by y^*, z^*, w^* the challenge ciphertext A gets, where $y^* = f(r^*)$, $z^* = H(r^*) \oplus x^*$ and $w^* = H'(x^*, r^*)$.

CCA Secure Public Key Encryption

- **Theorem 13.4** The above scheme is *CCA-secure* in the random oracle model.

  Proof. Let A be the algorithm in a CCA attack against the scheme. Denote by y^*, z^*, w^* the challenge ciphertext A gets, where $y^* = f(r^*)$, $z^* = H(r^*) \oplus x^*$ and $w^* = H'(x^*, r^*)$.

  Since H is a random oracle, we can always assume that no one (the sender, receiver, or A) can find two pairs x, r and x', r' such that $x||r \neq x'||r'$, but $H'(x, r) = H'(x', r')$.

CCA Secure Public Key Encryption

- **Theorem 13.4** The above scheme is *CCA-secure* in the random oracle model.

  Proof. Let A be the algorithm in a CCA attack against the scheme. Denote by y^*, z^*, w^* the challenge ciphertext A gets, where $y^* = f(r^*)$, $z^* = H(r^*) \oplus x^*$ and $w^* = H'(x^*, r^*)$.

  Since H is a random oracle, we can always assume that no one (the sender, receiver, or A) can find two pairs x, r and x', r' such that $x||r \neq x'||r'$, but $H'(x, r) = H'(x', r')$.

  At each step i of the attack, for every string $w \in \{0,1\}^n$, we define ${H'}_i^{-1}(w)$ as: if the oracle H was queried before with x, r and returned w, then ${H'}_i^{-1}(w) = (x, r)$; otherwise, ${H'}_i^{-1}(w) = \perp$.

- **Theorem 13.4** The above scheme is *CCA-secure* in the random oracle model.

Proof. Let A be the algorithm in a CCA attack against the scheme. Denote by y^*, z^*, w^* the challenge ciphertext A gets, where $y^* = f(r^*)$, $z^* = H(r^*) \oplus x^*$ and $w^* = H'(x^*, r^*)$.

Since H is a random oracle, we can always assume that no one (the sender, receiver, or A) can find two pairs x, r and x', r' such that $x||r \neq x'||r'$, but $H'(x, r) = H'(x', r')$.

At each step i of the attack, for every string $w \in \{0, 1\}^n$, we define ${H'}_i^{-1}(w)$ as: if the oracle H was queried before with x, r and returned w, then ${H'}_i^{-1}(w) = (x, r)$; otherwise, ${H'}_i^{-1}(w) = \perp$.
Observation: a pair x, r completely determines a ciphertext y, z, w, and y, z completely determine x, r.

- **Theorem 13.4** The above scheme is *CCA-secure* in the random oracle model.
  Proof cont'. Consider the experiment: at step i, we answer a query y, z, w of A in the following way: if ${H'}^{-1}(w)$ is equal to some x, r that determine y, z, w, then return x; otherwise, return $\perp$.

- **Theorem 13.4** The above scheme is *CCA-secure* in the random oracle model.
  Proof cont'. Consider the experiment: at step i, we answer a query y, z, w of A in the following way: if $H'^{-1}(w)$ is equal to some x, r that determine y, z, w, then return x; otherwise, return $\perp$.
  The difference between this oracle and the real decryption oracle is that we may answer $\perp$ when the real one would give an actural answer. However, we claim that A will not be able to tell apart the difference with non-negl. probability.

CCA Secure Public Key Encryption

- **Theorem 13.4** The above scheme is *CCA-secure* in the random oracle model.

  Proof cont'. Consider the experiment: at step i, we answer a query y, z, w of A in the following way: if $H'^{-1}(w)$ is equal to some x, r that determine y, z, w, then return x; otherwise, return $\perp$.
  The difference between this oracle and the real decryption oracle is that we may answer $\perp$ when the real one would give an actural answer. However, we claim that A will not be able to tell apart the difference with non-negl. probability.

  The only difference happens if A managed to ask the oracle a query y, z, w satisfying the following:
  - $w \neq w^*$.
  - w was not returned as the answer of any previous query x, r to $H'(\cdot)$ by A.
  - If we let x, r be the values determined by y, z, then $H'(x, r) = w$. However, since (x, r) was not asked before, the probability that this happens is only 2^{-n}.

CCA Secure Public Key Encryption

- **Theorem 13.4** The above scheme is *CCA-secure* in the random oracle model.
  Proof cont'. Consider the experiment: at step i, we answer a query y, z, w of A in the following way: if ${H'}^{-1}(w)$ is equal to some x, r that determine y, z, w, then return x; otherwise, return $\perp$.
  The difference between this oracle and the real decryption oracle is that we may answer $\perp$ when the real one would give an actural answer. However, we claim that A will not be able to tell apart the difference with non-negl. probability.

  Basically A has no use for the decryption box and hence it would be sufficient to prove that the scheme is just *CPA-secure*.

Review

- What you've leaned:
  - ◇ Foundations and principles of the science
  - ◇ Basic primitives and components
  - ◇ Definitions and proofs of security
  - ◇ High-level applications

Review

- What you've leaned:
  - ◇ Foundations and principles of the science
  - ◇ Basic primitives and components
  - ◇ Definitions and proofs of security
  - ◇ High-level applications

Perfect secrecy (*one-time pad*) (Def. 1.5 - Thm. 1.9)

ϵ-*statistical security* (Def. 2.2)

computational security (Def. 3.1)

Review

- What you've leaned:
  - ◇ Foundations and principles of the science
  - ◇ Basic primitives and components
  - ◇ Definitions and proofs of security
  - ◇ High-level applications

Perfect secrecy (*one-time pad*) (Def. 1.5 - Thm. 1.9)

ϵ-*statistical security* (Def. 2.2)

computational security (Def. 3.1)

PRG, *pseudorandomness* (Def. 3.2)

pseudo one-time pad (Thm. 3.3)

Review

- *What you've leaned*:
  - Foundations and principles of the science
  - Basic primitives and components
  - Definitions and proofs of security
  - High-level applications

Perfect secrecy (*one-time pad*) (Def. 1.5 - Thm. 1.9)

ϵ-statistical security (Def. 2.2)

computational security (Def. 3.1)

PRG, *pseudorandomness* (Def. 3.2)

pseudo one-time pad (Thm. 3.3)

multiple-message indistinguishable (Def. 3.4)

Review

- PRG $\rightarrow$ PRF $\rightarrow$ PRP (*block cipher*) (Def. 4.2, 4.3)

Review

- PRG $\rightarrow$ PRF $\rightarrow$ PRP (*block cipher*) (Def. 4.2, 4.3)

  PRF $\rightarrow$ *CPA security* (Def. 4.1, Thm. 5.1, 5.2)

  PRF $\rightarrow$ *CMA-secure MAC* (Def. 6.2, Thm. 6.3)

Review

- PRG $\rightarrow$ PRF $\rightarrow$ PRP (*block cipher*) (Def. 4.2, 4.3)

  PRF $\rightarrow$ *CPA security* (Def. 4.1, Thm. 5.1, 5.2)

  PRF $\rightarrow$ *CMA-secure MAC* (Def. 6.2, Thm. 6.3)

  EtA $\rightarrow$ *CCA security* (Def. 6.1, lec08)

Review

- PRG $\rightarrow$ PRF $\rightarrow$ PRP (*block cipher*) (Def. 4.2, 4.3)

  PRF $\rightarrow$ *CPA security* (Def. 4.1, Thm. 5.1, 5.2)

  PRF $\rightarrow$ *CMA-secure MAC* (Def. 6.2, Thm. 6.3)

  EtA $\rightarrow$ *CCA security* (Def. 6.1, lec08)

  Hash function (Def. 7.1, Thm.8.1, 8.2, 14.2)

Review

- PRG $\rightarrow$ PRF $\rightarrow$ PRP (*block cipher*) (Def. 4.2, 4.3)

  PRF $\rightarrow$ *CPA security* (Def. 4.1, Thm. 5.1, 5.2)

  PRF $\rightarrow$ *CMA-secure MAC* (Def. 6.2, Thm. 6.3)

  EtA $\rightarrow$ *CCA security* (Def. 6.1, lec08)

  Hash function (Def. 7.1, Thm.8.1, 8.2, 14.2)

  Stream/Block ciphers (lec09, lec10)

Review

- PRG $\rightarrow$ PRF $\rightarrow$ PRP (*block cipher*) (Def. 4.2, 4.3)

  PRF $\rightarrow$ *CPA security* (Def. 4.1, Thm. 5.1, 5.2)

  PRF $\rightarrow$ *CMA-secure MAC* (Def. 6.2, Thm. 6.3)

  EtA $\rightarrow$ *CCA security* (Def. 6.1, lec08)

  Hash function (Def. 7.1, Thm.8.1, 8.2, 14.2)

  Stream/Block ciphers (lec09, lec10)

  Math fundamentals (hard/trapdoor functions, lec11, lec12)

Review

- PRG $\rightarrow$ PRF $\rightarrow$ PRP (*block cipher*) (Def. 4.2, 4.3)

  PRF $\rightarrow$ *CPA security* (Def. 4.1, Thm. 5.1, 5.2)

  PRF $\rightarrow$ *CMA-secure MAC* (Def. 6.2, Thm. 6.3)

  EtA $\rightarrow$ *CCA security* (Def. 6.1, lec08)

  Hash function (Def. 7.1, Thm.8.1, 8.2, 14.2)

  Stream/Block ciphers (lec09, lec10)

  Math fundamentals (hard/trapdoor functions, lec11, lec12)

  Public key encryption (Def. 12.1 - Thm 12.5)

Review

- PRG → PRF → PRP (*block cipher*) (Def. 4.2, 4.3)
  PRF → *CPA security* (Def. 4.1, Thm. 5.1, 5.2)
  PRF → *CMA-secure MAC* (Def. 6.2, Thm. 6.3)
  EtA → *CCA security* (Def. 6.1, lec08)
  Hash function (Def. 7.1, Thm.8.1, 8.2, 14.2)
  Stream/Block ciphers (lec09, lec10)
  Math fundamentals (hard/trapdoor functions, lec11, lec12)
  Public key encryption (Def. 12.1 - Thm 12.5)
  CPA security (Def. 13.1, Thm. 13.2)
  CCA security (Def. 13.3, Thm. 13.4)

Review

- Rabin's trapdoor function, signature

  RSA trapdoor function, signature

Next Lecture

- digital signature ...