Modern key-exchange protocols

- Security against passive eavesdroppers is insufficient
- Want *authenticated* key exchange
  - This requires some form of setup in advance
- Modern key-exchange protocols provide this
  - We will return to this later

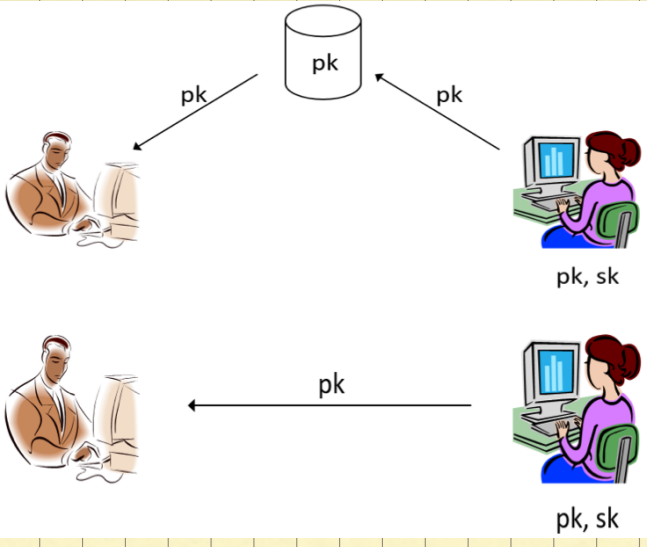
The public-key setting

- 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*

公钥公开
私钥保密

非对称加密

Public-key distribution I.II



还有identity-based encryption:
直接用对方的identity作为公钥.

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

若攻击者是active的,还需integrity保证密钥未被篡改.

Drawbacks of private-key crypto

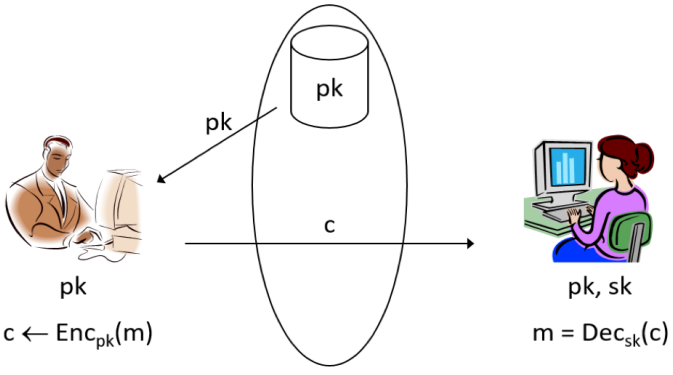
- Key distribution
  - Public keys can be distributed over *public* (but *authenticated*) channels!
- Key management in large systems of N users
  - Each user stores 1 private key and $N-1$ public keys; only N keys overall
  - Public keys can be stored in a central directory
- Applicability in "open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

公钥加密需要更长的密钥
对称加密效率更高

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 $\perp$

  For all m and pk, sk output by *Gen*,

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

对公钥加密，一开始就必须为CPA-security（因为pk是公开的，相当于已经有了Encryption Oracle）

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

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

⇒ No *perfectly secret* public-key encryption

⇒ No *deterministic* public-key encryption can be CPA-secure

⇒ CPA-security implies security for encryption multiple messages as in the private-key case

公钥加密必须非完美安全以及非确定性的。

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.

假设 $m_0 \neq m_1$.

A: 输入 (pk, c)

for each possible choice of random bits ← 这个遍历可能不是多项式时间的 (但完美安全并未限制多项式时间)

set $c_0 = Enc_{pk}(m_0)$

if $c = c_0$, output 0, end

Output 1, end.

注意: ① A肯定能停止 (有限的 random bits)

② A可能不是 PPT 的, 但完美安全不要求 PPT.

算法正确性: ① 若 $c = Enc_{pk}(m_0)$, 则 A 输出 0

② 若 $c = Enc_{pk}(m_1)$, 则 A 输出 1 (c不可能既是 $Enc_{pk}(m_0)$ 也是 $Enc_{pk}(m_1)$)

故 $\Pr[A(pk,c)=0 | c \leftarrow Enc_{pk}(m_0)] = 1$

$\Pr[A(pk,c)=0 | c \leftarrow Enc_{pk}(m_1)] = 0$ (或者反过来)

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 $c \in \mathbb{Z}_N^*$

求出 $c^{\frac{1}{e}} \bmod N$ 是困难的
(也即 $x \bmod N$)

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!

需要增加随机性

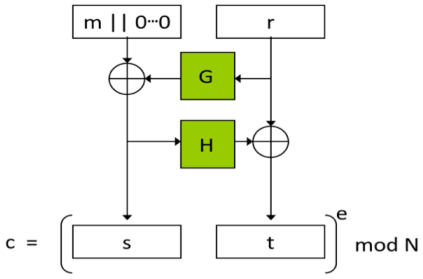
PKCS #1 v1.5

- 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



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

$r \xleftarrow{R} \{0,1\}^{k_0}$

$|m| = \ell$

$|0\ldots0| = k_1$

$G: \{0,1\}^{k_0} \to \{0,1\}^{\ell+k_1}$

$H: \{0,1\}^{\ell+k_1} \to \{0,1\}^{k_0}$

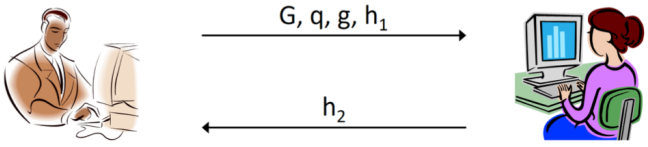
$s = (m \| 0\ldots0) \oplus G(r)$

$t = r \oplus H(s)$

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

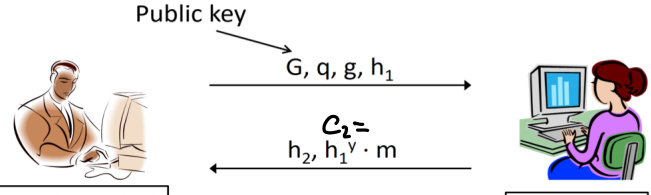


$(G, q, g) \leftarrow \mathcal{G}(1^n)$
$x \leftarrow \mathbb{Z}_q$
$h_1 = g^x$

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

$k = (h_1)^y$

El Gamal encryption



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

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

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

$k = (h_1)^y$

- $Gen(1^n)$
  - 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

$c_1 = g^y$, $c_2 = h^y \cdot m = g^{xy} m$

$c_2 / c_1^x = g^{xy} m / g^{xy} = m$

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

El Gamal 需要更强的DDH假设，仅有Dlog不够

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*

先生成一个 key k 再加密

El Gamal 不是CCA安全的.

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 $c_1, c'_2 = g^y, h^y \cdot (\alpha m)$
  - 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.

原本密文 $(c_1, c_2) = (g^y, h^x \cdot m)$

CCA密文 $(c_1, c_2) = (g^y, Enc'_k(m))$

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.

$H(r) \oplus x$ 是 malleable 的，故非CCA-secure

Recall

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

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.*.

  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!

  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 $\perp$.

增加 $H'(x,r)$ 来验证，确保不能延展.

- **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.

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}.

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