In the previous lecture

- Definition of a (perfectly secure) **symmetric cryptosystem** (but how do you transmit the key?)
- The one-time pad, Shannon's theorem
- Definitions of an efficient adversary, and indistinguishability notions

Introduction to Cryptography
Part II: Public-Key Encryption – RSA

André Schrottenloher

Inria Rennes
Team CAPSULE

1. Public-Key Encryption
2. Prime Numbers and Factoring
3. Textbook RSA
4. Padded RSA

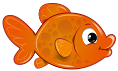
Public-Key Encryption

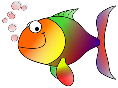
Asymmetric encryption

A PKE scheme is a triple of PPT algorithms $\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$:

$$\begin{cases} \mathsf{KeyGen}: & 1^n & \mapsto & \mathsf{sk}, \mathsf{pk} \\ \mathsf{Enc}: & \mathsf{m}, \mathsf{pk} & \mapsto & \mathsf{c} \\ \mathsf{Dec}: & \mathsf{c}, \mathsf{sk} & \mapsto & \mathsf{m} \end{cases} \tag{1}$$

such that $\forall \mathsf{m}$, $\mathsf{Dec}(\mathsf{sk}, (\mathsf{Enc}(\mathsf{pk}, \mathsf{m}), \mathsf{m})) = \mathsf{m}$.





$\mathsf{sk}, \mathsf{pk} = \mathsf{KeyGen}(1^n)$

$\xrightarrow{\mathsf{pk}}$

$\mathsf{c} = \mathsf{Enc}(\mathsf{m}, \mathsf{pk})$

$\xleftarrow{\mathsf{c}}$

$\mathsf{m} = \mathsf{Dec}(\mathsf{c}, \mathsf{sk})$

Color code: **not secret**, **secret**, no color = public parameter.

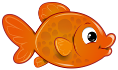
Security of PKE

- "The adversary cannot learn anything on the ciphertext from the plaintext" = perfect security (One-time Pad).
- By restricting to PPT adversaries we get the notion of **semantic security**. However it's hard to prove / use in practice.
- Instead we use **ciphertext indistinguishability**, which is equivalent and easier to use.

IND-CPA

The IND-CPA security game for PKE is defined as follows.

- **Initialization :** $\mathcal{C}$ chooses $b \hookleftarrow U(0,1)$ and keys $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}(1^n)$, sends pk to $\mathcal{A}$
- **Find stage :** $\mathcal{A}$ chooses messages m_0, m_1 and sends to $\mathcal{C}$, who returns $c^* = \mathrm{Enc}(\mathrm{pk}, m_b)$ (the **challenge ciphertext**
- **Guess stage :** $\mathcal{A}$ computes b' and wins the game if $b = b'$.



Choose $b \hookleftarrow U(0,1)$

Choose $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}(1^n)$

$\xrightarrow{\quad \mathrm{pk} \quad}$

$\xleftarrow{\quad m_0, m_1 \quad}$

$\xrightarrow{\quad c^* = \mathrm{Enc}(\mathrm{pk}, m_b) \quad}$

Return b'

IND-CPA (ctd.)

The **advantage** of $\mathcal{A}$ is:

$$\mathrm{Adv}^{CPA}(\mathcal{A}) = \left| \Pr\left[\mathcal{A} \text{ wins}\right] - \frac{1}{2} \right| \ .$$

If the advantage of any PPT adversary is negligible, then the cipher is said to be **IND-CPA secure.**

IND-CPA (ctd.)

The **advantage** of $\mathcal{A}$ is:

$$\mathrm{Adv}^{CPA}(\mathcal{A}) = \left|\Pr\left[\mathcal{A} \text{ wins}\right] - \frac{1}{2}\right| \ .$$

If the advantage of any PPT adversary is negligible, then the cipher is said to be **IND-CPA secure.**

Note that:

- The adversary may encrypt at will during the game (since they have the public key) $\implies$ "chosen-plaintext"
- The encryption **must** be probabilistic, otherwise there is a trivial attack

IND-CCA

- IND-CCA is a stronger notion: IND-CPA + decryption queries.
- Decryption queries should not allow the adversary to win trivially (e.g., decrypt c^*)

IND-CCA

- IND-CCA is a stronger notion: IND-CPA + decryption queries.
- Decryption queries should not allow the adversary to win trivially (e.g., decrypt c^*)

The IND-CCA security game is defined like the IND-CPA game, during which $\mathcal{A}$ can additionally perform **decryption queries**. They are answered as follows:

- $\mathcal{A}$ chooses a ciphertext c and sends c to $\mathcal{C}$
- If $c \neq c^*$, $\mathcal{C}$ returns $\mathrm{Dec}(\mathsf{sk}, c)$
- Otherwise $\mathcal{C}$ returns $\perp$

IND-CCA (ctd.)

There are two variants:

- **IND-CCA1** ("non-adaptive"): queries only in the "find stage" (before c^* is known)
- **IND-CCA2** ("adaptive"): queries at any point

The advantage of the adversary is defined by:

$$\mathrm{Adv}^{CCA}(\mathcal{A}) = \left| \Pr\left[\mathcal{A} \ \mathit{Wins}\right] - \frac{1}{2} \right| \ .$$

If the advantage of any PPT adversary is negligible, then the cipher is said to be IND-CCA(1,2) secure.

Prime Numbers and Factoring

Prime numbers and how to find them

Prime number theorem

There are $\mathcal{O}(2^n/n)$ prime numbers with n bits.

$\implies$ if you select a random n-bit integer, it's prime with probability $\mathcal{O}(1/n)$.

Fermat's little theorem

If p is prime, for any $a < p$, $a^{p-1} = 1 \pmod{p}$.

$\implies$ Fermat primality test: pick a random a and check if this condition holds. For most non-primes, the condition breaks with constant probability.

- However there are bad cases, so we use instead the Miller-Rabin primality test: if p is non-prime, the condition breaks with probability 3/4.
- Repeat *ad lib* until you're satisfied with the probability of success

Factoring

- Multiplying integers ($P, Q \to PQ$) is easy
- Factoring ($PQ \to P, Q$) is **hard**

- The best algorithm for factoring has **subexponential** complexity (GNFS):

$$\exp\left[\left((64/9)^{1/3} + o(1)\right)(\log n)^{1/3}(\log\log n)^{2/3}\right] \simeq 2^{\mathcal{O}\left(n^{1/3}\right)}$$

Some arithmetic

We work in the group $\mathbb{Z}_N$, and $\mathbb{Z}_N^*$ is the (multiplicative) subgroup of invertible elements (integers $< N$ prime with N).

Euler's totient function

$$\phi(N) = |\mathbb{Z}_N^*|$$

Properties:

$$\begin{aligned} \phi(p) &= p - 1 \text{ for } p \text{ prime} \\ \phi(p_1 \cdots p_\ell) &= \phi(p_1) \cdots \phi(p_\ell) \text{ for } p_1, \ldots, p_\ell \text{ coprime} \\ \phi(p^e) &= p^{e-1}(p-1) \text{ for } p \text{ prime} \\ \phi(pq) &= (p-1)(q-1) \text{ for } p, q \text{ distinct primes} \end{aligned}$$

Some arithmetic (ctd.)

Lagrange's theorem

If H is a subgroup of the group G, then the order of H divides the order of G.

Corollary

In any group $\mathrm{G}, \cdot$ of order n, for any $a \in \mathrm{G}$, $a^n = 1$.

Consequence: Fermat's little theorem

For any N, for any a prime with N, $a^{\phi(N)} = 1 \pmod{N}$.

Some arithmetic (ctd.)

Chinese remainder theorem (CRT)

Let $N = PQ$ where P, Q are coprime:

$$\begin{cases} \mathbb{Z}_N \simeq \mathbb{Z}_P \times \mathbb{Z}_Q \\ \mathbb{Z}_N^* \simeq \mathbb{Z}_P^* \times \mathbb{Z}_Q^* \end{cases}$$

The function $f(x) = (x \mod P, x \mod Q)$ is such an isomorphism.

Some arithmetic (ctd.)

Chinese remainder theorem (CRT)

Let $N = PQ$ where P, Q are coprime:

$$\begin{cases} \mathbb{Z}_N \simeq \mathbb{Z}_P \times \mathbb{Z}_Q \\ \mathbb{Z}_N^* \simeq \mathbb{Z}_P^* \times \mathbb{Z}_Q^* \end{cases}$$

The function $f(x) = (x \mod P, x \mod Q)$ is such an isomorphism.

If P, Q are known, the inverse of f can be computed in polynomial time.

- Use Euclide's algorithm to find x, y such that $xP + yQ = 1$.
- Given $(a, b) \in \mathbb{Z}_P \times \mathbb{Z}_Q$, compute: $c = yQa + xPb \pmod N$
- Check that $c \pmod P = yQa \pmod P = a$ and $c \pmod Q = xPb \pmod Q = b$.

Textbook RSA

Constructing a PKE

The Holy Grail of public-key encryption is a **trapdoor one-way function**.

- **One-way**: a function f that is easy to compute ($x \rightarrow f(x)$), but difficult to invert
- **Trapdoor**: the knowledge of some additional information should make this problem easy again

Constructing a PKE

The Holy Grail of public-key encryption is a **trapdoor one-way function**.

- **One-way**: a function f that is easy to compute ($x \rightarrow f(x)$), but difficult to invert
- **Trapdoor**: the knowledge of some additional information should make this problem easy again

RSA is **the** most well-known cryptosystem, and still one of the most used.

Textbook RSA

We work in $\mathbb{Z}_N^*$.

KeyGen:

- Choose P, Q prime, $N = PQ$
- Choose e prime with $\phi(N)$, compute d s.t. $ed = 1 \pmod{\phi(N)}$.
- $\mathrm{sk} = d, \mathrm{pk} = (N, e)$

Enc ($\mathrm{m} \in \mathbb{Z}_N^*$):

- $\mathrm{c} = \mathrm{m}^e$

Dec:

- $\mathrm{m} = \mathrm{c}^{\mathrm{d}}$.

Correctness:

$$(m^e)^d = m^{ed} = m \pmod{N} .$$

Wait... is this efficient?

KeyGen: in time $\mathrm{poly}(n)$, we can generate probable primes (probability of failure $= 2^{-n}$) with Miller-Rabin.

Enc and Dec perform **modular exponentiation**.

Let $e = e_0 + 2e_1 + \ldots + 2^{n-1}e_{n-1}$:

$$m^e = m^{e_0 + 2e_1 + \ldots + 2^{n-1}e_{n-1}} = m^{e_0 + 2(e_1 + 2(e_2 + \ldots)\ldots)}$$

- Compute $m^{e_{n-1}}$
- Square: $m^{2e_{n-1}}$
- Multiply: $m^{e_{n-2} + 2e_{n-1}}$
- Square: $m^{2e_{n-2} + 2^2 e_{n-1}}$
- $\ldots \implies \mathcal{O}(n)$ modular operations

DO NOT USE this algorithm in actual software.

RSA problem

The **RSA problem** is:

- Given $x^e \pmod{N}$, with public parameters (e, N), find x

The **RSA assumption** is that the problem is difficult.

Lemma

Factorisation is harder than RSA: if there is a PPT algorithm solving the factorisation problem, there is a PPT algorithm solving the RSA problem.

Knowing P and Q, we can compute $\phi(N)$, d, and compute $(x^e)^d = x$.

The converse is not known to be true!

The trapdoor function in RSA

Under the **RSA assumption**:

$$f(x) = x^e \pmod{N}$$

is a trapdoor one-way function with d as the trapdoor.

Padded RSA

Padded RSA

Is “textbook RSA” IND-CPA?

Padded RSA

Is “textbook RSA” IND-CPA?
(No)

Padded RSA

- Textbook RSA is not IND-CPA

Padded RSA

- Textbook RSA is not IND-CPA (because deterministic)
- To make it IND-CPA, we can add a random **padding** to the message.

Padded RSA

- Textbook RSA is not IND-CPA (because deterministic)
- To make it IND-CPA, we can add a random **padding** to the message.

Padded RSA PKE

KeyGen:

- Choose P, Q prime, $N = PQ$
- Choose e prime with $\phi(N)$, compute d s.t. $\mathsf{ed} = 1 \pmod{\phi(N)}$.
- $\mathsf{sk} = \mathsf{d}, \mathsf{pk} = (N, \mathsf{e})$

Enc $\mathsf{m} \in \{0,1\}^{\ell}$

- Choose $\mathsf{r} \hookleftarrow U(\{0,1\}^{\log_2 N - \ell})$
- Compute $\mathsf{m}' \in \mathbb{Z}_N$ which has binary representation $(\mathsf{r} \| \mathsf{m})$
- Return $\mathsf{c} = (\mathsf{m}')^{\mathsf{e}}$.

Dec:

- Return the ℓ LSBs of $\mathsf{m} = \mathsf{c}^{\mathsf{d}} \mod N$.

Question

Is Padded-RSA IND-CCA secure?

(Assume that Dec returns the entire $C^{d} \mod N$).

Question

Is Padded-RSA IND-CCA secure?

(Assume that Dec returns the entire $C^d \mod N$).

- Choose a random k
- Compute $c' = k^e c \mod N$
- Send c' to the decryption oracle, get $m' = (c')^d \mod N$
- We have: $(r \| m) = m' \cdot k^{-1} \mod N$

Theorem

Theorem

If you have access to a black-box that, on input c, outputs whether $m = (c^d \mod N) < N/2$, then you can construct a decryption algorithm in $\mathcal{O}(n)$ calls to the black-box.

Theorem

Theorem

If you have access to a black-box that, on input c, outputs whether $m = (c^d \mod N) < N/2$, then you can construct a decryption algorithm in $\mathcal{O}(n)$ calls to the black-box.

Proof idea:

- Query with c: learn if $m \in [0; N/2[$
- Query with $2^{-e}c$: learn if $m \in [0; N/4[$ or ... assume that $m \in [N/4; N/2[$
- Query with $2^{-2e}c$: learn if $4m \mod N = 4m - N$ belongs to $[0; N/2[$
- ... (each time we manage to reduce the range)

This is from the MSB. We can do the same with the LSB.

Consequence

1.
Padded RSA is CPA-secure (under RSA assumption) $\implies$ we can transform a CPA distinguisher into an attacker for the RSA assumption.

Consequence

1.

Padded RSA is CPA-secure (under RSA assumption) $\Longrightarrow$ we can transform a CPA distinguisher into an attacker for the RSA assumption.

2.

Padded RSA is CCA-insecure.

Some more remarks / caveats

- N should be at least 2048 bits
- e with small Hamming weight makes the encryption more efficient
- BUT e should not be "too small"
- In padded-RSA, use $\ell = \mathcal{O}(\log N)$.
  RFC standard RSAES-PKCS1-V1_5 uses "at least 8 octets" of randomness.

Recap

- RSA relies on Fermat's little theorem and $(x^e)^d = x^{ed}$, where e is a public exponent and d a private one
- The security of RSA is **not** known to be equivalent to factoring (that's just the only way we attack the scheme in general)
- It relies on the **RSA assumption**, which is that the function $x \mapsto x^e \pmod{N}$ is a one-way trapdoor function
- Do NOT use "textbook" RSA, do NOT use the square-multiply algorithm for exponentiation