Public-Key Encryption

Viet Tung Hoang

Some slides are based on material from Prof. Stefano Tessaro, University of Washington
1. High-level PKE

2. Building PKE

3. Padding-oracle attack on PKCS1

4. CCA Security and OAEP
Motivation

**Problem:** Alice and Bob must be online simultaneously for key exchange
Public-Key Encryption (PKE): Syntax

Key Gen

\[ \mathcal{K} \rightarrow \$ \rightarrow \text{pk} \quad \text{sk} \]

Encrypt

\[ M \rightarrow \mathcal{E} \rightarrow \$ \rightarrow \text{C} \]

Decrypt

\[ \text{C} \rightarrow \mathcal{D} \rightarrow \text{M} \text{ or } \perp \]

Key Gen

\[ \mathcal{K} \rightarrow \$ \rightarrow \text{pk} \quad \text{sk} \]

Encrypt

\[ M \rightarrow \mathcal{E} \rightarrow \$ \rightarrow \text{C} \]

Decrypt

\[ \text{C} \rightarrow \mathcal{D} \rightarrow \text{M} \text{ or } \perp \]

\[ \text{pk} \quad \text{sk} \]

\[ \mathcal{K} \rightarrow \$ \rightarrow \text{pk} \quad \text{sk} \]

\[ M \rightarrow \mathcal{E} \rightarrow \$ \rightarrow \text{C} \]

\[ \text{C} \rightarrow \mathcal{D} \rightarrow \text{M} \text{ or } \perp \]

\[ \text{pk} \quad \text{sk} \]
Alice generates a pair of secret key and public key.
She keeps $sk$ to herself, and stores $pk$ in a public, trusted database.
PKE Usage

First retrieve Alice’s public key

\[ pk \]

Then email the encrypted message to Alice under her public key

\[ \mathcal{E}_{pk}(M) \]

Alice can later decrypt using her secret key

- Alice’s public key
- Bob’s public key
- Carol’s public key
- ...
Exercise: Sharing Encrypted Files

Encrypt a file so that when we place the ciphertext in a shared folder, only selected people can decrypt, assuming everybody has a public key.
PKE: CPA Security

- Similar to the Left-or-Right security of Symmetric encryption
- **Difference**: The adversary is given the public key

**Left**

procedure $\text{Enc}(m_0, m_1)$

Return $\mathcal{E}_{pk}(m_0)$

**Right**

procedure $\text{Enc}(m_0, m_1)$

Return $\mathcal{E}_{pk}(m_1)$

---

**Diagram:**

- **Enc**
  - Input: $(m_0, m_1)$
  - Output: $c$

- **A**
  - Input: $pk$
  - Output: $b'$
Performance Issue

Standard PKE schemes can only encrypt short messages (say ≤ 2048 bits)
How should we encrypt long ones?

A (not so good) solution:

- Break the message into small chunks
- Encrypt each chunk individually

\[
\begin{array}{c|c|c|c}
M_1 & M_2 & M_3 & M_4 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\mathcal{E}_{pk}(M_1) & \mathcal{E}_{pk}(M_2) & \mathcal{E}_{pk}(M_3) & \mathcal{E}_{pk}(M_4) \\
\end{array}
\]

**Problem**: PKE is very expensive, so this solution is several thousands times slower than AES-CTR
Hybrid Encryption

- Generate a random key $K$
- Encrypt the key $K$ by PKE, and use CTR under key $K$ to encrypt the message

Can replace CTR by your favorite symmetric encryption
Agenda

1. High-level PKE

2. Building PKE

3. Padding-oracle attack on PKCS1

4. CCA Security and OAEP
Number Theory Basics

For $n \in \{1, 2, 3, \ldots\}$, define

$\mathbb{Z}^*_n = \{t \in \mathbb{Z}_n \mid \text{gcd}(t, n) = 1\}$

$\varphi(n) = |\mathbb{Z}^*_n|$

**Theorem:**

- For any $s \in \mathbb{Z}^*_n$, $s^{\varphi(n)} \equiv 1 \pmod{n}$
- $\varphi$ is multiplicative: if $\text{gcd}(a, b) = 1$ then $\varphi(ab) = \varphi(a)\varphi(b)$

**Examples:** For distinct primes $p$ and $q$:

$\varphi(p) = p - 1$

$\varphi(pq) = (p - 1)(q - 1)$
The RSA Function

Given \( e, d \in \mathbb{Z}_{\varphi(n)}^* \) such that \( ed \equiv 1 \pmod{\varphi(n)} \)

Define a permutation \( f \) and its inverse \( f^{-1} \) as follows:

\[
f(x) = x^e \mod n
\]

\[
f^{-1}(y) = y^d \mod n
\]

Exercise: Try \( n = 55 \) and \( e = 3 \)
A Bad PKE: Plain RSA

**Key generation:**
- Pick two large primes $p, q$ and compute $n = pq$
- Pick $e, d \in \mathbb{Z}_{\varphi(n)}^*$ such that $ed \equiv 1 \pmod{\varphi(n)}$
- Return $pk \leftarrow (n, e), sk \leftarrow (n, d)$

**Encryption:**
- To encrypt message $x$ under $pk = (n, e)$, return $c \leftarrow x^e \mod n$

**Decrypt:**
- To decrypt a ciphertext $c$ under $sk = (n, d)$, return $x \leftarrow c^d \mod n$
Cracking Plain RSA: First Attempt

Public $e$, $N=pq$  \hspace{1cm}  \text{Secret } d

\[ ed \equiv 1 \pmod{(p-1)(q-1)} \]

A plausible attack:
- Recover $(p-1)(q-1)$
- Compute $d$ such that $ed \equiv 1 \pmod{(p-1)(q-1)}$

$O(\log(N))$ time using (extended) Euclidean algorithm

Question: Given $N=pq$ and $(p-1)(q-1)$, recover $p$ and $q$
Cracking Plain RSA: Second Attempt

For $e = 3$, a very common choice

For small messages $x < n^{1/3}$:

$$c = x^3 \mod n \quad \Rightarrow \quad x = c^{1/3}$$

**Exercise:** Recover message $x$ when one encrypts $x, x + 1, x + 2$
Why Is Plain RSA Bad?

It doesn’t meet the CPA notion

**Reason:** Plain RSA is *deterministic*

---

In 2016, QQ Browser was found to use Plain RSA to encrypt user data.

---

**China’s Top Web Browsers Leave User Data Vulnerable, Group Says**

Report from Citizen Lab accuses Tencent of weak encryption practices with its QQ Browser

*By Juro Osawa and Eva Dou*

March 28, 2016 5:00 p.m. ET
What Plain RSA Gives: Trapdoor permutation

A triple of algorithms (Gen, Samp, Inv)

\[(f, d) \leftarrow \text{Gen}, \text{ with } f : \text{Dom} \rightarrow \text{Range}\]

For \(x \leftarrow \text{Samp}\), it’s easy to compute \(y = f(x)\), but hard to invert \(f^{-1}(y)\) without knowing the trapdoor \(d\)

\[
\begin{align*}
\text{Dom} & \quad \text{easy via } \text{Inv}(d, \cdot) \\
\text{x} & \quad \text{easy} \\
\text{y} & = f(x) \\
\text{Range} & \quad \text{hard}
\end{align*}
\]
Building PKE from Trapdoor Permutation

Plain RSA $\rightarrow$ Hashed RSA

Given a trapdoor permutation (Gen, Samp, Inv) and a hash function $H$

**Key generation:** Run $(f, d) \leftarrow$ Gen and return $pk \leftarrow f$, $sk \leftarrow d$

**Encryption:** To encrypt message $M$ under $pk = f$

**Question:** How to decrypt?
Careful With Key Generation

Implementation issue: If initial randomness is weak (i.e. generated at boot time), many systems are likely to generate the same $p$

Question: Given $N_1 = pq_1$, $N_2 = pq_2$, recover $p, q_1, q_2$
0.75% of TLS certificates share keys, and another 1.7% may be susceptible
Agenda

1. High-level PKE

2. Building PKE

3. Padding-oracle attack on PKCS1

4. CCA Security and OAEP
PKCS #1 Encryption

- encrypt byte strings only
- Give shorter ciphertexts than Hashed RSA
- Uses encrypt-with-redundancy paradigm:
  Decryption will reject if the format is incorrect

1024 bits padded message

Plain RSA Enc

2 bytes

0002

2 bytes

non-zero, random bytes

$$

1 byte

00

1 byte

M

1 byte

X

Give shorter ciphertexts than Hashed RSA

Uses encrypt-with-redundancy paradigm:
Decryption will reject if the format is incorrect

Encrypt byte strings only
Padding-Oracle Attack

**Context:** Alice is establishing a TLS session with a server

server’s public key

\[ C \leftarrow E_{pk}(M) \]

48-byte secret

Adversary uses server as a decryption oracle by observing server’s accepting/rejecting of its fake ciphertexts
Padding-Oracle Attack

Recall $C = X^e \mod n$, with $pk = (e, n)$

Padded message

Pick some $r$

$C' \leftarrow C r^e \mod n$

$(X r)^e \mod n$

Accept only if $X r$ has valid PKCS encoding

By using several $r$, can fully recover $X$, and also $M$
Illustrative Toy Problem

\[ C' \leftarrow C r^e \mod n \]

Accept only if \((Xr \mod n) < n/2\)

\[ C' = (Xr)^e \mod n \text{ since } C = X^e \mod n \]
Key Idea: Binary Search

Initial search range of $X$: $\{0, \ldots, n - 1\}$

At each step, try to half the range of $X$ by carefully choosing $r$

- If $X < n$, pick $r = 1$
  - If $(Xr \mod n) < n/2$, $X < n/2$
    - If $X < n/4$, $X < n/4$
    - If $n/4 < X < n/2$, $n/4 < X < n/2$
  - If $(Xr \mod n) \geq n/2$, $n/2 < X < n$
    - If $n/2 < X < 3n/4$, $n/2 < X < 3n/4$
    - If $3n/4 < X < n$, $3n/4 < X < n$

Yes

No
A Quick Fix and Its Problem

**Want:** Change only server side, for backward compatibility

The change in TLS 1.0:
- If format or length of the decrypted message is incorrect, decryption returns a random 48-byte strings

Hiding decryption failure

**Problem:** Might be **broken** if implementation is not done properly to ensure that the timing is constant in both decryption success and failure.
Agenda

1. High-level PKE

2. Building PKE

3. Padding-oracle attack on PKCS1

4. CCA Security and OAEP
Resisting Padding-Oracle Attacks: CCA Security

Left

procedure Enc(m₀, m₁)
Return \( E_{pk}(m₀) \)

A is **prohibited** from feeding ctx from Enc to Dec.

Right

procedure Enc(m₀, m₁)
Return \( E_{pk}(m₁) \)
Achieving CCA Security: OAEP

**Use:** 1024-bit Plain RSA and **two** hash functions $H$ and $G$

Modeled as independent random oracles

How to get two hash functions from SHA-256: **Domain separation**

```
1 \quad x
\downarrow
\quad SHA

H(x)
```

```
0 \quad x
\downarrow
\quad SHA

G(x)
```
OAEP Design: Feistel Networks

**Design paradigm:** Two-round (unbalanced) Feistel

Feistel (in **decryption**)

Inverse Feistel (in **encryption**)

Diagram showing the operations involved in both decryption and encryption processes for the Feistel network.
OAEP Encryption

Use `encrypt-with-redundancy`. 

128 bits 

896 bits 

Use `encrypt-with-redundancy`. 

Plain RSA Enc
If $X[1:128] = 0^{128}$ then
Decode $X$ to get $M$
Else return