Public-Key Encryption

Viet Tung Hoang

Some slides are based on material from Prof. Stefano Tessaro, University of Washington

Agenda

1. High-level PKE

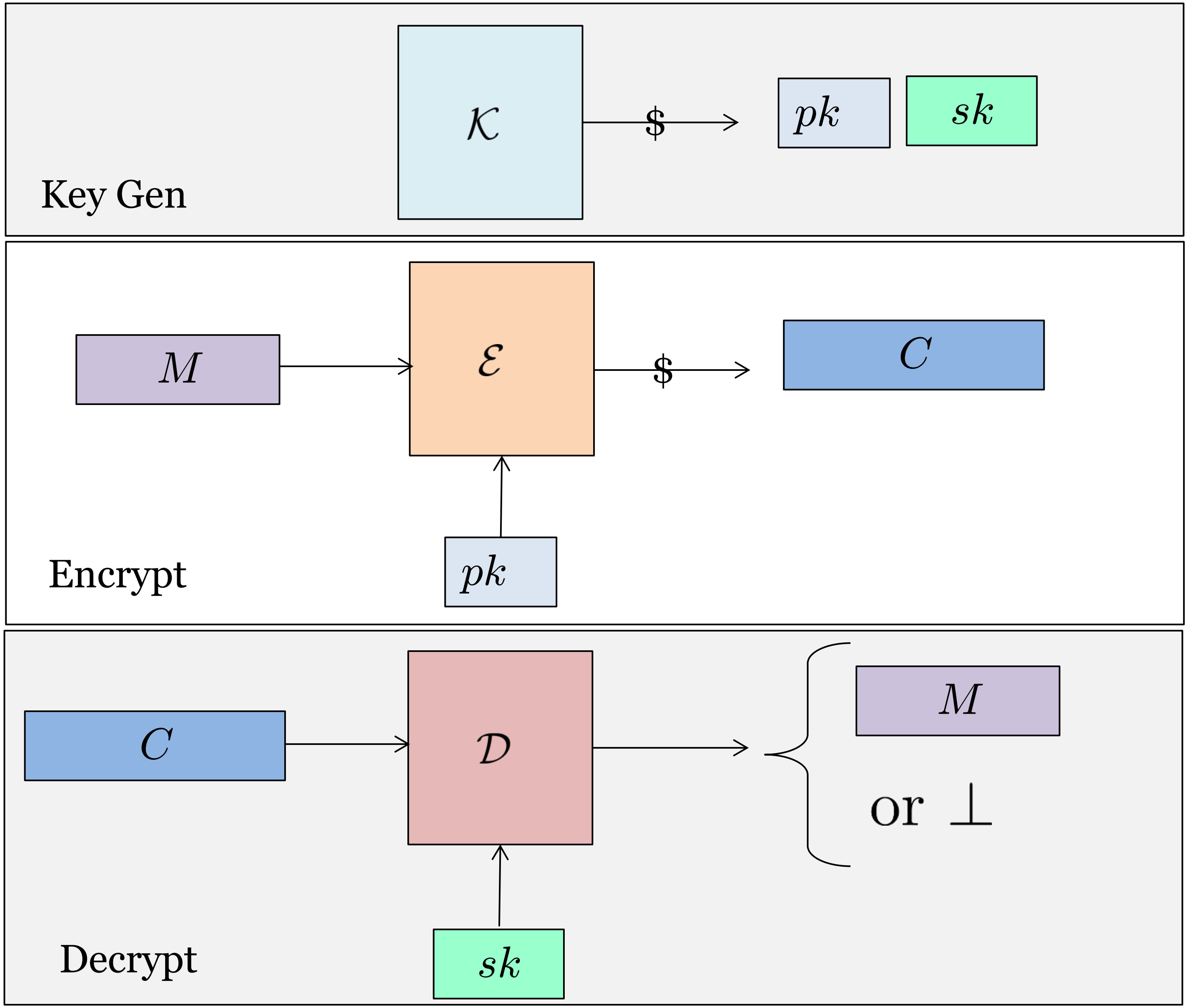
2. Building PKE

3. Padding-oracle attack on PKCS1

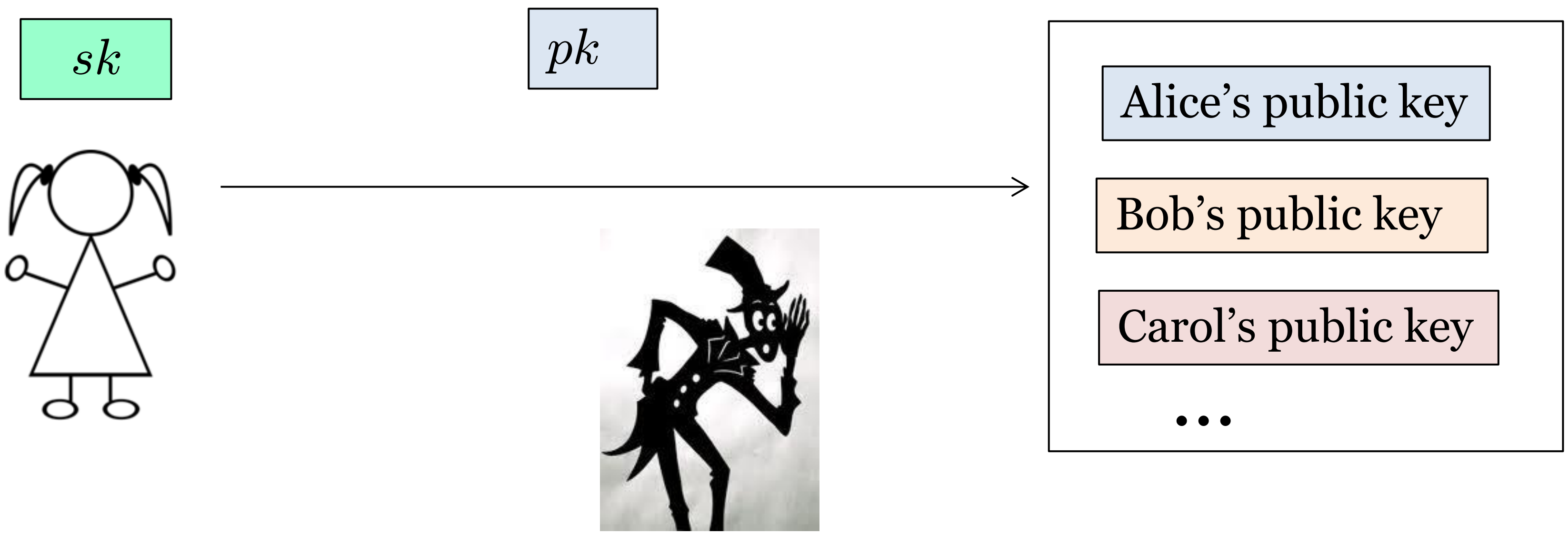
Motivation

Problem: Alice and Bob must be online simultaneously for key exchange

Public-Key Encryption (PKE): Syntax



PKE Usage

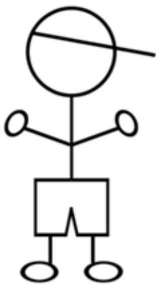


Alice generates a pair of secret key and public key.

She keeps sk to herself, and stores pk in a public, trusted database.

PKE Usage

First retrieve Alice's public key



pk

Alice's public key

Bob's public key

Carol's public key

...

$\mathcal{E}_{pk}(M)$

Then email the encrypted message

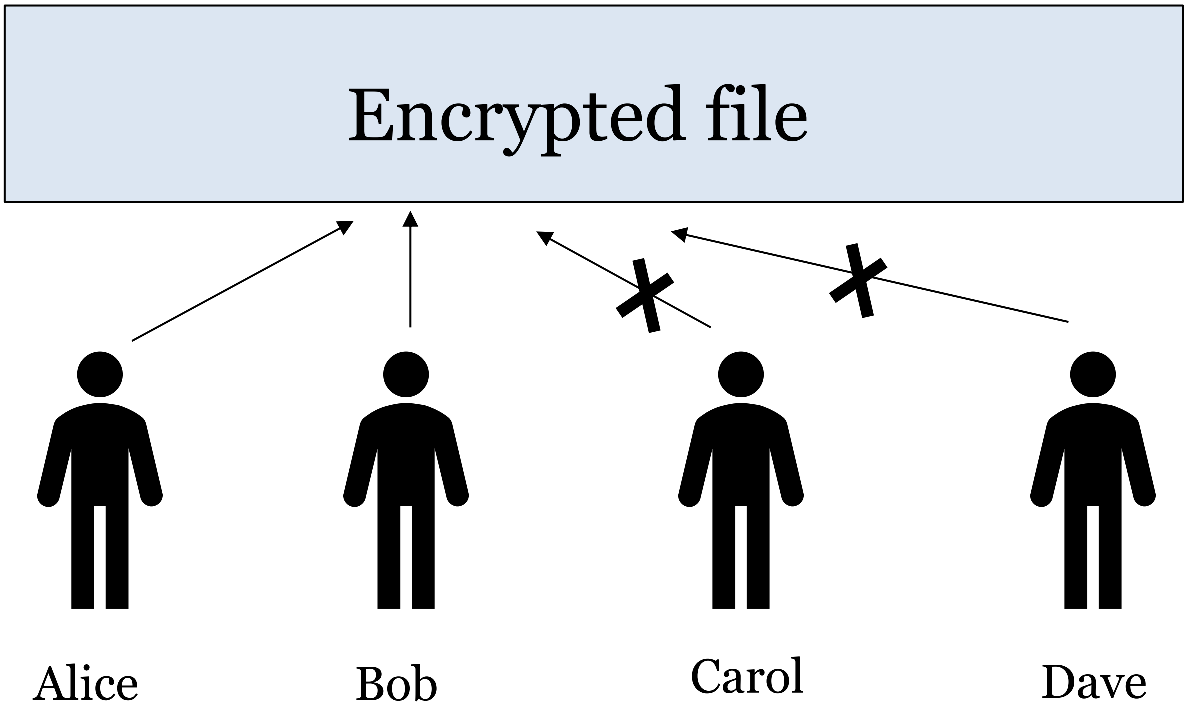
to Alice under her public key

Alice can later decrypt

using her secret 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

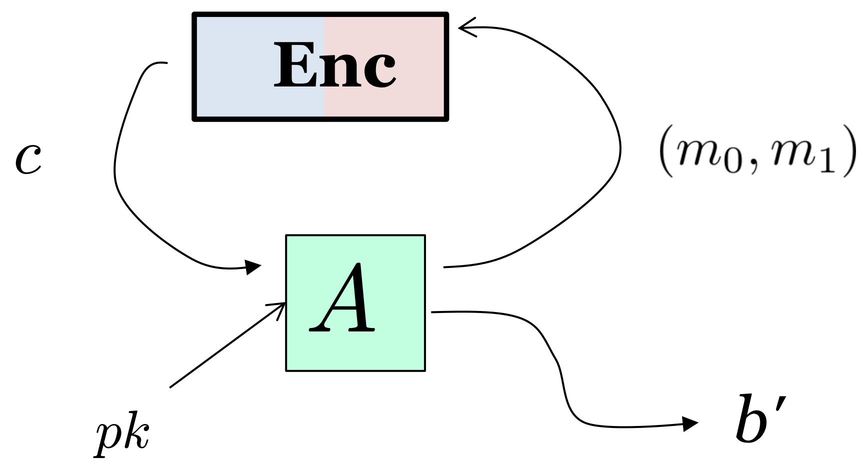
- Similar to the Left-or-Right security of Symmetric encryption
- **Difference**: The adversary is given the public key

Left

procedure Enc(m_0, m_1)
Return $\mathcal{E}_{pk}(m_0)$

Right

procedure Enc(m_0, m_1)
Return $\mathcal{E}_{pk}(m_1)$



Performance Issue

Standard PKE schemes can only encrypt short messages (say ≤ 2048 bits)

How should we encrypt long ones?

A (not so good) solution:

M_1	M_2	M_3	M_4

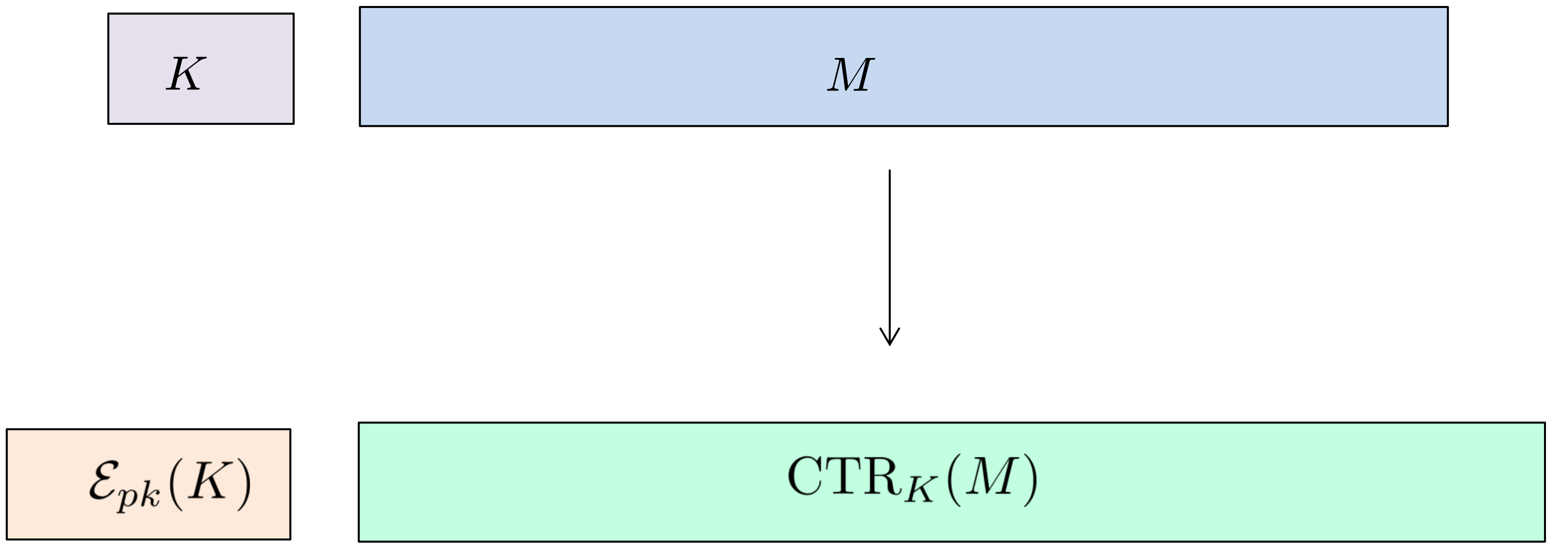
- -Break the message into small chunks
- -Encrypt each chunk individually

$$\downarrow$$

$\mathcal{E}_{pk}(M_1)$	$\mathcal{E}_{pk}(M_2)$	$\mathcal{E}_{pk}(M_3)$	$\mathcal{E}_{pk}(M_4)$

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

Number Theory Basics

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

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

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

Theorem:

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

Examples: For distinct primes p and q:

$$\varphi(p) = p - 1$$

$$\varphi(pq) = (p-1)(q-1)$$


RSA

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:

$$\begin{tikzcd} x \arrow[rr, bend left, "f(x) = x^e \bmod n"] & & y = f(x) \arrow[ll, bend left, "f^{-1}(y) = y^d \bmod n"] \end{tikzcd}$$

Exercise: Try $n = 55$ and $e = 3$

A Bad PKE: Plain RSA

Often e = 3 for efficiency

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 $c \leftarrow x^e \bmod n$

Decrypt:

- To decrypt a ciphertext c under $sk = (n, d)$, return $x \leftarrow c^d \bmod n$

Cracking Plain RSA: First Attempt

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

Public e, N=pq → Secret d

Require factoring N, which is a hard problem

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 $\underbrace{\text{Plain RSA:}}_{\text{For } e = 3\text{, a very common choice}}$ Second Attempt

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

$$c = x^3 \bmod n \Longrightarrow x = c^{1/3}$$

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

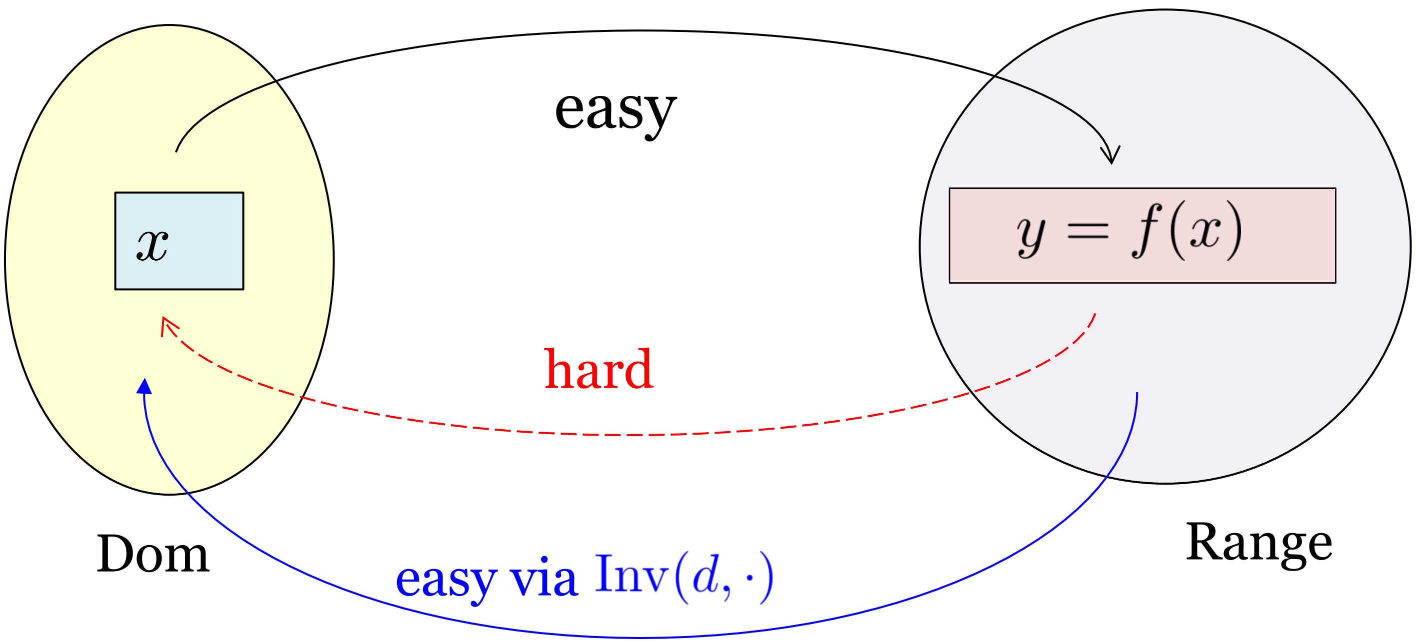
By Juro Osawa and Eva Dou

March 28, 2016 5:00 p.m. ET

What Plain RSA Gives: $\underbrace{\text{Trapdoor permutation}}_{\text{A triple of algorithms (Gen, Samp, Inv)}}$

$$(f, d) \leftarrow\!\!\$\ \mathrm{Gen},\ \text{with}\ f : \mathrm{Dom} \rightarrow \mathrm{Range}$$

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



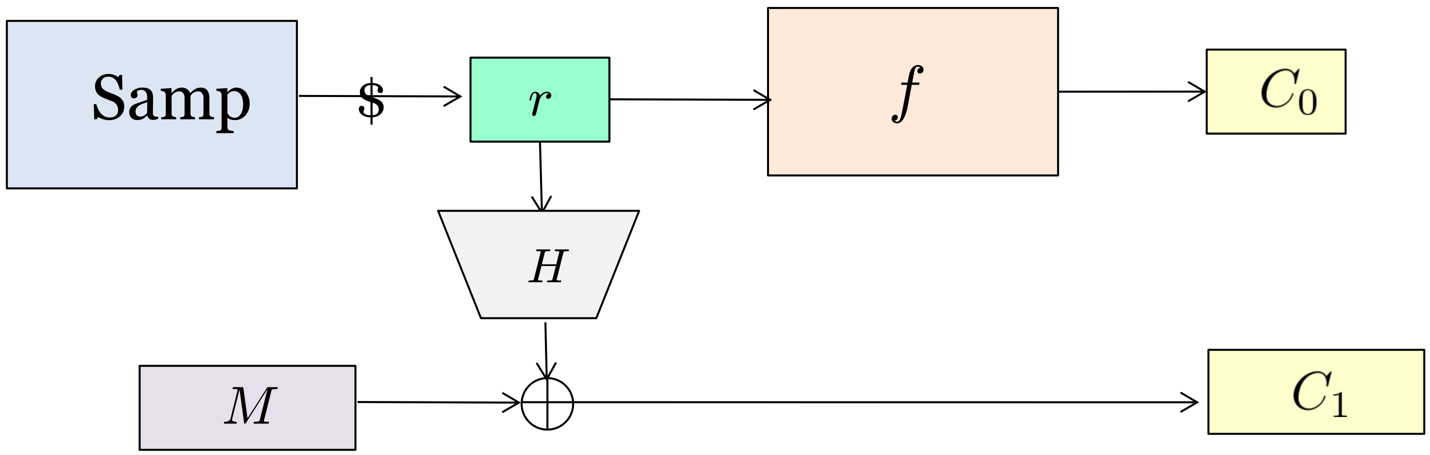
Building PKE from Trapdoor Permutation

Plain RSA → Hashed RSA

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

Key generation: Run $(f, d) \leftarrow_{\$} \mathrm{Gen}$ and return $pk \leftarrow f, sk \leftarrow d$

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



Question: How to decrypt?

Agenda

1. High-level PKE

2. Building PKE

3. Padding-oracle attack on PKCS1

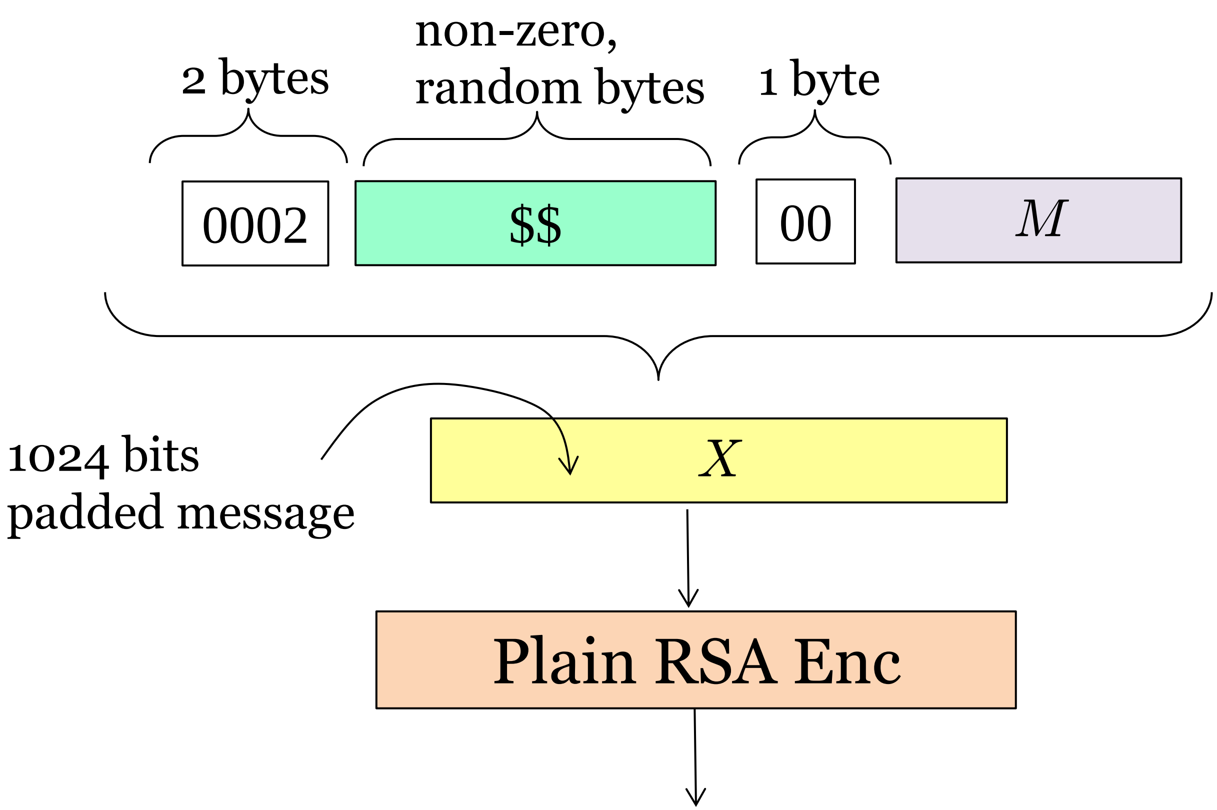
PKCS #1 Encryption

encrypt byte strings only

Give shorter ciphertexts than Hashed RSA

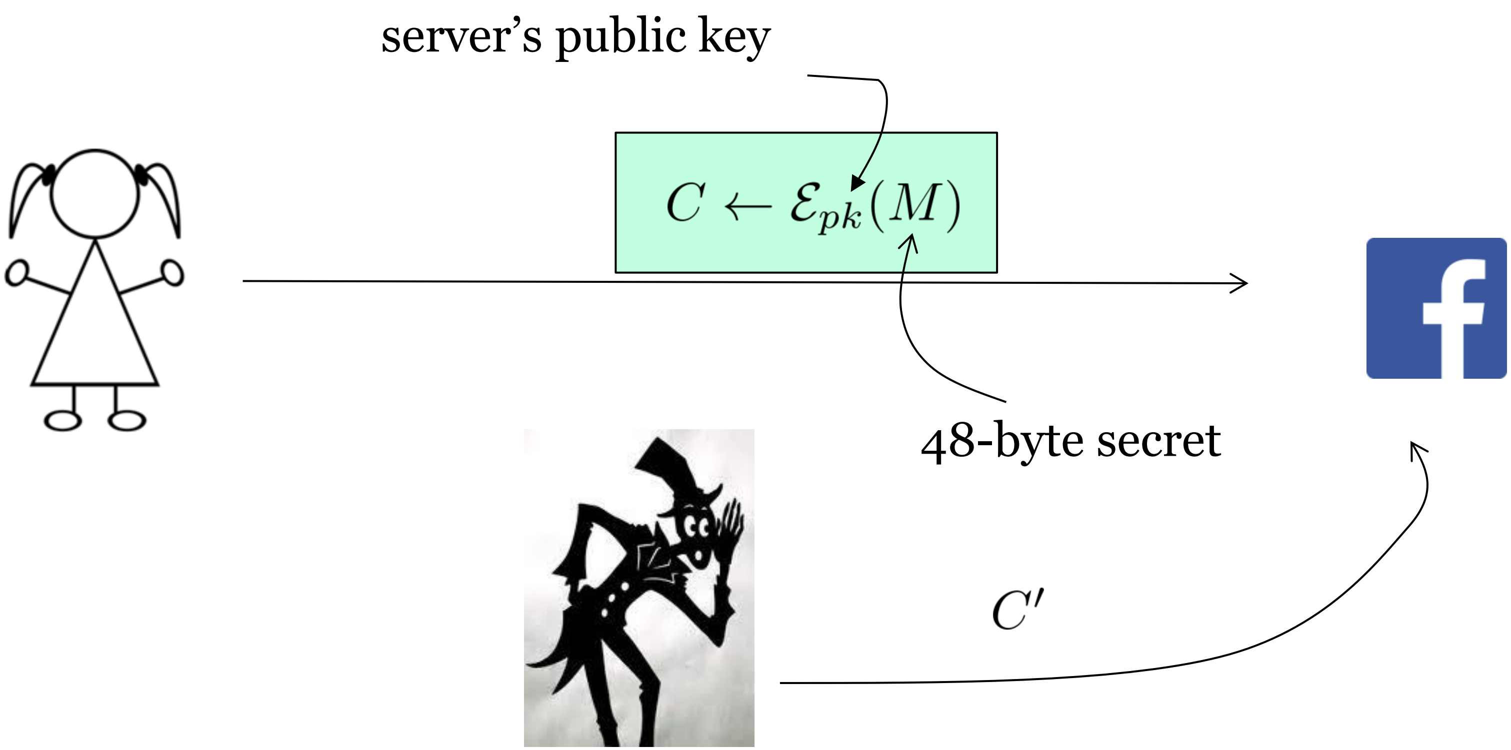
Uses encrypt-with-redundancy paradigm:

Decryption will reject if the format is incorrect



Padding-Oracle Attack

Context: Alice is establishing a TLS session with a server



Adversary uses server as a decryption oracle by observing

server's accepting/rejecting of its fake ciphertexts

Padding-Oracle Attack

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

Padded message

Pick some r

$C' \leftarrow Cr^e \bmod n$

$(Xr)^e \bmod n$

Accept only if Xr has valid PKCS encoding

By using several r, can fully recover X, and also M

Illustrative Toy Problem

$$\underbrace{\overbrace{\boxed{0}}^{\text{1 bit}}\ \boxed{M}}_{}$$

X

Plain RSA Enc

Only encrypt $M < n/2$

$C' \leftarrow Cr^e \bmod n$

Accept only if

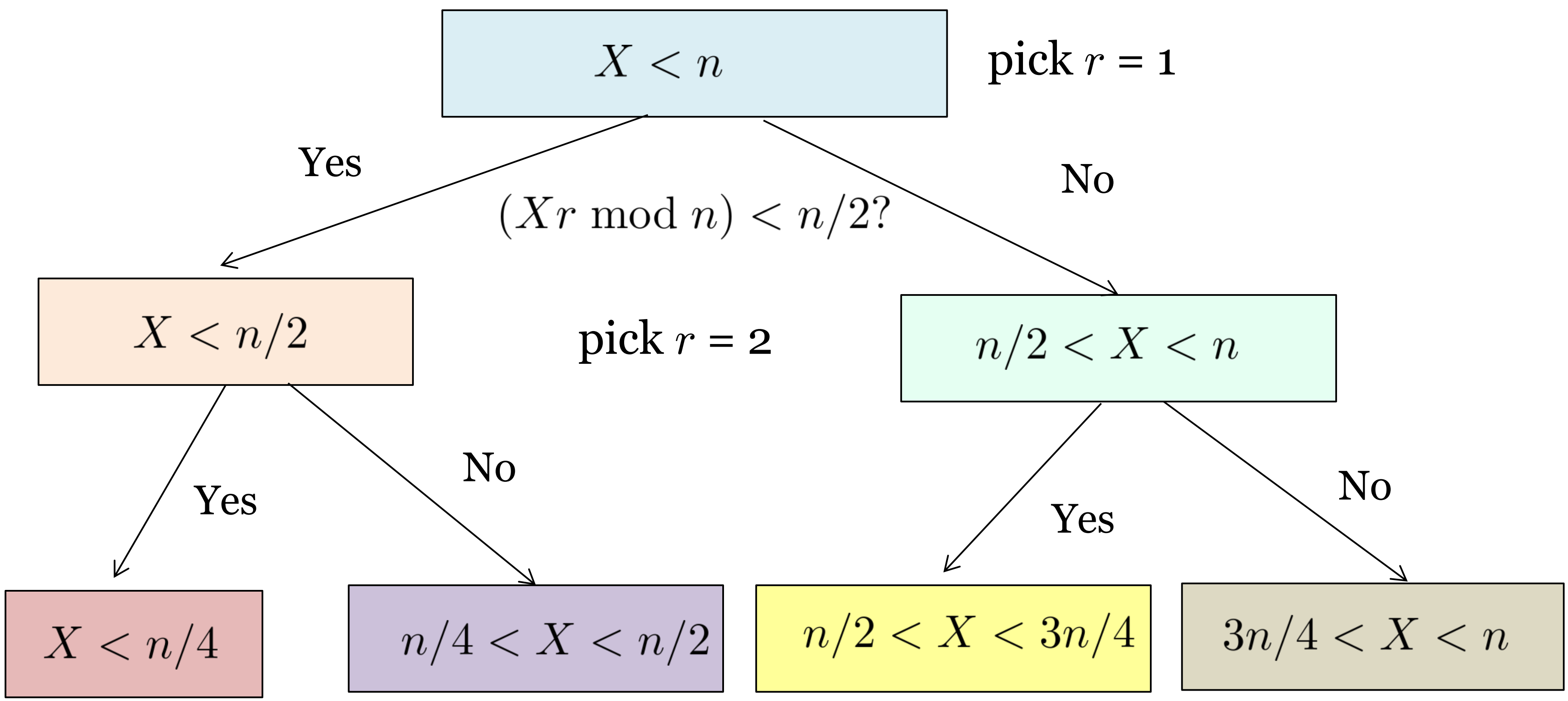
$(Xr \bmod n) < n/2$

$C' = (Xr)^e \bmod n$ since $C = X^e \bmod n$

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



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.