Lecture 10: Digital Signature

Viet Tung Hoang

The slides are loosely based on those of Prof. Mihir Bellare, UC San Diego.
Agenda

1. High-level Overview

2. Building Signature Scheme
Previously

Alice generates a pair of secret key and public key. She keeps $sk$ to herself, and stores $pk$ in a public, trusted database.
Problem

The adversary may replace Alice’s real key with its fake one
Solution

Alice has her $pk$ **certified** by CA

Tampering with $pk$ means one has to forge the CA’s digital signature

(Trusted) Certificate Authority (CA)

Alice’s certificate, signed by CA

Alice’s public key

Bob’s public key

Carol’s public key

…
Digital Signature Scheme: Syntax

<table>
<thead>
<tr>
<th>Key Gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{K}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
</tr>
<tr>
<td>$sk$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma$, $M$</td>
</tr>
<tr>
<td>$pk$</td>
</tr>
</tbody>
</table>
Digital Signature versus MAC

MAC

- Verifier needs to share a secret key with signer
- Verifier can impersonate signer

Digital Signature

- Verifier needs no secret
- Verifier cannot impersonate signer
Digital Signature: Unforgeability Security

- Similar to MAC security
- **Difference**: The adversary is given the public key

Again, digital signature doesn’t directly thwart replay attack.
Agenda

1. High-level Overview

2. Building Signature Scheme
A Bad Scheme: Plain RSA Signature

**Key generation:** Like RSA PKE
- 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 $pk \leftarrow (n, e), sk \leftarrow (n, d)$

**Sign:**
- To sign message $x$ under $sk = (n, d)$, “decrypt” it: $\sigma \leftarrow x^d \mod n$

**Question:** How to verify?
Issues with Plain RSA Signature

- **Feasibility**: Can sign only short messages

- **Security**: Can easily break unforgeability security

\[ pk = (n, e) \]

\[ (M', \sigma') = (x^e \mod n, x) \]

No tag query needed!
Hash-then-Sign Paradigm
Plain RSA Signature → Full Domain Hash (FDH)

Key generation: Like Plain RSA

Sign: To sign message $M$

$M$

$H$

$H : \{0, 1\}^* \rightarrow \mathbb{Z}_N^*$

Plain RSA Signing
Security Requirement for Hash Function

The hash must be collision-resistant

**Question:** Given a collision of the hash, break security of signature scheme

For the proof to go through, the hash has to be modeled as a random oracle
A Common Wrong Way to Hash

2048-bit string $Y$, viewed as a number in $\mathbb{Z}_N$

Broken by Desmedt and Odlyzko in 1985
How to Hash Properly

Use the first \( m = \left\lfloor \log_2(N) \right\rfloor \) bits and take mod \( N \)
Hashing in PKCS#1

19 fixed bytes for SHA-256

2 bytes padding

0001 FF FF ... FF 00 digest info

SHA

$x$

2048-bit string $Y$, viewed as a number in $\mathbb{Z}_N$
Verification in PKCS#1

Question: How would you implement the encoding check?

Is $Y$ an encoding of $\text{SHA-256}(x)$?
Bad Check for PKCS#1 Signature

Do not check if SHA output is the end of the encoded string

Totally 2048 bits

0001  FF FF ... FF 00  digest info  SHA-256(x)  

Allow some suffix
Attacking PKCS#1 Signature with Bad Check

Using $e = 3$ as verification key in RSA

How to forge signature for message $x$:

Choose suffix length so that $|Z| = 680 \approx 2048/3$
Want $\sigma$ s.t. $Y = \sigma^3 \mod N \Rightarrow$ Output $s = \lceil Y^{1/3} \rceil$
Why It Works

\[ 0 \leq Y' - Y < 2^{1368} \]

\[ s^3 = Y' \]

0001 FF FF ... FF 00 digest info SHA-256(x) Y' - Y

Z

1368 bits

Still pass the bad checking!