CS555 Cryptography Notes - Lecture 14

Digital Signatures

October 9, 2025

Constructing a collision-resistant hash function from a OWF is an open problem, but it has been shown to be impossible with certain CRHFs.

1 Stateful Many-time Signatures

Idea: Signature Chains

Alice starts with secret signing key SK_0 (shared consensus).

When signing a message m_1:

- Generate new pair (VK_1, SK_1)
- Produce signature $\sigma_1 \leftarrow \text{Sign}(SK_0, m_1||VK_1)$
- Output $VK_1||\sigma_1$
- Remember $VK_1||m_1||\sigma_1$ as well as SK_1

The signature includes VK_i to make the next signature verifiable. Every signature should include the verification key for the next signature.

1.1 For next message m_2

Generate new pair (VK_2, SK_2). Produce signature $\sigma_2 \leftarrow \text{Sign}(SK_1, m_2||VK_2)$.

Output $VK_1||m_1||\sigma_1||VK_2||\sigma_2$.

$$VK_0 \xrightarrow{\sigma_1} VK_1 \xrightarrow{\sigma_2} VK_2 \rightarrow \cdots$$

The whole chain can be verified.

The cost is that the chain grows in size. Signature trees are nice to shrink the chain.

2 Step 2: Shrinking Signatures with Signature Trees

Alice (the stateful signer) computes many (VK, SK) pairs and arranges them in a tree of depth = sec. param. λ.

$$
\begin{tikzcd}
 & & & VK_\epsilon \arrow[dlll] \arrow[drr] & & & \\
VK_0 \arrow[d] \arrow[dr] & & & & & VK_1 \arrow[d] \arrow[dr] & \\
VK_{00} \arrow[d] & VK_{01} & & & & VK_{10} & VK_{11} \\
VK_{000} & VK_{001} & & & & &
\end{tikzcd}
$$

Every signature has λ nodes that are signed.
Every level has a verification key.

2.1 Signature of first message m_0

$$\sigma_\epsilon \leftarrow \text{Sign}(SK_\epsilon, VK_0 || VK_1)$$

$$\sigma_0 \leftarrow \text{Sign}(SK_0, VK_{00} || VK_{01})$$

$$\sigma_{00} \leftarrow \text{Sign}(SK_{00}, VK_{000} || VK_{001}), \quad \tau_0 \leftarrow \text{Sign}(SK_{000}, m_0)$$

Output: $(\sigma_\epsilon, \sigma_0, \sigma_{00}, \tau_0)$ - authentication path for VK_{000}.

2.2 Complete Signature Construction

The signature of message m_0 consists of:

- Authentication path for VK_{000}
- $\tau_0 \leftarrow \text{Sign}(SK_{000}, m_0)$

GOOD NEWS:

- Each verification key (including at the leaves) is used only once, so one-time security suffices!
- Signatures consist of λ one-time signatures and do not grow with time!

BAD NEWS:

- Signer generates and keeps the entire ($\approx 2^\lambda$-size) signature tree in memory!

3 Step 3: Pseudorandom Signature Trees

To reduce storage, use pseudorandom trees.
Tree of pseudorandom values: Populate the nodes with $r_x = \text{PRF}(K, x)$.
The signing key is a PRF key K.
Use r_x to derive the keys $(VK_x, SK_x) \leftarrow \text{Gen}(1^\lambda; r_x)$.
GOOD NEWS:

- Short signatures and small storage for the signer

BAD NEWS:

- Signer needs to keep a counter indicating which leaf (which tells her which secret key) to use next

4 Step 4: Making Signer Stateless

If we don't have a fresh key, this becomes a deterministic function, which breaks security properties.

4.1 Statelessness via Randomization

Pick a random leaf r. Use VK_r to sign m.

$\sigma_r \leftarrow \text{Sign}(SK_r, m)$.

Output $(r, \sigma_r, \text{authentication path for } VK_r)$.

GOOD NEWS:

- No need to keep state

Key Idea: If the signer produces Q signatures, the probability she picks the same leaf twice is $\leq q^2/2^\lambda$.

We can't sign 2^λ (exponential) messages, but we can still sign poly many messages.

5 Step 5: Making Signer Deterministic

Idea: PRFs.

Generate r pseudo-randomly. Have another PRF key K' and let $r = \text{PRF}(K', m)$.

6 Summary

Putting everything together, we showed that assuming the existence of one-way functions and collision-resistant hash function families, there are digital signature schemes.

Theorem 1. *Digital signature schemes exist if and only if one-way functions exist.*

Collision-resistant hashing is not necessary [unnecessary, sufficient].

$$\boxed{OWF \rightarrow PRG \rightarrow PRF}$$

$$\downarrow$$

Digital signatures

7 Direct Constructions

7.1 "Vanilla" RSA Signatures

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\phi(N)$ and let $d = e^{-1} \pmod{\phi(N)}$.

$SK = (N, d)$ and $VK = (N, e)$.

Sign(SK, m): Output signature $\sigma = m^d \pmod N$.

Verify(VK, m, σ): Check if $\sigma^e = m \pmod N$.

7.2 Is this secure?

Security definition of signatures:

Adversary interacts with user:

- Adv sends $m_1, m_2, \ldots, m_Q$
- User responds with $\text{Sign}(m_1), \text{Sign}(m_2), \ldots$
- Adv outputs forgery: $(m^*, \text{Sign}(m^*))$

Success condition: $m^* \notin \{m_1, \ldots, m_Q\}$ or $\text{Sign}(m^*) \neq$ any previous signature.

Can be m_i if Sign$(m^*) \neq$ Sign(m_i).

7.3 Problems with Vanilla RSA

Problem 1: Existentially forgeable!

Attack: Pick a random σ and output $(m = \sigma^e, \sigma)$ as a valid forgery.

Problem 2: Malleable!

Attack: Given a signature of m, you can produce a signature of $2^e \cdot m$, $3^e \cdot m$, $\ldots$, m^2, m^3, $\ldots$

8 How to Fix Vanilla RSA

Start with any trapdoor permutation, e.g. RSA.

Gen(1^λ): Pick primes (P, Q) and let $N = PQ$. Pick e relatively prime to $\phi(N)$ and let $d = e^{-1} \pmod{\phi(N)}$.

$SK = (N, d)$ and $VK = (N, e, H)$.

Sign(SK, m): Output signature $\sigma = H(m)^d \pmod N$.

Verify(VK, m, σ): Check if $\sigma^e = H(m) \pmod N$.

8.1 What should H be?

H should be one-way to prevent attack where $m^* = \sigma^e$ (reverse-engineering the message from the signature).

H must be hard to "algebraically manipulate" $H(m)$ to $H(\text{related } m')$ to prevent picking m^* from $\Sigma m_1, \ldots, m_Q$ such that $H(m^*) = (m_1, m_2, \ldots)^d$ (squared or other operations).

Collision-resistant hash functions do not readily satisfy this property.

What does this second property mean?

9 Random Oracle Heuristic

Want: a public H that is "non-malleable".

Given $H(m)$, it is hard to produce $H(m')$ for any non-trivially related m'.

For every PPT adversary A and every non-trivial relation R,

$$\Pr[A(H(m)) = H(m') : R(m, m') = 1] = \text{negl}(\lambda)$$

Proxy: A public H that "behaves like a random function".

(A PRF also behaves like a random function, but PRF_K is NOT publicly computable).

In practice, we let H be the SHA-3 hash function. There are no known attacks or proofs of security for RSA + SHA3.