

Lecture 21: Collision-Resistant Hash Functions and General Digital Signature Scheme

Instructor: Rafael Pass

Scribe: Chin Isradisaikul

In this lecture we discuss several attacks on collision-resistant hash functions, construct families of collision-resistant hash functions from reasonable assumptions, and provide a general signature scheme for signing many messages.

1 Collision-Resistant Hash Function

Recall that $h : \{0, 1\}^n \rightarrow \{0, 1\}^m$ is a collision-resistant hash function (shorthand CRHF) if it satisfies the following properties:

- (length-compressing): $m < n$. Typically, $m = n/2$.
- (hard to find collisions): For all nuPPT A , there exists a negligible function ε such that for all security parameters $n \in \mathbb{N}$,

$$\Pr[(x_0, x_1) \leftarrow A(1^n, h) : x_0 \neq x_1 \wedge h(x_0) = h(x_1)] \leq \varepsilon(n).$$

However, a nonuniform adversary can simply remember x_0 and x_1 such that $x_0 \neq x_1$ and $h(x_0) = h(x_1)$ (such a pair exists by length compression) for each security parameter, and therefore it is not hard to find a collision for h using a nonuniform adversary. Hence, we require a family \mathcal{H} of CRHFs, where we can sample a CRHF from \mathcal{H} easily. The collision-resistant property then requires that any nonuniform adversary A should not find a collision easily given a CRHF h drawn randomly from \mathcal{H} .

1.1 Breaking Collision-Resistant Hash Function

We now consider several ways to attack and find a collision for a CRHF.

1.1.1 Enumeration

Consider a one-bit compression h , i.e., $|h(x)| = |x| - 1$. Let $n = |h(x)|$. The probability that two random elements x and x' hash to the same value is at least $\frac{1}{2^n}$. However, it might be the case that $x = x'$, which occurs with probability $\frac{1}{2^{n+1}}$. Hence, the probability that two random elements collide is at least $\frac{1}{2^n} - \frac{1}{2^{n+1}}$. That is, to ensure that two random elements collide, we have to search the number of message pairs approximately almost the size of the range of h .

1.1.2 Birthday Attack

Instead of enumerating on pairs of messages, we will try to pick a sufficient number of messages so that two of them collide. This is known as the Birthday Attack. Suppose we pick t random messages. The probability that two of these messages collide is at least

$$\binom{t}{2} \left(\frac{1}{2^n} - \frac{1}{2^{n+1}} \right) \approx O \left(\frac{t^2}{2^n} \right).$$

If $t \approx \sqrt{2^n}$, then the probability is decent, i.e., a constant, so it is likely that two of these messages collide.

1.2 Constructing a Collision-Resistant Hash Function from Reasonable Assumptions

1.2.1 CRHFs from Discrete Log Assumption

Let p be an n -bit prime, where n is the given security parameter. Let g be a generator of \mathbb{Z}_p^* . Recall that the Discrete Log assumption states that it is difficult to find $x \in \mathbb{Z}_p^*$ given $g^x \in \mathbb{Z}_p^*$, where x is drawn randomly from \mathbb{Z}_p . Construct a CRHF as follows:

- Pick $y \in \mathbb{Z}_p^*$ at random.
- $h_{p,g,y}(x, b) = y^b g^x \bmod p$, where $x \in \mathbb{Z}_p^*$ and $b \in \{0, 1\}$.

Observe that h is one-bit compressing.

Theorem 1 Under the Discrete Log assumption, $h_{p,g,y}$ is a one-bit compressing collision-resistant hash function.

Proof: Suppose there is an adversary A that finds (x, b) and (x', b') so that they collide. There are two cases to consider:

- $b = b'$: Then $g^x \bmod p = g^{x'} \bmod p$. Since g is a generator of \mathbb{Z}_p^* , the set g generates, which is

$$\{g^x \bmod p \mid x \in \mathbb{Z}_p^*\},$$

is a permutation of \mathbb{Z}_p^* . Hence, $x = x'$, so these two pairs are identical and they do not represent a collision.

- $b \neq b'$: Without loss of generality, assume that $b = 0$ and $b' = 1$. Then

$$g^x = h(x, b) = h(x', b') = g^{x'} y \pmod{p}.$$

That is, $g^{x-x'} \equiv y \pmod{p}$. In this case, we can construct another adversary A' that breaks the Discrete Log assumption. On input (p, g, y) , A' does the following:

- Feed p, g, y into A . Let (x, b) and (x', b') be the result. Observe that $b \neq b'$ by the argument above.
- If $b = 0$, output $x - x'$. Otherwise, output $x' - x$.

It follows that if A succeeds with nonnegligible probability, A' also succeeds with that nonnegligible probability, so A' breaks the Discrete Log Assumption. \square

Remark: By the Birthday Attack above, Discrete Log can be broken in time $2^{\lfloor p/2 \rfloor}$ because we can keep generating messages and see if the hash values (from modular exponentiation) of any two of them are the same.

Note: Having a one-bit compressing CRHF h , we can obtain polynomial-length compressing CRHF by repeatedly applying h . Observe that this method is not multi-message secure for digital signatures because m and $h(m)$ hash to the same value.

1.2.2 CRHFs from Factoring Assumption

We state such a collection without a proof.

Let N be a safe prime. Define $h_{N,y} = y^b x^2 \pmod N$, where $x \in \{1, \dots, \frac{N-1}{2}\}$ —the first half of the values in \mathbb{Z}_N^* , $y \in QR_N = \{y \in \mathbb{Z}_N^* \mid \exists x \in \mathbb{Z}_N^* : y = x^2\}$ —set of elements in \mathbb{Z}_N^* that have a square root, and $b \in \{0, 1\}$.

Note that for $x, -x \in \mathbb{Z}_N^*$, only one is in the first half of \mathbb{Z}_N^* .

1.3 History of Hash Functions in Practice

Hash Scheme	Year Constructed	Number of Bits	Year Broken
MD4	1990	128	1995
MD5	1992	128	1998
SHA1	1994	160	2005*
SHA-256	2005	256	?

For SHA1, only a collision is found in 2005, but it is not generally broken (yet).

2 One-Time Digital Signature Scheme for any Message

Let $(\text{Gen}, \text{Sign}, \text{Ver})$ be a one-time secure digital signature scheme for messages in $\{0, 1\}^n$ and $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$ be a collision-resistant hash function. We construct a one-time digital signature scheme $(\text{Gen}', \text{Sign}', \text{Ver}')$ as follows. The scheme is based on “Hash and Sign” idea:

- Gen' : Generate a public-private key pair (pk, sk) and sample a CRHF h .
- $\text{Sign}'_{sk}(m)$: Sign the hash of m , i.e., output $\text{Sign}_{sk}(h(m))$.

- $\text{Ver}'_{pk}(\sigma, m)$: Verify that σ is the signature of $h(m)$, i.e., output $\text{Ver}_{pk}(h(m), \sigma)$.

Theorem 2 If there exists a collision-resistant hash function and there exists a one-way function, then $(\text{Gen}', \text{Sign}', \text{Ver}')$ is a one-time secure digital signature scheme for $\{0, 1\}^*$.

Proof Idea: Suppose there is an adversary A that queries the Sign oracle with message m (only once) and outputs $m', \text{Sign}'(m')$, where $m' \neq m$. There are two cases to consider:

- $h(m) = h(m')$: In this case, A is able to find a collision for h . We can use A to construct another adversary that uses A to find a collision for h , contradicting the assumption that h is a CRHF.
- $h(m) \neq h(m')$: In this case, A is able to obtain $\text{Sign}'(m')$ directly, i.e., it finds a signature for $h(m')$. This violates the assumption that the original signature scheme is one-time secure.

Therefore, $(\text{Gen}', \text{Sign}', \text{Ver}')$ is a one-time secure digital signature scheme for $\{0, 1\}^*$. \square

3 General Digital Signature Scheme for Signing Many Messages

Having a digital signature scheme that can only be used for a single message is not so useful. In this section we construct a general signature scheme for signing multiple messages. The idea is to generate new keys each time a message is to be signed. By generating new keys, the scheme acts as if it were to be used only once, and one-time security follows.

3.1 Approach 1: Generate a new key pair after signing each message

At the beginning, we are given the initial public-private key pair (pk_0, sk_0) from a trusted party. We sign messages as follows:

- First message:
 - Generate (pk_1, sk_1) .
 - $\sigma_1 \leftarrow \text{Sign}_{sk_0}(m_1 \parallel pk_1)$
 - Output $\sigma'_1 = (1, \sigma_1, m_1, pk_1)$.
- Second message:
 - Generate (pk_2, sk_2) .
 - $\sigma_2 \leftarrow \text{Sign}_{sk_1}(m_2 \parallel pk_2)$

- Output $\sigma'_2 = (2, \sigma_2, m_2, pk_2, \sigma'_1)$.

- In general, for message i :

- Generate (pk_i, sk_i) .

- $\sigma_i \leftarrow \text{Sign}_{sk_{i-1}}(m_i \parallel pk_i)$

- Output $\sigma'_i = (i, \sigma_i, m_i, pk_i, \sigma'_{i-1})$,

where σ'_0 is empty.

We can see that this scheme has a disadvantage: The length of a signature grows linearly with the number of messages.

3.2 Approach 2: Generate two new key pairs

To avoid linear-order-length signatures, we generate two new key pairs (instead of one) from each established key pair. Then we can layer these key pairs in a complete binary tree, where each key signs all of its children. We only sign messages with the key pair at a leaf. Hence, a signature containing $n + 1$ key pairs (making a tree of height n) can be used to sign 2^n messages. A signature for each message only needs to contain the sequence of key pairs that generate the key that signs the message, where the last in the sequence is the key pair at the root of the tree, which is provided by a trusted party. In this way, the recipient of the message can verify that the message is signed by the key pair at the leaf, and by tracing back the provided key-pair sequence, the recipient can verify that the sender actually signed this message.

Note that we have to generate the key pairs only when they are needed. This process is shown in Figure 1 for a complete height-2 tree. Even though this approach dramatically reduces

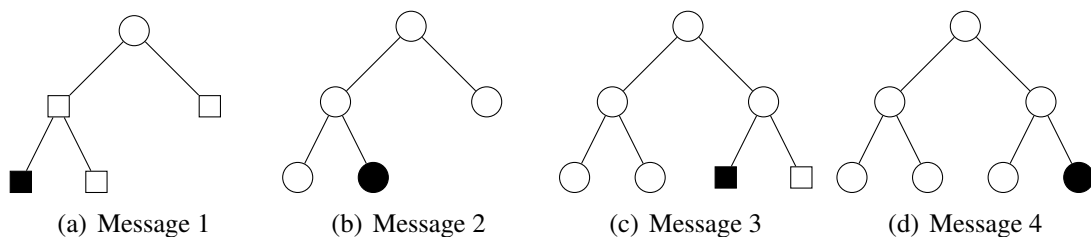


Figure 1: The generations of key pairs for a complete height-2 tree. The rectangles indicate the new key pairs generated before the corresponding message is signed.

the length of signatures, we still have to keep track of the state, i.e., how many message we have signed so far and which key pairs we have generated, so that we can generate the next key pairs correctly. Note that every pair of key is only used to **sign one message**; non-leaf key-pairs only sign their children, and leaf key-pairs only sign a message once. This is crucial for security.

3.3 Approach 3: Sign message m using the m^{th} leaf

To eliminate state, we can pre-generate all the key pairs for all the tree node. Since (the hash of) messages are in $\{0, 1\}^n$, where n is the height of the tree, we can treat these hash values as a number from 0 to $2^n - 1$. To sign a hash value m , we use the key-pair sequence from the root to the leaf numbered m .

The obvious disadvantage for this approach is that we have to generate all the key pairs, even though we might sign only a few messages of length n . This also consumes a lot of space.

3.4 Approach 4: Use a pseudorandom number generator with a seed to generate key pairs

To avoid taking up space, we can use the initial key pair (from trusted party) to sign a seed for a pseudorandom number generator. Having a seed, we can generate key pairs as needed by tracing the path from the leaf to the root. In this way, we do not even have to keep track of the last key-pair sequence used for the last message we signed.