Winter 2018 CS 485/585 Introduction to Cryptography

Lecture 8

Portland State University

Lecturer: Fang Song

Jan. 25, 2018

Draft note. Version: February 4, 2018. Email fang.song@pdx.edu for comments and corrections.

Agenda

- (Last time) PRF-MAC, Domain-extension: Cascade
- Hash functions, collision resistance, generic security
-

Hash functions

Today we introduce another basic primitive in cryptography – *hash functions*. Roughly they are functions that compress long inputs to short *digests*. The primary requirement is to avoid *collision*[1].

[1] a collision is a pair of inputs (x_1, x_2) such that $h(x_1) = h(x_2)$.

In data structures you've heard about building a hash table that enables quick look up for an element. A "good" hash table introduces as few *collisions* as possible.

The basic idea is similar in the cryptographic setting, but with significantly more stringent criteria. Therefore we call *cryptographic* hash functions to stress this. [2]

[2] As for PRG, ordinary hash tables should not be used for cryptographic purposes.

- Collision resistant is a *must* rather than a feature "nice-to-have".
- It is fair to assume that the data elements in the context of data structures are not chosen to cause collision intentionally. But in the crypto-setting, attackers are making every effort to create collisions.

Defining collision-resistance

Definition 1. A hash function is an efficient (deterministic poly-time) algorithm $H : \{0,1\}^* \to \{0,1\}^{\ell(n)}$. If H is defined only for inputs $x \in \{0,1\}^{\ell'(n)}$ with $\ell'(n) > \ell(n)$, then we call H a compression function.

For technical reason (i.e., non-uniform adversaries), the textbook considers keyed hash functions $H : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$: $H^s(x) := H(s, x)$. Here the key is not meant to be kept secret, so it is written in superscript. A *non-uniform* adversary can hardwire a collision (x, x') for $h : \{0,1\}^* \to \{0,1\}^n$ and break collision resistance trivially. Therefore the key, or rather a *system parameter* as Boneh-Shoup call it, s is introduced, to resolve this technicality since no efficient adversary can hardwire a collision for every possible s.

Collision-resistance will be our security goal, and we give a formal definition by the following *collision-finding* game.

Definition 2. H is collision resistant if for any PPT $\mathcal{A}$

$$\Pr[\mathsf{H\text{-}coll}_{\mathcal{A},H}(n) = 1] \leq \mathrm{negl}(n)\,.$$

1. Adversary $\mathcal{A}$ is given 1^n and output (x, x').
2. $\mathcal{A}$ succeeds if $x \neq x'$ and $H(x) = H(x')$. Define the output of the game $\mathsf{H\text{-}coll}_{\mathcal{A},H}(n) = 1$ in this case, and $\mathsf{H\text{-}coll}_{\mathcal{A},H}(n) = 0$ otherwise.

Figure 1: Collision-finding experiment $\mathsf{H\text{-}coll}_{\mathcal{A},H}(n)$

Generic attacks on hash functions

Let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a hash function. How hard is it to find collisions in H? We consider generic attacks, which do not rely on the specific structure of a hash function and hence apply to arbitrary hash functions. This gives guideline for the minimum security one should aim for.

Direct attack: evaluate $2^\ell + 1$ distinct inputs, and there must be a collision[3]. How about evaluating q elements? what is probability that there is a collision? We analyze it for a random function, and this leads us to the famous *birthday problem.*

[3] Pigeonhole Principle

The birthday problem

Choose q elements $y_1, \ldots, y_q$ from a set of size N uniformly at random with replacement, what is the probability that there exist $i \neq j$ with $y_i = y_j$?

Let *coll* denote this event, and $Coll_{i,j}$ denote the event that (y_i, y_j) form a collision.

Lemma 3. $\Pr[Coll] = \Theta(q^2/N)$. *Specifically*

$$\Pr[Coll] \leq q^2/2N, \quad \textit{and } \Pr[Coll] \geq \frac{q(q-1)}{4N} \textit{ for } q \leq \sqrt{2N}\,.$$

The upper bound ensures that when q is small, it is very unlikely to see a collision[4]. The lower bound, on the other hand, promises that when $q = \Omega(\sqrt{N})$, collisions will most likely occur (with constant probability).

[4] This is the reason that a PRP is also a PRF when the codomain is big enough.

Why call this the birthday problem? Assume each person's birthday (month & day) are uniform in 365 days of a year. How are there two people having the same birthday in a group of people? I claim if there are at least 23 people, then this will happen with probability at least 1/2.

Proof. Note that for each distinct pair $i \neq j$, $\Pr[Coll_{i,j}] = 1/N$.

$$
\begin{aligned}
\Pr[Coll] &= \Pr[\cup_{i \neq j} Coll_{i,j}] \\
&\leq \sum i \neq j \Pr[Coll_{i,j}] \quad \text{union bound} \\
&= \binom{q}{2} \cdot \frac{1}{N} \leq q^2/2N \,.
\end{aligned}
$$

□

Back to our discussion on finding collision in a random function, if we evaluate q distinct inputs, this amounts to sampling q times independently from the codomain $\{0,1\}^\ell$. Therefore when $q = \Theta(\sqrt{2^\ell})$, we will have at least 1/2 chance of finding a collision. To give you a concrete sense: to find a collision in a hash function of output length 256 bits, basically you only need to invest 2^{128} unit of computation resource. You might have heard statements that a system offers 128-bit of security. This means that breaking the system is roughly as difficult as exhaustively searching a 2^{128}-bit key space. [5]

There are other properties we need from a cryptographic hash function: Preimage resistant, second-preimage resistant, etc. Read the book and do the HW problems.

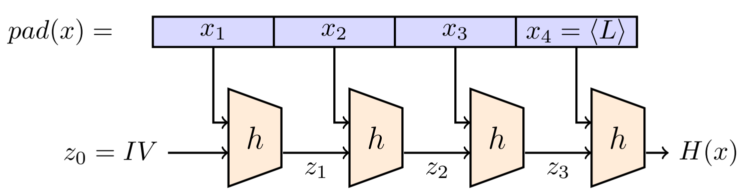
[5] Read [KL: 5.4.2] about how to reduce the memory cost of the birthday attack as well as finding meaningful collisions rather than an arbitrary one.

Constructing hash functions

We first show how to extend the domain of a function on a small domain (a compression function) to handle long messages. We then discuss a dominant approach in practice to construct compression functions from block ciphers.

Domain extension: Merkle-Damgård Transformation

Let h be a fixed-length hash function: $h : \{0,1\}^{2n} \to \{0,1\}^n$ (e.g., think of n as 128 bits). Construct H to handle variable-length inputs.



On input string x of length L

1. (**Padding**) Set $B := \lceil L/n \rceil$ i.e. number of blocks in x. Pad the last block with 0 to make it a full block. Denote the padded input $x_1, \ldots, x_B$, and let $x_{B+1} := \langle L \rangle$, i.e. the length represented as an n-bit string.
2. (IV) Set $z_0 := \mathsf{IV} = 0^n$.

Discussion in Class.

- $H(x) := h(x_1 \| x_2) \| h(x_3 \| h_4) \| \cdots$. Ignore the issue of variable-length output, is H collision resistant (assuming h is)?
- Picking a random IV? Hash function needs to be deterministic: the same message better produces the same digest no matter who and when hashes it. In SHA family, some peculiar IV rather than 0^n is used.
- Without encoding message length in last block? Explicit attack is possible depending on the compression function. Including the length makes the proof simple and universal. Read more at `https://eprint.iacr.org/2009/325`.

3. (**Cascading**) For $i = 1, \dots, B+1$, compute $z_i = h(z_{i-1} \| x_i)$.
4. Output z_{B+1}.

Theorem 4 ([KL: Thm. 5.4]). *If h is collision resistant, so is H.*

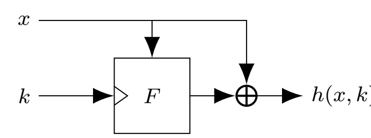
Proof skipped. Idea: use a collision in H to find one in h.

Compression functions from block ciphers: Davies-Meyer construction

How do we get compression functions on a small domain? Block ciphers are the hero again. [KL: Section 6.3]

Let F be a block cipher (PRP): $\{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$. Davies-Meyer proposed the following design.

$$\begin{aligned} h :& \{0,1\}^{n+\ell} \to \{0,1\}^\ell \\ & k \| x \mapsto F_k(x) \oplus x \,. \end{aligned}$$



Unfortunately, we don't know how to prove collision resistance of the Davies-Meyer compression function solely based on the assumption that F is a PRP. Instead, we resolve to an idealized model, *ideal cipher model*, which assumes that a random permutation and its inverse are publicly available as oracles to all users. We do not get into it in this course.

Examples

What to specify in a Merkle-Damgård hash function, e.g., SHA-256? Merkle-Damgård: IV and $h : \{0,1\}^{512} \to \{0,1\}^{256}$. Then h is a Davies-Meyer compression function, and hence we need to describe the block cipher E_{SHA256}.

Name	year	digest (bits)	block (bits)	best attack
MD4	1990	128	512	2^1
MD5	1992	128	512	2^{30}
SHA-0	1993	160	512	2^{39} (2005)
SHA-1	1995	160	512	2^{63} (2017)
SHA-2 (SHA-256)	2002	256	512	
SHA-2 (SHA-512)	2002	512	512	

Table 1: Mekle-Damgård hash functions

Full attack on SHA-1 https://shattered.io/

The new standard SHA-3, the Keccak family, is based on a very cute new design. Read more on `http://keccak.noekeon.org/`.

Application: Hash-and-MAC

A general paradigm: $S'(m) = S(H(m))$.

Theorem 5 (KL-Thm. 5.6). *If Π is a secure MAC, and H is a collision resistant hash function (for arb. length input), then Π' is a secure MAC for arb. length messages.*

This paradigm should not be used literally for two reasons.

1. In practice, hash functions have fixed small output length. Once one finds a collision offline, breaking any MAC scheme of this kind is trivial.
2. It relies on two primitives, a collision resistant hash and a secure MAC. It is preferable, from the implementation point of view, to rely on one primitive only.

It inspires the popular HMAC widely used on the Internet.