Disclaimer. Draft note. No guarantee on completeness nor soundness. Read with caution, and shoot me an email at fsong@pdx.edu for corrections/comments (they are always welcome!)

Logistics. Makeup office hr. Review PRF.
Last time. MAC from PRFs, domain extension
Today. Hash functions, MAC from hash

Review: PRF domain extension

Encrypted PRF.

- **ECBC**.

$$\mathsf{ECBC}_{k_1,k_2}(\cdot) := F_{k_2}(\mathsf{CBC}_{F_{k_1}}(\cdot)).$$

  Variants as ANSI (ANSI X9.9 and ANSI X9.19) and ISO (ISO 8731-1 and ISO/IEC 9797) standards.

- **Encrypted Cascade** a.k.a **NMAC** (Nested MAC). A variant of it (using a hash function instead of a PRF in the cascade construction) called HMAC is widely used in the Internet (rfc2104).

$$\mathsf{ECAS}_{k_1,k_2}(\cdot) := F_{k_2}(\mathsf{CASCADE}_{F_{k_1}}(\cdot)).$$

Theorem 1. *ECBC and NMAC are* PRF*s.*

Formal proofs are beyond the scope of this course. Read Boneh-Shoup Chapter 7 if interested.

[Note that both are **streaming** MACs, since we do not need to know the message length ahead of time.]

1 Hash functions

Today we introduce another basic primitive in cryptography – *hash functions*. Roughly they are functions that map long inputs to short *digests*. The primary requirement is to avoid *collision*, i.e., two inputs that map to the same digest. You've heard about hash functions in data structures to build a hash table that enables quick look up for an element. A "good" hash function would be on that introduces as few *collisions* as possible.

The basic idea is similar in the cryptographic setting, but with significant distinction. Therefore we sometimes say *cryptographic* hash functions to stress this.

- Collision resistant is a *requirement* rather than "better-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.

1.1 Definition

Consider keyed hash functions $H : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$: $H^s(x) := H(s,x)$. Here the key is usually not kept secret, so we write it in superscript.

Definition 2. A hash function is a pair of PPT (G, H)

- G: $s \leftarrow G(1^n)$.
- H: on input s and string $x \in \{0,1\}^*$, outputs $H^s(x) \in \{0,1\}^{\ell(n)}$.

If H^s is defined only for inputs $x \in \{0,1\}^{\ell' n}$ with $\ell'(n) > \ell(n)$, then we call H a compression function.

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

FS NOTE: Draw coll finding diagram

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

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

Definition 3. $\Pi = (G, H)$ is collision resistant if for any PPT $\mathcal{A}$

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

In practice, hash functions (e.g. SHA) are *unkeyed*. We consider keyed functions for technical reasons[1].

1.2 Generic attacks on hash functions

How hard is it to find collisions in a hash function? 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.

Let $H^s : \{0,1\}^* \to \{0,1\}^\ell$ be a hash function. Trivial attack: evaluate $2^\ell + 1$ distinct inputs, and there must be a collision. How about evaluation q elements, what is probability that there is a collision? We analyze it fora random function, and this leads us to the famous *birthday problem*.
The birthday problem.

[1] 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.

Choose q elements $y_1, \ldots, y_q$ from a set of size N, 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 4. $\Pr[Coll] = \Theta(q^2/N)$. *Specifically*

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

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.

[**TS**: Ask birthday of students: no year, just month and day] Let proof the upper bound. Read the lower bound proof in [KL: Appendix A.4].

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'd hear statements that a system gives 128-bit of security, meaning it is roughly as difficult as exhaustive search a 2^{128}-bit key space.

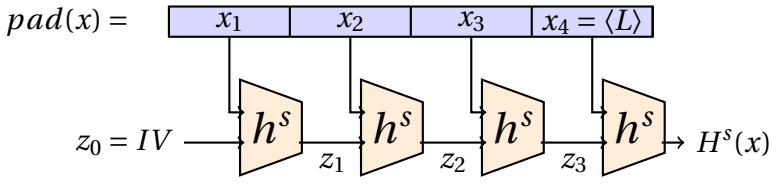
Read [KL: 5.4.2] about how to reduce the memory cost of the birthday attack as well as finding meaningful collisions rather than an arbitrary one.

2 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.

2.1 Domain extension: Merkle-Damgård Transformation

Let (G, h) be a fixed length hash function: $h^s : \{0,1\}^{2n} \to n$. Construct (G, H) to handle variable-length inputs.



On input s and 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 $z_0 := \mathsf{IV} = 0^n$.
3. (**Cascading**) For $i = 1, \ldots, B+1$, compute $z_i = h^s(z_{i-1} \| x_i)$.
4. Output z_{B+1}.

Theorem 5 (KL-Thm. 5.4)**.** *If (G, h) is collision resistant, so is (G, H).*

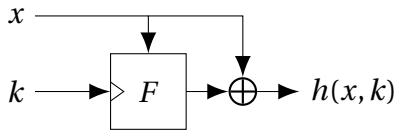
Proof skipped. Idea: use a collision in H to find one in h.

2.2 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): $\{0,1\}^n \times \{0,1\}^\ell \{0,1\}^\ell$. Davies-Meyer proposed the following design of a compression function $h : \{0,1\}^{n+\ell} \to \{0,1\}^\ell$.

$$h(k, x) := F_k(x) \oplus x\,.$$



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.

2.3 Examples

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

3 Application: Hash-and-MAC

A general paradigm:

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

Table 1: Mekle-Damgård hash functions

FS NOTE: Draw Hash-and-mac diagram
$S'(m) = S(H^s(m))$

Theorem 6 (KL-Thm. 5.6)**.** *If* Π *is a secure MAC, and* Π_H *is a collision resistant hash function (for arb. length input), then* Π' *is a secure MAC for arb. length messages.*
HMAC. This paradigm should not be used literally for two reasons.

1. In practice, hash functions have fixed small output length. One can find a collision offline, and as long as that happens, 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.

Here comes the popular HMAC widely used on the Internet. It's variant of the two-key NMAC.

$$t := H^s(k_2 \| H^s(k_1 \| m)).$$

FS NOTE: Draw HMAC diagram

$$\mathsf{ipad} := \text{byte } \texttt{0x36} \text{ repeated multiple times},$$
$$\mathsf{opad} := \text{byte } \texttt{0x5C} \text{ multiple times}$$

Connection to NMAC. Key distinction: derive two keys from one uniform key k: $k_1 = k \oplus \mathsf{ipad}$, and $k_2 := k \oplus \mathsf{opad}$. $k_{in} := h^s(\mathsf{IV} \| k_1)$ and $k_{out} := h^s(\mathsf{IV} \| k_2)$.
Connection to Hash-and-MAC paradigm.

$$\tilde{H}^s(m) := H^s(k_1 \| m), \quad \tilde{S}_k(y) := h^s(k \| \hat{y})$$
$$HMAC_{s,k}(m) = \tilde{S}_{k_{out}}(\tilde{H}^s(m))$$

Theorem 7 (KL-Thm. 5.8)**.** *If* k_{in} *and* k_{out} *are pseudorandom,* $\tilde{S}$ *is a secure fixed-length MAC, then HMAC is secure.*

Common used ones: HMAC-SHA1 and HMAC-SHA-256.
FS NOTE: Draw HMAC diagram [KL: Fig. 5.2]

4 The Random oracle model

An idealized model to design and analyze crypto constructions involving hash functions. Hash functions are so efficient, but often we cannot prove security based solely on Collision resistance or other reasonable assumptions. Instead, settle with a "middle-land".

Random oracle $\mathcal{O}(\cdot)$: *a publicly available black-box that implements a random function.* Everyone, including an adversary, can only query $\mathcal{O}(\cdot)$ for evaluating $y = \mathcal{O}(x)$. Note that a PRF is not a RO.

The random oracle *heuristic*

> If a "natural" scheme is designed and proven secure in the random-oracle model, then we can *instantiate* the RO with a "nice" hash function and the resulting real-world scheme remains secure.

This is only a heuristic, since we do not know what "natural" and "nice" should mean. Unfortunately, there are exammples that violates this heuristic, but you may argue these examples are "unnatural". RO heuristic is still under active debate and research in the crypto community.

Next time, we'll talk about some useful properties of the RO model that makes constructing and analyzing cryptosystems easier (and more efficient).