Today: Hash functions (Cont.)

1. **Recap: Def and applications**
2. **Constructions: Sponge construction (SHA3)**

See Section 8 in the Applied Cryptography book by Boneh-Damgard

Definition: A **hash function** $H: \{0,1\}^* \to \{0,1\}^k$ maps strings of arbitrary length to strings of length k.

A hash function is deterministic, efficient, and public (**no secret keys**).

Last class: We saw several **applications**:

1. Authenticating long files via a short hash
2. password storage
3. hash-&-sign
4. The Fiat-Shamir paradigm
5. commitment scheme

Application: Commitment Scheme

A commitment scheme is a digital analogue of a locked box.

It is a randomized function $Com: M \times \{0,1\}^k \to C$

where M is the message space and C is the set of possible commitments.

It should satisfy the following two security requirements:

Statistical Binding: There do not exist distinct msgs $m_1, m_2 \in M$ and $r_1, r_2 \in \{0,1\}^k$ s.t.

$$Com(m_1, r_1) = Com(m_2, r_2)$$

Computational Hiding: For every $m_1, m_2 \in M$,

$$Com(m_1, r_1) \approx Com(m_2, r_2)$$

for random $r_1, r_2 \leftarrow \{0,1\}^k$

One can switch the requirements to require computational hiding and statistical binding:

Computational Binding: It is **computationally hard** to find distinct $m_1, m_2 \in M$ and $r_1, r_2 \in \{0,1\}^k$ s.t.

$$Com(m_1, r_1) = Com(m_2, r_2)$$

Statistical Hiding: For every $m_1, m_2 \in M$,

$$Com(m_1, r_1) \equiv Com(m_2, r_2)$$

for random $r_1, r_2 \leftarrow \{0,1\}^k$, where $\equiv$ denotes statistical closeness

Definition: A family of distributions $\{D_k\}$ and $\{D_k'\}$ are **statistically close** if there exists a negligible function μ s.t. for any (all powerful) A and for every $\mathrm{k} \in N$,

$$\Pr[A(x) = 1] - \Pr[A(x') = 1]| \leq \mu(k)$$

where $x \leftarrow D_k$ and $x' \leftarrow D_k'$

Construction: $\boldsymbol{Com(m, r) = H(m||r)}$.

In the ROM this commitment scheme is statistically hiding, assuming $M = \{0,1\}^k$, and is computationally binding.

To get computational binding collision resistance suffices.

Constructions of hash functions: Common design

Step 1: Construct $\boldsymbol{H_{small}}: \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}} \rightarrow \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{k}}$

for some $n \gg k$ (e.g., $n = 2k$ and $k = 256$).

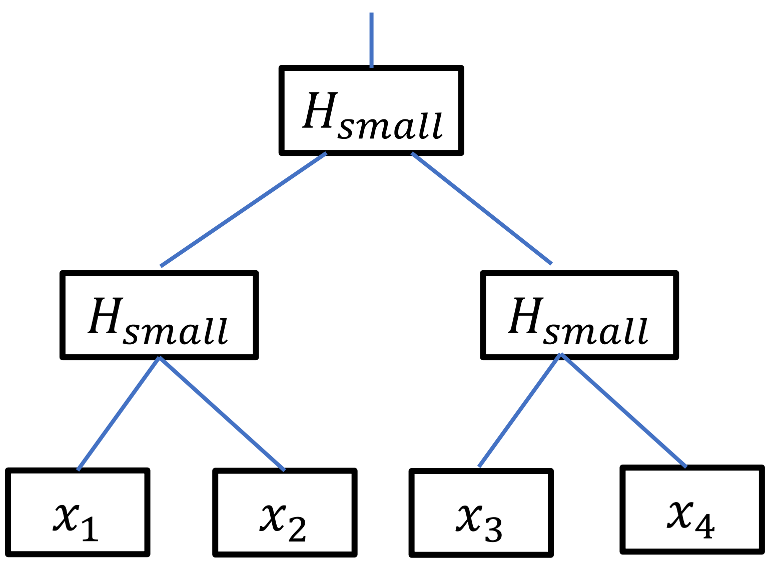
This step is an "engineering" step.

(Come up with a candidate, try to break it, come up with an improved candidate…)

Step 2: Use H_{small} to construct $H: \{0,1\}^* \rightarrow \{0,1\}^k$.

Implementing Step 2 using Merkle-Hash:

Suppose we are given $H_{small}: \{0,1\}^{2k} \rightarrow \{0,1\}^k$



The output contains the value of the root and the depth of this tree (i.e., the input length).

Padding: We assume that the msg $x = (x_1, \dots, x_t)$ is of length that is a multiple of $2^{\ell} \cdot k$ for some $\ell \in N$.

If this is not the case, then pad x.

Padding should be done carefully, to ensure that it is invertible.

Example: $PAD(x) = (x, 1{,}0^*)$.

Don't implement yourself!

Claim: If H_{small} is collision resistant then so is $H: \{0,1\}^* \rightarrow \{0,1\}^k$

"Proof": Suppose someone found a collision in H, i.e., found distinct x, y such that $H(x) = H(y)$. Note that it must be that $|x| = |y|$.

Note that the values of the root agree, since $H(x) = H(y)$, whereas the values of the input layer differ since $x \neq y$.

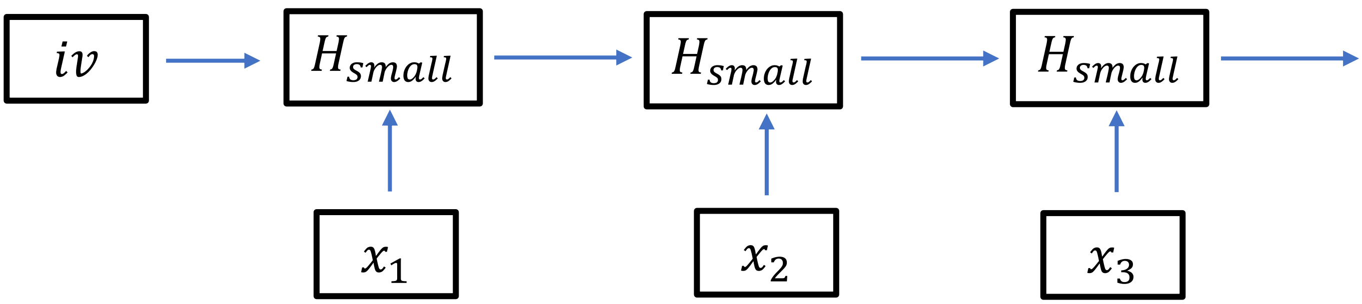
Consider the layer closest to the root s.t. the hashes corresponding to x differ from the hashes corresponding to y.

These hash values can be used as collisions to H_{small}.

Alternative construction: Merkle-Damgard

Given $H_{small}: \{0,1\}^n \to \{0,1\}^k$ where $n > k$, compute $H: \{0,1\}^* \to \{0,1\}^k$ as follows:

Given $x \in \{0,1\}^*$, first pad x so that $|x| = t \cdot (n-k)$ for some t $\in N$. Partition $x = (x_1, \dots, x_t)$, where $|x_i| = n - k$



The initial value iv can be set to be the all zero string of size k.

Claim: If H_{small} is collision resistant then so is $H: \{0,1\}^* \to \{0,1\}^k$

"Proof": Similar to that of the Merkle hash construction.

This construction is not parallelizable (unlike Merkle hash)!

Constructing H_{small}

History:

1990/1991: First standardized construction: **MD4** and **MD5** by Ron Rivest (MD = Message Digest).

It has a 128-bit output.

2007: Broken in time 2^{24}.

1993: NSA designed hash function **SHA1**

(SHA = Secure Hash Algorithm)

It has a 160-bit output.

2017: Broken in time 2^{63}.

2001: NSA designed **SHA2**

NIST Competitions: SHA3 (2015)

SHA2 is not broken and SHA3 was standardized to have a backup in case SHA2 breaks.

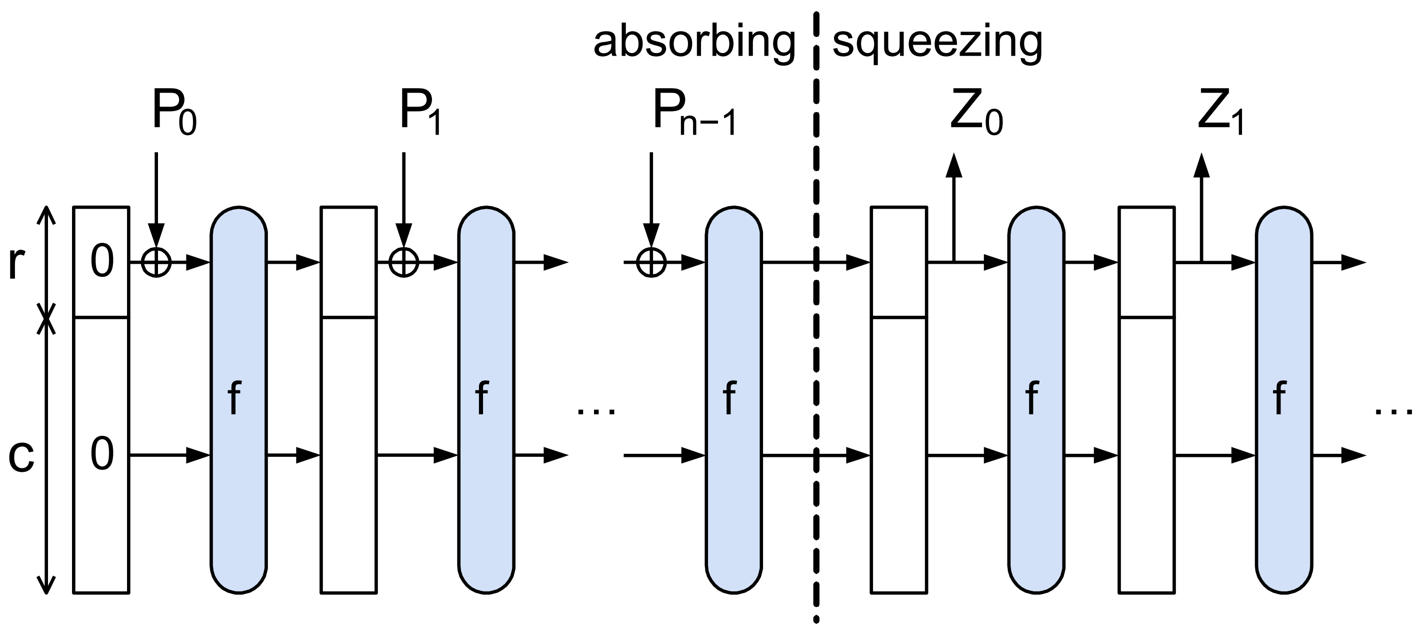
SHA3 – Sponge construction: (Section 8.8 in Boneh-Shoup Book)

Different than the MD5-like structure of SHA1 and SHA2.

The sponge construction is based on a permutation f.

It takes as input message of **arbitrary** length, and outputs a message of **arbitrary** length, while being "pseudorandom".

It is called a sponge since it absorbs any amount of data and squeezes out any amount of data.



r = rate, c = capacity. $n = r + c$.

Larger r implies better efficiency, larger c implies better security.

SHA3 is associated with a permutation $f: \{0,1\}^n \to \{0,1\}^n$

where $\mathrm{n} = r + c = 1600.$

We will not describe f here, but it is engineered to look random.

In the security analysis of SHA3 it is assumed to be an ideal random permutation.

To hash a message $\boldsymbol{m}$:

First pad m so that its length is a multiple of r.

Let $m = (P_0, \ldots, P_{t-1})$, where $P_i \in \{0,1\}^r$.

Abosorb all blocks P_i of a padded input string as follows:

- The initial state $S = (R, C) \in \{0,1\}^n$ is initialized to zero
- For each block P_i
  - Replace R with $R \oplus P_i$ and update $S = (R, C)$.
  - Replace S with $f(S)$

The sponge function output is now ready to be produced ("squeezed out") as follows:

- Repeat
  - Output the *R* portion of *S*
  - *S* is replaced by *f*(*S*) unless the output is full

The permutation f chosen in SHA3 is the Keccak permutation, which sets $n = 1600$ (where recall that n is the input and output lengths of f. (We will not describe f here.)

It has several possible settings for r and c, depending on the security and efficiency tradeoffs that are desired.

Example: SHA3(256) takes $c = 512$ and $r = 1088$. It has a fixed output length of 256 bits.

There are other SHA3 instantiations with different parameter settings and with variable input length.