Cryptographic Hash Function

Elena Kirshanova

Course “Information and Network Security”
Lecture 6
12 апреля 2020 г.

Agenda

Last time:

- Achieve message integrity using MACs
- Construction of MACs from block ciphers. Example: CBC-MAC

Today:
Construct a more efficient MAC using hash functions (HMAC)

Cryptographic Hash Function: definition

A Hash function is a pair of polynomial time algorithms $(\mathsf{Gen}, \mathcal{H})$:

1. Probabilistic Gen : $s \leftarrow \mathsf{Gen}(1^\lambda)$
2. Deterministic $\mathcal{H}_s : \{0,1\}^\star \rightarrow \{0,1\}^\ell$.

Cryptographic Hash Function: definition

A Hash function is a pair of polynomial time algorithms $(\mathsf{Gen}, \mathcal{H})$:

1. Probabilistic Gen : $s \leftarrow \mathsf{Gen}(1^\lambda)$
2. Deterministic $\mathcal{H}_s : \{0,1\}^\star \rightarrow \{0,1\}^\ell$.

Most important property of $\mathcal{H}_s$ is collision resistant:
Given s, there is no efficient adversary who can find two inputs $x, x'(x! = x')$ to $\mathcal{H}_s$ s.t.

$$\mathcal{H}_s(x) = \mathcal{H}(x')$$

! A "hash" function in general sense does not necessarily has this property. A cryptographic hash function must be collision resistant.

There are many collisions for $\mathcal{H}_s$, but it must be hard to find any.

Properties of cryptographic hash function

- Pre-image resistance (or one-wayness)

Given (s, y)
Find x s.t. $\mathcal{H}_s(x) = y$

A collision resistant hash function is also pre-image resistant

Properties of cryptographic hash function

I Pre-image resistance (or one-wayness)

Given (s, y)
Find x s.t. $\mathcal{H}_s(x) = y$

A collision resistant hash function is also pre-image resistant

II 2^{nd} Pre-image resistance

Given (s, x)
Find $x'! = x$ s.t. $\mathcal{H}_s(x) = \mathcal{H}_s(x')$

A collision resistant hash function is also 2^{nd} pre-image resistant

Conclusion: Collision resistance is the strongest requirement

A word of caution: Exotic property of hash functions

In BitCoind world the above three properties: **pre-image resistance**, 2^{nd} **pre-image resistance**, **collision resistance** may have other names: **"hiding"**, **"puzzle friendliness"**, **collision resistance**.

These are not special properties! BitCoin uses standardized cryptographic hash function (wait until the end of the lecture).

See e.g. Section 1.1. in `https://d28rh4a8wq0iu5.cloudfront.net/bitcointech/readings/princeton_bitcoin_book.pdf?a=1`

Generic attack on any hash function: birthday paradox

Remainder: Let $h_1, h_2, \ldots, h_n \in \{0,1\}^\ell$ be independent identically distributed bit strings. Then Birthday paradox says that

$$\text{For } n = \mathcal{O}\left(\sqrt{|\{0,1\}^\ell|}\right) = \mathcal{O}\left(2^{\ell/2}\right) \quad \Pr[\exists(i! = j) \ : \ h_i = h_j] > 1/2.$$

Generic attack on any hash function: birthday paradox

Remainder: Let $h_1, h_2, \ldots, h_n \in \{0,1\}^\ell$ be independent identically distributed bit strings. Then Birthday paradox says that

For $n = \mathcal{O}\left(\sqrt{|\{0,1\}^\ell|}\right) = \mathcal{O}\left(2^{\ell/2}\right) \quad \Pr[\exists (i! = j) \ : \ h_i = h_j] > 1/2.$

Generic algorithm finds a collision in $\mathcal{O}(2^{\ell/2})$ hash evaluations:

1. Choose $2^{\ell/2}$ random bit strings (messages) $m_1, \ldots, m_{2^{\ell/2}}$
2. For each m_i compute $h_i = \mathcal{H}_s(m_i)$, sort pairs to (h_i, m_i) w.r.t. h_i
3. Find in the sorted list $h_i = h_j$. A collision (m_i, m_j).

Birthday paradox ensures that the above algorithm succeeds with constant success probability.

Conclusion: Require $\ell \geq 160$.

Real world hash functions: historical overview

1. 1980s: MD4 (Message Digest) by R. Rivest. $\ell = 128$
   Status: Broken. A collision can be found within seconds

Real world hash functions: historical overview

1. 1980s: MD4 (Message Digest) by R. Rivest. $\ell = 128$
   Status: Broken. A collision can be found within seconds

2. 1990: MD5. $\ell = 128$
   Status: Broken. A collision can be found within seconds

Real world hash functions: historical overview

1. 1980s: MD4 (Message Digest) by R. Rivest. $\ell = 128$
   Status: Broken. A collision can be found within seconds
2. 1990: MD5. $\ell = 128$
   Status: Broken. A collision can be found within seconds
3. 1995: SHA-1 (Secure Hash Algorithm 1) $\ell = 160$
   Status: Broken. See `https://shattered.io/` for two PDFs with the same SHA-1 values.
   Caution: may still be used by some systems (i.e., GIT).

Real world hash functions: historical overview

1. 1980s: MD4 (Message Digest) by R. Rivest. $\ell = 128$
   Status: Broken. A collision can be found within seconds
2. 1990: MD5. $\ell = 128$
   Status: Broken. A collision can be found within seconds
3. 1995: SHA-1 (Secure Hash Algorithm 1) $\ell = 160$
   Status: Broken. See `https://shattered.io/` for two PDFs with the same SHA-1 values.
   Caution: may still be used by some systems (i.e., GIT).
4. 2001: SHA-2 (SHA-256, SHA-384, SHA-512). $\ell = 256, 348, 512$
   Status: Considered secure

Real world hash functions: historical overview

1. 1980s: MD4 (Message Digest) by R. Rivest. $\ell = 128$
   Status: Broken. A collision can be found within seconds
2. 1990: MD5. $\ell = 128$
   Status: Broken. A collision can be found within seconds
3. 1995: SHA-1 (Secure Hash Algorithm 1) $\ell = 160$
   Status: Broken. See `https://shattered.io/` for two PDFs with the same SHA-1 values.
   Caution: may still be used by some systems (i.e., GIT).
4. 2001: SHA-2 (SHA-256, SHA-384, SHA-512). $\ell = 256, 348, 512$
   Status: Considered secure
5. 2012: SHA-3 (Keccak). SHA-$3 - 224/256/348/512$.
   Status: Considered secure

Real world hash functions: historical overview

1. 1980s: MD4 (Message Digest) by R. Rivest. $\ell = 128$
   Status: Broken. A collision can be found within seconds
2. 1990: MD5. $\ell = 128$
   Status: Broken. A collision can be found within seconds
3. 1995: SHA-1 (Secure Hash Algorithm 1) $\ell = 160$
   Status: Broken. See `https://shattered.io/` for two PDFs with the same SHA-1 values.
   Caution: may still be used by some systems (i.e., GIT).
4. 2001: SHA-2 (SHA-256, SHA-384, SHA-512). $\ell = 256, 348, 512$
   Status: Considered secure
5. 2012: SHA-3 (Keccak). SHA-$3 - 224/256/348/512$.
   Status: Considered secure

In Russia

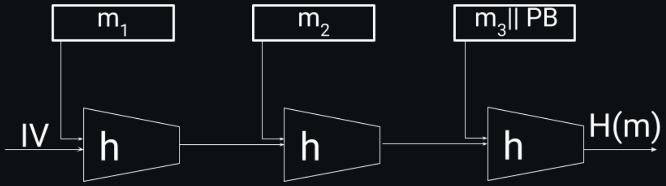
1. GOST R 34.11-94 and GOST 34.311-95. $\ell = 256$
   Status: Depricated Collision in 2^{105} time
2. GOST R 34.11-2012. Streebog $\ell = 256, 512$
   Status: Should be used in certified products

Construction of a hash function: Merkle-Damgård paradigm

Given a compression function (will be defined later)

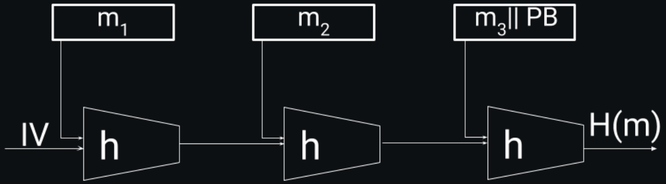
$$h : \mathcal{K} \times \mathcal{M} \to \mathcal{K}$$

Construct $\mathcal{H} : \mathcal{M}^\star \to \mathcal{K}$
Let $m = (m_1, m_2, m_3)$ of arbitrary length.



IV - Initial Value (fixed for given hash function)
PB - Padding Block $[100\ldots0||$mes. length$]$. If **PB** does not fit add another block

Security of Merkle-Damgård construction



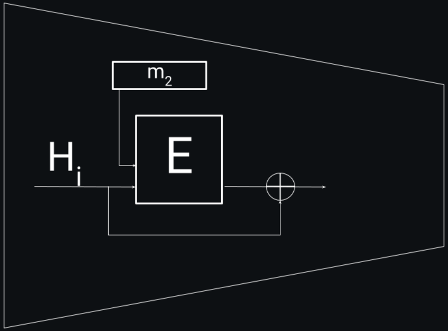
Theorem: If h is collision resistant so is H.

Construction of compressing function h

$\mathsf{Enc} : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ – a block-cipher.

Davies-Meyer construction:

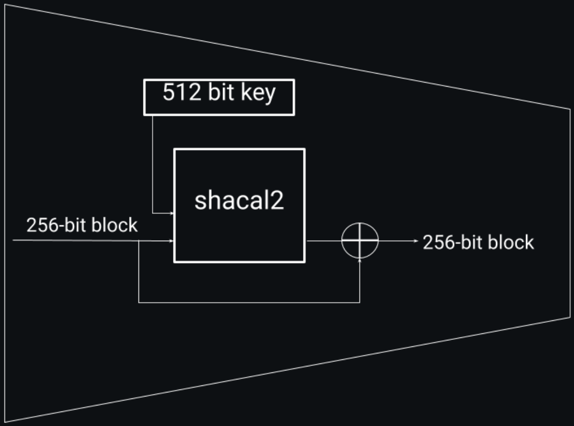
$$h(H_i, m) = \mathsf{Enc}(H_i, m) \oplus H_i$$



Theorem (Informal): If Enc is a "good" cipher (i.e., Enc is a random permutation for fixed $k \in \mathcal{K}$), then finding a collision $h(H, m) = h(H', m')$ takes $2^{n/2}$ evaluations of $(\mathsf{Enc}, \mathsf{Dec})$.

Example: SHA-256

In SHA-256 the compression function is:



Merkle-Damgård construction is used to allow for arbitrary message length.

Alternative construction of h

Davies-Meyer construction:

$$h(H, m) = \mathsf{Enc}(H, m) \oplus H$$

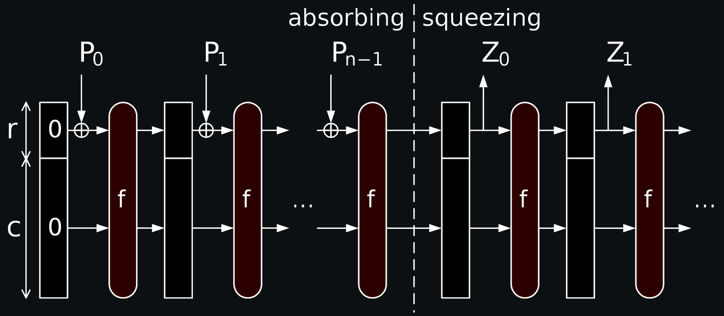
Miyaguchi–Preneel constriction:

$$h(H, m) = \mathsf{Enc}(H, m) \oplus H \oplus m$$

Other variants of combinations of Enc, H, m exist. Not all combination are secure!
GOST P 34.11-2012 (Streebog) uses Miyaguchi–Preneel.

Sponge construction: SHA-3

SHA-3 (Keccak) is not based on compression function. It is a Sponge (рус. Губка) construction.
$P_0, \ldots P_{n-1}$ are derived from the input message. $Z_0, Z_1, \ldots$ is the output



The block transformation f is a permutation consisting of 5 primitive function (small permutations, bitwise operations).

Hash functions in BitCoin

Basic concept in BitCoint: Proof of Work (PoW)
Intuition: if a user has computing power $\implies$ he should be able to prove it via doing some work

- PoW introduced to crypto by Dwork & Naor (1992) as a countermeasure against spam
- Idea: force users to solve some “moderately hard” puzzle (a solution should be fast to verify)

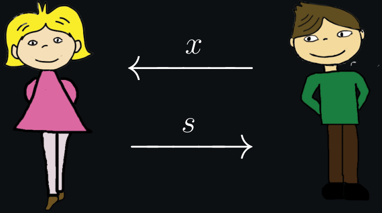
Hash functions in BitCoin: constructing PoW

Main primitive: cryptographic hash function $\mathcal{H} : \{0,1\}^\star \to \{0,1\}^\ell$ that takes $T(\mathcal{H})$ time to evaluate

Alice
Prover

Bob
Verifier

$x \in \{0,1\}^\star$



Finds $s \in \{0,1\}^\star$
s.t. $\mathcal{H}(s||x)$
starts with n 0's

Time: $2^n T(\mathcal{H})$

Checks if
$\mathcal{H}(s||x)$ has n 0's
Time: $T(\mathcal{H})$

For a cryptographic hash function $\mathcal{H}$ Alice cannot do better than brute-force over s. This is a pre-image search.