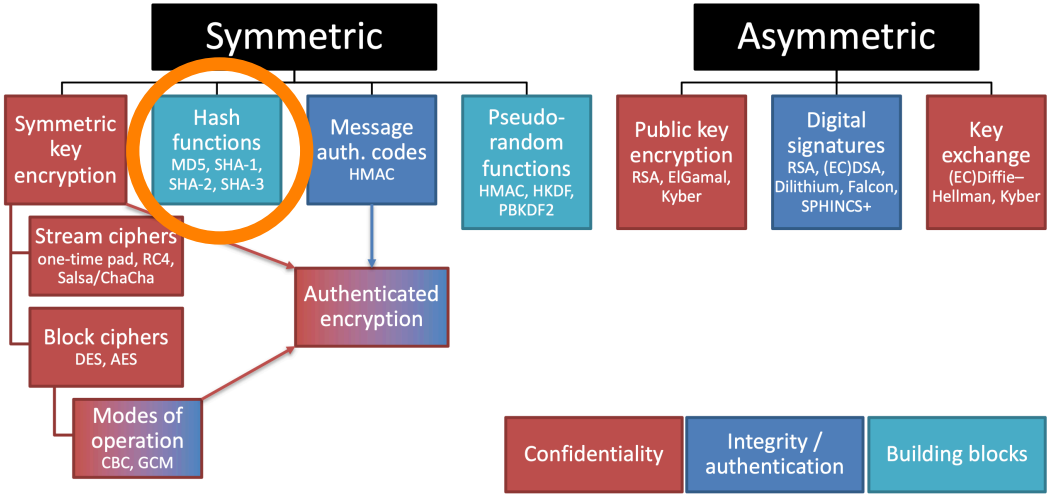
Topic 2.1
Integrity – Hash functions

Douglas Stebila

CO 487/687: Applied Cryptography
Fall 2024

Map of cryptographic primitives

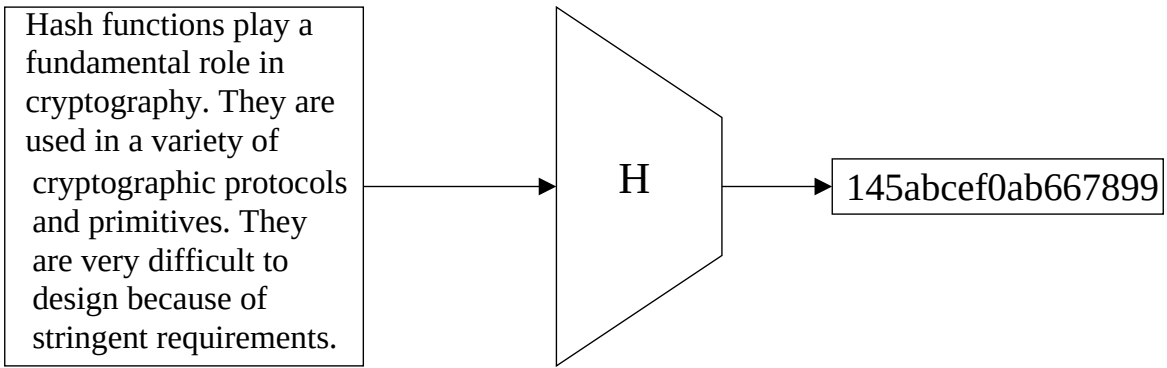


Outline

What is a hash function?

- Hash functions play a fundamental role in cryptography.
- They are used in a variety of cryptographic primitives and protocols.
- They are very difficult to design because of very stringent security and performance requirements.
- The main options available today are: SHA-224, SHA-256, SHA-384, SHA-512, and SHA-3.

What is a hash function?



A hash function is a *checksum* designed to be safe from *malicious tampering*.

Definition of a Hash Function

Definition (Hash function)

A hash function is a mapping H such that:

1. H maps inputs of arbitrary lengths to outputs of length n, where n is fixed (H: $\{0,1\}^* \to \{0,1\}^n$)
   (More generally, H maps elements of a set S to a set T where $|S| > |T|$.)
2. $H(x)$ can be efficiently computed for all $x \in \{0,1\}^*$.

H is called an n-bit hash function.

$H(x)$ is called the hash value, hash, or message digest of x.

Note: The description of a hash function is public. There are no secret keys.

Toy Hash Function $H : \{0,1\}^{\leq 4} \longrightarrow \{0,1\}^2$

x	$H(x)$	x	$H(x)$	x	$H(x)$	x	$H(x)$
0	00	1	01				
00	11	01	01	10	01	11	00
000	00	001	10	010	11	011	11
100	11	101	01	110	01	111	10
0000	00	0001	11	0010	11	0011	00
0100	01	0101	10	0110	10	0111	01
1000	11	1001	01	1010	00	1011	01
1100	10	1101	00	1110	00	1111	11

- (00,1000) is a collision.
- 1001 is a preimage of 01.
- 10 is a second preimage of 1011.

Example: SHA256

SHA256: $\{0,1\}^* \longrightarrow \{0,1\}^{256}$

SHA256(“Hello there”) =

`0x4e47826698bb4630fb4451010062fadbf85d61427cbdfaed7ad0f23f239bed89`

SHA256(“Hello There”) =

`0xabf5dacd019d2229174f1daa9e62852554ab1b955fe6ae6bbbb214bab611f6f5`

SHA256(“Welcome to CO 487”) =

`0x819ba4b1e568e516738b48d15b568952d4a35ea73f801c873907d3ae1f5546fb`

SHA256(“Welcome to CO 687”) =

`0x404fb0ee527b8f9f01c337e915e8beb6e03983cfd9544296b8cf0e09c9d8753d`

Typical Cryptographic Requirements (informally)

- Preimage resistance: Hard to invert given just an output.
- 2nd preimage resistance: Hard to find a second input that has the same hash value as a given first input.
- Collision resistance: Hard to find two different inputs that have the same hash values.

Preimage resistance

Definition (Preimage resistance)

Let m be a positive integer. We say that H is *preimage resistant* for messages of length m if, given $y = H(x)$ for x picked uniformly at random from $\{0,1\}^m$, it is computationally infeasible to find (with non-negligible success probability) any input z such that $H(z) = y$.

- z is called a preimage of y.
- Some books use different definitions, which are intuitively similar but may have slightly different implications formally.

Second preimage resistance

Definition (Second preimage resistance)

Let m be a positive integer. We say that H is *second preimage resistant* for messages of length m if, given an input $x \in_R \{0,1\}^m$, it is computationally infeasible to find (with non-negligible probability of success) a second input $x' \neq x$ such that $H(x) = H(x')$.

- $x \in_R \{0,1\}^m$ means x chosen uniformly at random from $\{0,1\}^m$.

Collision resistance

Definition (Collision resistance)

It is computationally infeasible to find (with non-negligible probability of success) two distinct inputs x, x' such that $H(x) = H(x')$.

- The pair (x, x') is called a collision for H.

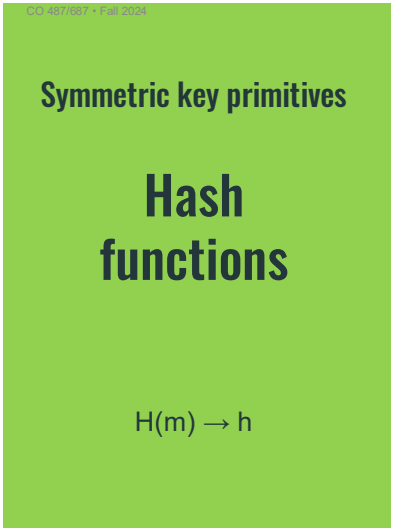
How are second preimage resistance and collision resistance different?

- Collision resistance: the attacker has freedom to pick both x and x'
- Second preimage resistance: the attacker given some x and has to find an x'

Terminology

- A hash function that is preimage resistant is sometimes called a one-way hash function (OWHF).
- A hash function that is preimage-, second preimage-, and collision-resistant is called a cryptographic hash function.

4+8+7 things to remember from CO 487



- General building block.
- Security goals: preimage resistance (hard to invert), second preimage resistance, collision resistance.
- Secure options as of 2024:
  - SHA2 family (SHA-256, SHA-384, SHA-512)
  - SHA-3 (Keccak)

Some Applications of Hash Functions

1. Password protection on a multi-user computer system:
   - Server stores (userid, H(password)) in a password file. Thus, if an attacker gets a copy of the password file, she does not learn any passwords.
   - Requires preimage-resistance.
   - See Topic 2.4 for an investigation of this application

2. Modification Detection Codes (MDCs).
   - To ensure that a message m is not modified by unauthorized means, one computes $H(m)$ and protects $H(m)$ from unauthorized modification.
   - e.g. virus protection.
   - Requires 2nd preimage resistance.

Some Applications of Hash Functions

3. Message digests for digital signature schemes:
   - For reasons of efficiency, instead of signing a (long) message, the (much shorter) message digest is signed.
   - Requires preimage-resistance, 2nd preimage resistance and collision resistance.
   - To see why collision resistance is required:
     - Suppose that Alice can find two messages x_1 and x_2, with $x_1 \neq x_2$ and $H(x_1) = H(x_2)$.
     - Alice can sign x_1 and later claim to have signed x_2.

4. Message Authentication Codes (MACs).
   - Provides data integrity & data origin authentication.

Some Applications of Hash Functions

5. Pseudorandom bit generation:
   - Distilling random bits s from several "random" sources $x_1, x_2, \ldots, x_t$.
   - Output $s = H(x_1, x_2, \ldots, x_t)$.
   - Used in OpenSSL and `/dev/random`

6. Key derivation function (KDF): Deriving a cryptographic key from a shared secret.

Notes:

- Collision resistance is not always necessary.
- Depending on the application, other properties may be needed, for example 'near-collision resistance', 'partial preimage resistance', etc.

Outline

The contrapositive proof technique in cryptography

In cryptography we often want to show theorems like the following:

Theorem

If scheme S satisfies security property X, then scheme T (built using S) satisfies security property Y.

Another way of phrasing this is:

Theorem

If there is no efficient adversary that breaks security property X of scheme S, then there is no efficient adversary that breaks security property Y of scheme T.

The contrapositive proof technique in cryptography

Theorem

If there is no efficient adversary that breaks security property X of scheme S, then there is no efficient adversary that breaks security property Y of scheme T.

It is often more convenient to prove the (equivalent) contrapositive statement:

Theorem

Suppose there exists an efficient adversary B that breaks security property Y of scheme T. Then there exists an efficient adversary A (that uses B) that breaks security property X of scheme S.

To prove the contrapositive statement, our task is come up with the algorithm A, assuming that algorithm B exists with the desired property.

Collision resistance implies 2nd preimage resistance

Theorem

Collision resistance implies 2nd preimage resistance.

Written slightly differently:

Theorem

If H is collision resistant, then H is 2nd preimage resistant.

Written in the contrapositive form:

Theorem

If there exists and efficient algorithm B that breaks the 2nd preimage resistance of H, then there exists an efficient algorithM A that breaks the collision resistants of H.

Collision resistance implies 2nd preimage resistance

Theorem

If there exists and efficient algorithm B that breaks the 2nd preimage resistance of H, then there exists an efficient algorithM A that breaks the collision resistants of H.

Proof. Suppose B is an efficient algorithm that breaks 2nd preimage resistance of H for messages of length m.

Algorithm A proceeds as follows.

Select $x \in_R \{0,1\}^m$.

Now, since H is not 2nd preimage resistant for messages of length m, we can use B to efficiently find $x' \in \{0,1\}^*$ with $x' \neq x$ and $H(x') = H(x)$.

Thus, we have efficiently found a collision (x, x') for H.

Hence H is not collision resistant. □

The degenerate counterexample proof technique in cryptography

In cryptography we sometimes want to show theorems like the following:

Theorem

Security property X does not imply security property Y.

Another way of phrasing this is:

Theorem

There exists a scheme S that has security property X but does not have security property Y.

Often we prove this by taking a generic scheme S that has property X, then construct an "artificially degenerate" modified scheme $\overline{S}$ that still has property X, but clearly doesn't has property Y.

2nd preimage resistance does not imply collision resistance

Theorem

2nd preimage resistance does not guarantee collision resistance.

Proof. Let $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ be a 2nd preimage resistant hash function for messages of length m.

Consider $\overline{H} : \{0,1\}^* \longrightarrow \{0,1\}^n$ defined by:

$$\overline{H}(x) = \begin{cases} H(0^m), & \text{if } x = 1^m \\ H(x), & \text{if } x \neq 1^m. \end{cases}$$

- $\overline{H}$ is still 2nd preimage resistant:
  Suppose the random 2nd preimage challenge is $x \in \{0,1\}^m$. With high probability $x \neq 0^m$. So if an given adversary finds x' such that $\overline{H}(x) = \overline{H}(x')$, then x' is also a 2nd preimage of x in H.

2nd preimage resistance does not imply collision resistance

Theorem

2nd preimage resistance does not guarantee collision resistance.

Proof. Let $H : \{0,1\}^* \longrightarrow \{0,1\}^n$ be a 2nd preimage resistant hash function for messages of length m.

Consider $\overline{H} : \{0,1\}^* \longrightarrow \{0,1\}^n$ defined by:

$$\overline{H}(x) = \begin{cases} H(0^m), & \text{if } x = 1^m \\ H(x), & \text{if } x \neq 1^m. \end{cases}$$

- $\overline{H}$ is not collision resistant:
  $\overline{H}(0^m) = \overline{H}(1^m)$

□

Collision resistance does not imply preimage resistance

Theorem

Collision resistance does not guarantee preimage resistance.

Proof. Let $H : \{0,1\}^* \to \{0,1\}^n$ be a collision-resistant hash function.

Consider $\overline{H} : \{0,1\}^* \to \{0,1\}^{n+1}$ defined by:

$$\overline{H}(x) = \begin{cases} 1\|x, & \text{if } x \in \{0,1\}^n \\ 0\|H(x), & \text{if } x \notin \{0,1\}^n. \end{cases}$$

- $\overline{H}$ is collision-resistant. (Sketch: No collisions on outputs starting with 1. Collisions on outputs starting with 0 imply collisions in H.)
- $\overline{H}$ is not preimage-resistant for messages of length n. (Easy to invert outputs starting with 1.)

□

Collision resistance and randomness implies preimage resistance

However, if H is "somewhat uniform" (i.e., all hash values have roughly the same number of preimages), and the message space much larger than the output space, then the following argument shows that collision resistance of H does indeed guarantee preimage resistance:

Suppose that H is not preimage resistant. Select random $x \in \{0,1\}^m$ for some $m \geq 2n$ and compute $y = H(x)$. Find a preimage x' of y, this is successful with some non-negligible probability. Then, if H is uniform, we expect that $x' \neq x$ with very high probability. Thus, we have efficiently found a collision (x, x') for H, hence H is not collision resistant.

Popular hash functions

- MD5 (RSA Laboratories; 1991) [128 bits]
- SHA-1 (NIST/NSA; 1995) [160 bits]
- SHA-2 (NIST/NSA; 2001) [various bit lengths]
  - SHA-224, SHA-256, SHA-384, SHA-512
- SHA-3 (Bertoni, Daemen, Peeters, Van Assche; 2012) [various bit lengths]
  - SHA3-224, SHA3-256, SHA3-384, SHA3-512
  - SHAKE128, SHAKE256, ...

Less widely used hash functions: MD2, MD4, RIPEMD160, Tiger

The following are NOT cryptographic hash functions: "Hash tables", CRC8, CRC16, CRC32, ...

Outline

Generic Attacks

- A generic attack on hash functions $H : \{0,1\}^* \to \{0,1\}^n$ does not exploit any properties a specific hash function may have.
- In the analysis of a generic attack, we view H as a random function in the sense that for each $x \in \{0,1\}^*$, the value $y = H(x)$ was chosen by selecting y uniformly at random from $\{0,1\}^n$ (written $y \in_R \{0,1\}^n$).
- From a security point of view, a random function is an ideal hash function. However, random functions are not suitable for practical applications because they cannot be compactly stored.

Generic Attack for Finding Preimages

Generic Attack for Finding Preimages

Given $y \in \{0,1\}^n$, repeatedly select distinct $x' \in \{0,1\}^*$ until $H(x') = y$.

- Expected number of steps is $\approx 2^n$.
  (Here, a step is a hash function evaluation.)
- This attack is infeasible if $n \geq 128$.

Note: It has been proven that this generic attack for finding preimages is optimal, i.e., no better *generic* attack exists.

Generic Attack for Finding Collisions

Generic Attack for Finding Collisions

Repeatedly select arbitrary distinct $x \in \{0,1\}^*$ and store $(H(x), x)$ in a table sorted by first entry. Continue until a collision is found.

- Expected number of steps: $\sqrt{\pi 2^n/2} \approx \sqrt{2^n}$ (by birthday paradox). (Here, a step is a hash function evaluation.)
- It has been proven that this generic attack for finding collisions is optimal in terms of the number of hash function evaluations.
- Expected space required: $\sqrt{\pi 2^n/2} \approx \sqrt{2^n}$.
- This attack is infeasible if $n \geq 160$.
- If $n = 128$:
  - Expected running time: 2^{64} steps. [barely feasible]
  - Expected space required: 7×10^8 Tbytes. [infeasible]

The birthday paradox

Theorem (The birthday paradox, formal)

Suppose q values are picked uniformly at random, with replacement, from a set of size N. Then the probability of no collisions among the q selected values is

$$\text{Prob}(\textit{no collision}) = \prod_{i=1}^{t-1} \left(1 - \frac{i}{N}\right)$$

The birthday paradox

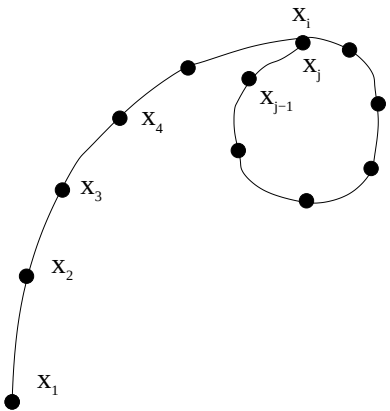
The following approximation will usually be good enough for most purposes in this classes:

Theorem (The birthday paradox, approximation)

Suppose q values are picked uniformly at random, with replacement, from a set of size N. Then the probability of a collision among the q selected values is approximately

$$\text{Prob}(\textit{collision}) \approx \frac{q^2}{N}$$

Pollard's rho algorithm for collision-finding



- Set $x_{i+1} = H(x_i)$
- Search for $x_i = x_j$
- Then $H(x_{i-1}) = H(x_{j-1})$

Naive implementation needs to store all the x_i values to know when it has looped

Floyd's cycle-finding algorithm

To avoid storing all the x_i's:

1. Set $y_0 = x_0$
2. Set $y_{i+1} = H(H(y_i))$
3. Then $y_i = x_{2i}$

Claim. $x_i = y_i$ for some i.

- x_i is eventually periodic.
- Suppose $x_i = x_{i+t}$ for $i > N$
- Let $kt > N$
- Then $y_{kt} = x_{2kt} = x_{kt+kt} = x_{kt}$ since $kt > N$

Hence one can detect collisions with only two elements of storage and 3 times the computational cost:

- Store x_i and y_i
- Compute x_{i+1} and y_{i+1} and check if they are equal
- If they are equal, we have a collision
- If they are not equal, replace x_i and y_i with x_{i+1} and y_{i+1} and repeat.

VW Parallel Collision Search

- Van Oorschot & Wiener (1993)
- Very small space requirements.
- Easy to parallelize:
  - m-fold speedup with m processors.
- Described a 1996 US$10 million machine which can find collisions for 128-bit hash functions in 21 days. Collisions for 160-bit hash functions such as SHA-1 would take about 3500 years.
- The collision-finding algorithm can easily be modified to find “meaningful” collisions.

VW Parallel Collision Search

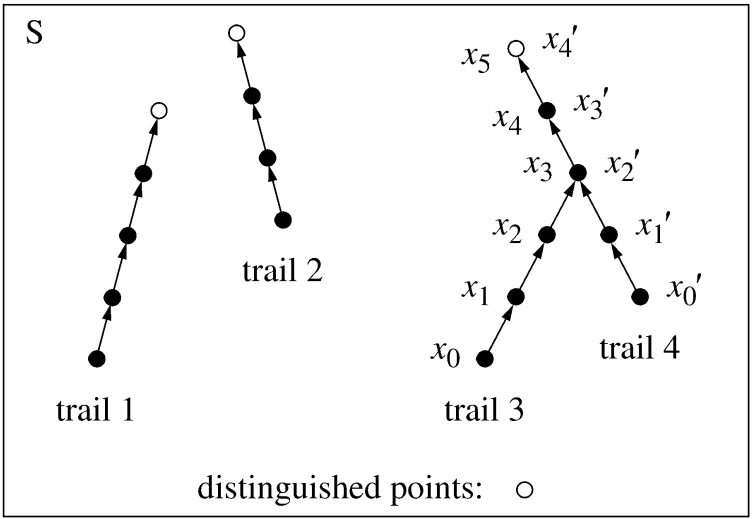
Define a distinguished point to be a bitstring whose first k bits are zeros.

- Compute $x_i = H(x_{i-1})$
- Store x_i if and only if x_i is a distinguished point.
- If x_i is a distinguished point, compare it to all previously stored distinguised points, to see if there is a match.
- If there is a match, "backtrack" to find the collision.

Compared to brute-force search, VW Parallel Collision search:

- Reduces storage by a factor of 2^k
- Increases computational time by an additive 2^k
- Parallelizes trivially to multiple machines or processors

VW Parallel Collision Search



Outline

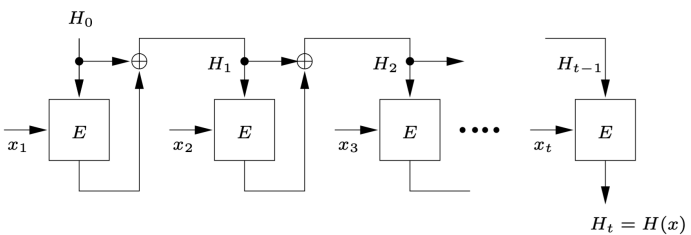
The Davies–Meyer construction

Idea: Build a hash function from a block cipher

Let E_k be an m-bit block cipher with n-bit key k.
Let IV be a fixed m-bit initializing value.

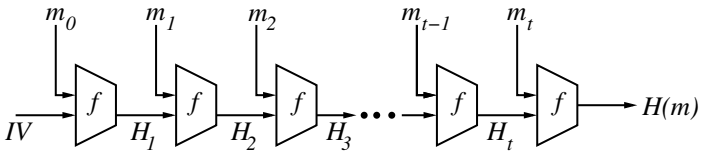
To compute $H(x)$, do:

1. Break up $x\|1$ into n-bit blocks:
   $\overline{x} = x_1, x_2, \ldots, x_t$,
   padding out the last block with 0 bits if necessary.
2. Define $H_0 = IV$.
3. Compute $H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}$
   for $i = 1, 2, \ldots, t$.
4. Define $H(x) = H_t$.



Outline

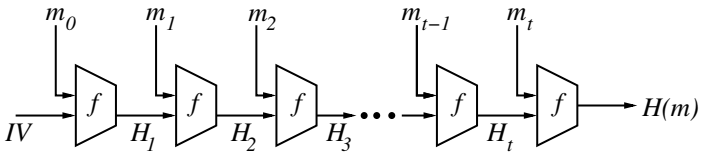
The Merkle–Damgård construction



Components:

- Fixed initialization vector $IV \in \{0,1\}^n$.
- Compression function $f\colon \{0,1\}^{n+r} \longrightarrow \{0,1\}^n$ (efficiently computable).

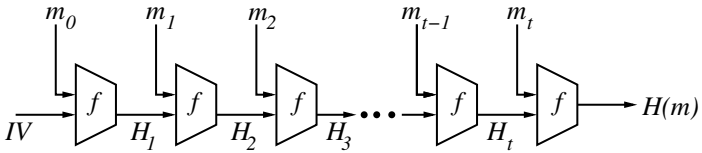
The Merkle–Damgård construction



To compute $H(m)$ where m has bitlength $b < 2^r$ do:

1. Break up m into r-bit blocks: $\overline{m} = m_0, m_1, m_2, \ldots, m_{t-1}$, padding out the last block with 0 bits if necessary.
2. Define m_t, the length-block, to hold the right-justified binary representation of b.
3. Define $H_0 = IV$.
4. Compute $H_{i+1} = f(H_i, m_i)$ for $i = 0, 1, 2, \ldots, t$.
5. H_i's are called chaining variables.
6. Return $H(m) = H_{t+1}$.

Collision Resistance of Merkle–Damgård



Theorem (Merkle)

If the compression function f is collision resistant, then the hash function H is also collision resistant.

Merkle's theorem reduces the problem of designing collision-resistant hash functions to that of designing collision-resistant compression functions.

Proof sketch – see handwritten notes.

Outline

MDx-Family of Hash Functions

- MDx is a family of iterated hash functions using the Merkle–Damgård construction.
- MD4 was proposed by Ron Rivest in 1990.
- MD4 has 128-bit outputs.

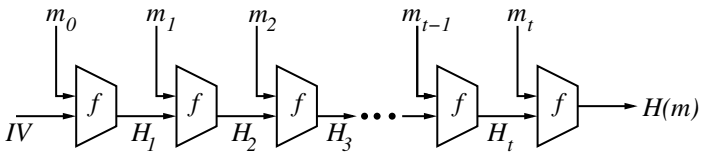
- MD5 is a strengthened version of MD4.
- Designed by Ron Rivest in 1991.
- MD5 has 128-bit outputs.

SHA-1 and SHA-2

- Secure Hash Algorithm (SHA) was designed by NSA and published by NIST in 1993 (FIPS 180).
- 160-bit iterated hash function, based on MD4.
- Slightly modified to SHA-1 (FIPS 180-1) in 1994 in order to fix an (undisclosed) security weakness.

- SHA-2 family designed by NSA and published by NIST in 2001
- Based on MDx family of designs, but more complicated
- Four security levels: SHA-224, SHA-256, SHA-384, SHA-512 with the corresponding number of output bits

General structure of MDx hash functions



- MD4, MD5, SHA-1, and SHA-2 are *iterated hash functions*.
- Input is divided into 512-bit blocks $m_0, m_2, \ldots, m_t$ (1024-bit pieces for SHA-512).
- Input is padded (1 followed by 0's) to a bitlength congruent to 448 mod 512 (896 mod 1024 for SHA-512).
- A final (zero-padded) 64-bit number containing the bitlength of the input in binary is appended (128-bit for SHA-512).
- f is a complicated function $\{0,1\}^n \times \{0,1\}^{512} \to \{0,1\}^n$.
- IV is a public constant.

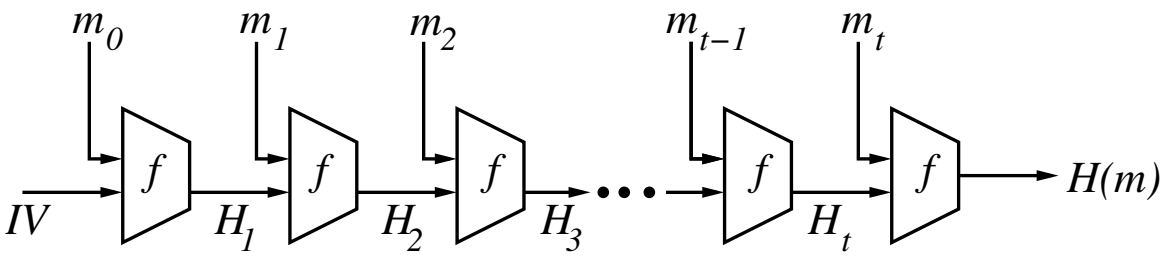
General structure of MDx compression function

Hash	Number of rounds in f
MD4	48
MD5	64
SHA-1	80
SHA-256	64
SHA-512	80

Note: Do not confuse the number of rounds with t, which is simply the number of 512-bit message blocks.

- A single 512-bit input block is divided into 32-bit words (1024-bit block and 64-bit words for SHA-512).
- The sixteen 32-bit words are used in the first 16 rounds of f.
- Subsequent rounds of f use either some combination of the original 16 words (MD4, MD5), or words derived from the original sixteen (SHA-1, SHA-2) via a *message schedule*.
- CAUTION: Endian-ness of words and padding differs between MDx and SHA-x.

High-Level Description of SHA-1



- Iterated hash function (Merkle–Damgård construction).
- $n = 160$, $r = 512$.
- Compression function is $f\colon \{0,1\}^{160+512} \longrightarrow \{0,1\}^{160}$.
- Input: bitstring x of arbitrary bitlength $b \geq 0$.
- Output: 160-bit hash value $H(x)$ of x.

SHA-1 notation

A, B, C, D, E	32-bit quantities.
$+$	addition modulo 2^{32}.
$A \hookleftarrow s$	cyclic left rotation by s bit-positions.
$\neg A$	bitwise complement.
AB	bitwise AND.
$A \vee B$	bitwise inclusive-OR.
$A \oplus B$	bitwise exclusive-OR.
$F(A, B, C)$	$AB \vee (\neg A)C$ (select B or C, using A).
$G(A, B, C)$	$AB \vee AC \vee BC$ (majority rule).
$H(A, B, C)$	$A \oplus B \oplus C$ (bitwise add A, B, and C).

- $\mathrm{IV} = (h_1, h_2, h_3, h_4, h_5)$ where h_1 = 0x67452301, h_2 = 0xefcdab89, h_3 = 0x98badcfe, h_4 = 0x10325476, h_5 = 0xc3d2e1f0
- More constants: k_1 = 0x5a827999, k_2 = 0x6ed9eba1, k_3 = 0x8f1bbcdc, k_4 = 0xca62c1d6 (digits of $\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{10}$)

SHA-1 preprocessing

To compute SHA-1(x):

- Pad x (with 1 followed by 0's) so that its bitlength is 64 less than a multiple of 512.
- Append a 64-bit representation of $b \bmod 2^{64}$.
- The formatted input is $x_0, x_1, \ldots, x_{16m-1}$, where each x_i is a 32-bit word.
- Initialize chaining variables: $(H_1, H_2, H_3, H_4, H_5) \leftarrow (h_1, h_2, h_3, h_4, h_5)$.

SHA-1 compression function

f: $\{0,1\}^{160} \times \{0,1\}^{512} \to \{0,1\}^{160}$ is the *compression* function.

1. Let $(H_1, H_2, H_3, H_4, H_5) \in \{0,1\}^{160}$ be the left input (each H_i is 32 bits).
2. Set $(A, B, C, D, E) \leftarrow (H_1, H_2, H_3, H_4, H_5)$.
3. Let $M \in \{0,1\}^{512}$ be the other input (sixteen 32-bit words).
4. Message schedule:
   For j from 0 to 15: $X_j \leftarrow M_j$
   For j from 16 to 79: $X_j \leftarrow (X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) \hookleftarrow 1$.

Note: The rotate operation "$\hookleftarrow 1$" is not present in SHA. This is the only difference between SHA and SHA-1.

SHA-1 compression function continued

5. (Rounds 1-20) For j from 0 to 19 do:
   $t \leftarrow ((A \hookleftarrow 5) + F(B, C, D) + E + X_j + k_1)$, $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
5. (Rounds 21-40) For j from 20 to 39 do:
   $t \leftarrow ((A \hookleftarrow 5) + H(B, C, D) + E + X_j + k_2)$, $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
5. (Rounds 41-60) For j from 40 to 59 do:
   $t \leftarrow ((A \hookleftarrow 5) + G(B, C, D) + E + X_j + k_3)$, $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
5. (Rounds 61-80) For j from 60 to 79 do:
   $t \leftarrow ((A \hookleftarrow 5) + H(B, C, D) + E + X_j + k_4)$, $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
6. Update H values: $(H_1, H_2, H_3, H_4, H_5) \leftarrow (H_1 + A, H_2 + B, H_3 + C, H_4 + D, H_5 + E)$.
7. Output: $f(x) = (H_1, H_2, H_3, H_4, H_5)$.

MD4 notation

A, B, C, D	32-bit quantities.
$+$	addition modulo 2^{32}.
$A \hookleftarrow s$	cyclic left rotation by s bit-positions.
$\neg A$	bitwise complement.
AB	bitwise AND.
$A \vee B$	bitwise inclusive-OR.
$A \oplus B$	bitwise exclusive-OR.
$F(A, B, C)$	$AB \vee (\neg A)C$ (select B or C, using A).
$G(A, B, C)$	$AB \vee AC \vee BC$ (majority rule).
$H(A, B, C)$	$A \oplus B \oplus C$ (bitwise add A, B, and C).

- $\mathrm{IV} = (h_1, h_2, h_3, h_4)$ where h_1 = 0x67452301, h_2 = 0xefcdab89, h_3 = 0x98badcfe, h_4 = 0x10325476
- More constants: k_1 = 0x5a827999, k_2 = 0x6ed9eba1 (digits of $\sqrt{2}$, $\sqrt{3}$).

MD4 compression function

Compression function: f: $\{0,1\}^{128} \times \{0,1\}^{512} \to \{0,1\}^{128}$

1. Let $(H_1, H_2, H_3, H_4) \in \{0,1\}^{128}$ be the left input (each H_i is 32 bits).
2. Set $(A, B, C, D) \leftarrow (H_1, H_2, H_3, H_4)$.
3. Let $M \in \{0,1\}^{512}$ be the other input (sixteen 32-bit words).
4. Message schedule: $X_j \leftarrow M_{P[j]}$ where

$$P = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,$$
$$0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15,$$
$$0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]$$

5. Shift schedule: In round j, shift by $s[j]$ bits where

$$s = [3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19,$$
$$3, 5, 9, 13, 3, 5, 9, 13, 3, 5, 9, 13, 3, 5, 9, 13$$
$$3, 9, 11, 15, 3, 9, 11, 15, 3, 9, 11, 15, 3, 9, 11, 15]$$

MD4 processing

6. (Rounds 1-16) For j from 0 to 15 do:
   $t \leftarrow (A + F(B, C, D) + X_j) \hookleftarrow s[j]$, $(A, B, C, D) \leftarrow (D, t, B, C)$.
6. (Rounds 17-32) For j from 16 to 31 do:
   $t \leftarrow (A + F(B, C, D) + X_j + k_1) \hookleftarrow s[j]$, $(A, B, C, D) \leftarrow (D, t, B, C)$.
6. (Rounds 33-48) For j from 32 to 47 do:
   $t \leftarrow (A + F(B, C, D) + X_j + k_2) \hookleftarrow s[j]$, $(A, B, C, D) \leftarrow (D, t, B, C)$.
7. Update H values:
   $(H_1, H_2, H_3, H_4) \leftarrow (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$.
8. Output: $f(x) = (H_1, H_2, H_3, H_4)$.

MD5 notation

A, B, C, D	32-bit quantities.
$+$	addition modulo 2^{32}.
$A \leftrightarrow s$	cyclic left rotation by s bit-positions.
$\neg A$	bitwise complement.
AB	bitwise AND.
$A \vee B$	bitwise inclusive-OR.
$A \oplus B$	bitwise exclusive-OR.
$F(A, B, C)$	$AB \vee (\neg A)C$ (select B or C, using A).
$G(A, B, C)$	$A \oplus B \oplus C$ (bitwise add A, B, and C).
$H(A, B, C)$	$B \oplus (A \vee (\neg C))$

- $\mathrm{IV} = (h_1, h_2, h_3, h_4)$ where h_1 = 0x67452301, h_2 = 0xefcdab89, h_3 = 0x98badcfe, h_4 = 0x10325476
- More constants: $k_i = \lfloor 2^{32} \cdot |\sin(i+1)| \rfloor$ (in radians!)

MD5 compression function

Compression function: f: $\{0,1\}^{128} \times \{0,1\}^{512} \to \{0,1\}^{128}$

1. Let $(H_1, H_2, H_3, H_4) \in \{0,1\}^{128}$ be the left input.
2. Set $(A, B, C, D) \leftarrow (H_1, H_2, H_3, H_4)$.
3. Let $X \in \{0,1\}^{512}$ be the other input.
4. Message schedule: See next slide
5. Shift schedule: In round j, shift by $s[j]$ bits where

$$
\begin{aligned}
s = [&7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,\\
&5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,\\
&4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,\\
&6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]
\end{aligned}
$$

MD5 processing

6. (Rounds 1-16) For j from 0 to 15 do:
   $t \leftarrow (A + F(B, C, D) + M_j + k_j) \hookleftarrow s[j]$, $(A, B, C, D) \leftarrow (D, B + t, B, C)$.
6. (Rounds 17-32) For j from 16 to 31 do:
   $t \leftarrow (A + F(D, B, C) + M_{(5j+1) \bmod 16} + k_j) \hookleftarrow s[j]$, $(A, B, C, D) \leftarrow (D, B + t, B, C)$.
6. (Rounds 33-48) For j from 32 to 47 do:
   $t \leftarrow (A + G(B, C, D) + M_{(3j+5) \bmod 16} + k_j) \hookleftarrow s[j]$, $(A, B, C, D) \leftarrow (D, B + t, B, C)$.
6. (Rounds 49-64) For j from 48 to 63 do:
   $t \leftarrow (A + H(B, C, D) + M_{(7j) \bmod 16} + k_j) \hookleftarrow s[j]$, $(A, B, C, D) \leftarrow (D, B + t, B, C)$.
7. Update H values:
   $(H_1, H_2, H_3, H_4) \leftarrow (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$.
8. Output: $f(x) = (H_1, H_2, H_3, H_4)$.

Performance

Speed benchmarks for software implementation on an Intel Core i9 2.9 GHz six-core Coffee Lake (8950HK) using OpenSSL 1.1.1d

	Block length (bits)	Key length (bits)	Digest length (bits)	Speed (Mbytes/sec)
RC4	—	128		644
ChaCha20	—	256		1846
DES	64	56		98
3DES	64	(112)		39
AES (software)	128		128	254
AES (AES-NI)	128		128	1632
MD5	512	—	128	832
SHA-1	512	—	160	932
SHA-256	512	—	256	419
SHA-512	1024	—	512	550

Outline

Generic attacks

These attacks are generic because they work on *any* hash function.

- To find a preimage for an ℓ-bit hash:
  - Try random inputs until the desired hash is found.
  - Requires $O(2^\ell)$ operations on average.
- To find a collision for an ℓ-bit hash:
  - Try random inputs until two matching hashes are found.
  - Requires $O(2^{\ell/2})$ operations on average (birthday paradox!)
- Generic attacks against MD5:
  - 2^{64} operations (collision)
  - 2^{128} operations (preimage)
- Generic attacks against SHA-1:
  - 2^{80} operations (collision)
  - 2^{160} operations (preimage)

Non-generic attacks

- MD4 (RSA Laboratories):
  - Collisions found in 2^{15} steps (Dobbertin, 1996)
  - ...found in 2^{3} steps (Wang et al., 2005)
  - ...found in 2^{0} steps (Sasaki et al., 2009)
  - Preimages found in 2^{102} steps (Leurent, 2008)
- MD5 (RSA Laboratories):
  - Collisions found in MD5 compression function (Dobbertin, 1996)
  - ...found in 2^{39} steps (Wang and Yu, 2004)
  - ...found in 31 seconds on a notebook computer (Kilma, 2006)
  - Preimages (theoretical) in $2^{123.4}$ steps (Sasaki and Aoki, 2009)

- MD5 should not be used if collision resistance is required, but is not horrible as a one-way hash function.
- MD5 remains widely used in legacy software
- MD5 shows up about 850 times in Windows source code.

Non-generic attacks

Non-generic attacks are attacks which exploit a specific function.

- SHA (NIST/NSA, based on MD4):
  - Collisions (theoretical) in 2^{61} steps (Chabaud and Joux, 1998)
  - ...found in 2^{51} steps (Joux et al., 2004)
  - ...found in 2^{40} steps (Wang et al., 2004)
- SHA-1 (NIST/NSA, based on SHA):
  - Collisions (theoretical) in 2^{63} steps (Wang et al., 2005)
  - ...found in $2^{63.1}$ steps (CWI/Google, 2017)
  - No preimage or 2nd preimage attacks are known on SHA-1.

Wang's Collision-Finding Attack on SHA-1

- Fix any n-bit string I.
- Wang's attack finds two (different) 1-block messages $x = x_1$ and $y = y_1$ such that $F(I, x_1) = F(I, y_1)$.
- The attack gives limited, but not complete, control over x_1 and y_1.
- The attack takes about 2^{63} steps.
- By selecting $I = IV$ (where IV is the fixed initialization vector specified in SHA-1), Wang's attack can be used to find two one-block messages x and y such that SHA-1(x)=SHA-1(y). The attacker does not have much control over x and y, so these messages are essentially meaningless.

Wang's Collision-Finding Attack on MD5

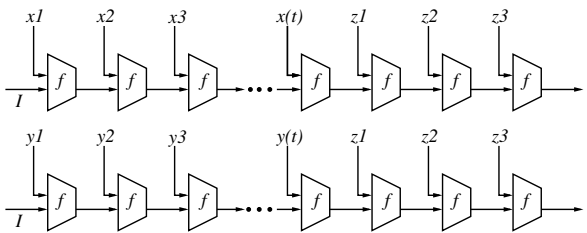
- Fix any n-bit string I.
- Wang's attack finds two (different) two-block messages $x = (x_1, x_2)$ and $y = (y_1, y_2)$ such that $F(I, x) = F(I, y)$.
- The attack gives limited, but not complete, control over x and y.
- The attack takes about 2^{39} steps.
- By selecting $I = IV$ (where IV is the fixed initialization vector specified in MD5), Wang's attack can be used to find two two-block messages x and y such that MD5(x)=MD5(y). The attacker does not have much control over x and y, so these messages are essentially meaningless.

How to Exploit a Single Hash Collision

- Let H be a hash function (such as MD5 or SHA-1).
- Suppose that we can find two different messages x and y so that $H(x) = H(y)$.
- Suppose that the collision-finding method we use does not allow us to control the structure of x and y, so that these messages are essentially meaningless.
- Suppose also that the collision-finding methods takes considerable (but feasible) time.
- Question: How can an attacker, who has expended considerable resources to find two meaningless messages x and y that collide, make repeated use of this collision in a practical setting?

Extending a Collision

- Notation: For a t-block message $x = (x_1, x_2, \ldots, x_t)$ and n-bit string I, define $F(I, x) = H_t$, where $H_0 = I$ and $H_i = f(H_{i-1}, x_i)$ for $i = 1, 2, \ldots, t$.
- Observation: Suppose x and y are two messages of the same block-length such that $F(I, x) = F(I, y)$. Then $F(I, x\|z) = F(I, y\|z)$ for any message z. Furthermore, if $I = IV$, then $H(x, z) = H(y, z)$ for any message z.



Exploiting Wang's Collisions

- Suppose that MD5 is being used in a hash-then-sign signature scheme. [The example also works for SHA-1]
- Let M_1 and M_2 be two documents in postscript format. Suppose that M_1 is "harmless" for Alice, and M_2 is "harmful" for Alice.
- Daum and Lucks (2005) showed how Wang's attack on MD5 can be used to find two new postscript files $\widehat{M}_1$ and $\widehat{M}_2$ such that:
  1. When $\widehat{M}_1$ is viewed (using a standard postscript viewer) or printed (on a postscript printer), it looks the same as M_1. Similarly for $\widehat{M}_2$.
  2. MD5($\widehat{M}_1$) = MD5($\widehat{M}_2$).

Exploiting Wang's Collisions

- The attacker Eve sends the postscript file $\widehat{M}_1$ to Alice. Alice views or prints the file, and then signs it and returns to Eve. Since MD5($\widehat{M}_1$) = MD5($\widehat{M}_2$), Eve also has Alice's signature on $\widehat{M}_2$.

Details of Daum and Lucks' Attack

The attacker Eve does the following:

1. Select a postscript "preamble" so that the string

   $$p = \text{``preamble (''}$$

   has bitlength a multiple of the block-length r. (If necessary, comments can be added as padding.)
2. Compute $I = F(IV, p)$.
3. Use Wang's attack to find two distinct two-block messages x and y such that $F(I, x) = F(I, y)$.
4. Select any two postscript files M_1 and M_2. Let T_1, T_2 be the postscript commands for displaying M_1 and M_2.

Details of Daum and Lucks' Attack

5. Set $\widehat{M}_1$ = preamble $(x)(x)\text{eq}\{T_1\}\{T_2\}$
   and $\widehat{M}_2$ = preamble $(y)(x)\text{eq}\{T_1\}\{T_2\}$

Why this works:

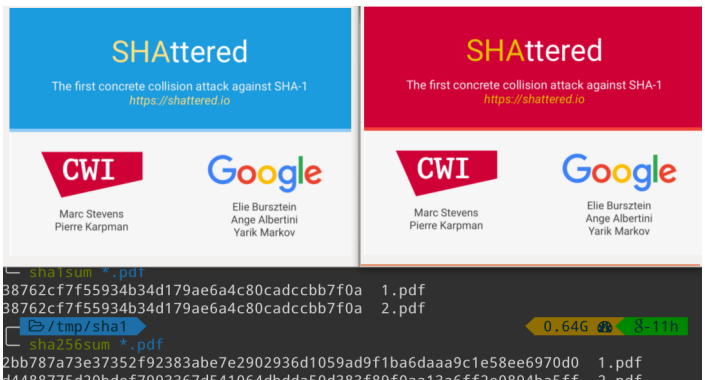
1. MD5($\widehat{M}_1$)= MD5($\widehat{M}_2$)!!
2. Postscript interprets the commands $(S_1)(S_2)\text{eq}\{T_1\}\{T_2\}$ as follows: If $S_1 = S_2$, then execute the commands T_1; else execute the commands T_2.
   Hence, if $\widehat{M}_1$ is viewed, then M_1 is displayed.
   If $\widehat{M}_2$ is viewed, then M_2 is displayed.

Notes on Daum and Lucks' Attack

- Of course, careful examination of the postscript files $\widehat{M}_1$ and $\widehat{M}_2$ would reveal Eve's deception—but no one reads raw postscript code before viewing or printing a postscript document.
- The collision x, y can be reused for any two postscript files M_1 and M_2.
- Gebhardt, Illies and Schindler (2005): Similar attacks on documents in PDF, TIFF and Word formats.
- Moral of this story: Don't be quick to dismiss attacks on cryptographic schemes, even if the attacks appear at first to have limited practical value.

Practical collisions in SHA-1

- 2017: “SHAttered”: First full collision on SHA-1
- Required 9,223,372,036,854,775,808 SHA-1 compressions to find collision (9 quintillion), 110 GPU-years
- `https://shattered.io/`
- 2020: “SHA-1 is a shambles”: Improved to cost $11k for a collision, $45k for chosen-prefix collision



Outline

SHA-3

- The SHA-2 design is similar to SHA-1, and thus there are concerns that the SHA-1 weaknesses will extend to SHA-2.
- SHA-3: NIST hash function competition.
  - 64 candidates submitted by Oct 31 2008 deadline.
  - 51 were accepted for the first round.
  - (July 2009) 14 were selected for the second round.
  - Encourage the public to study the hash functions.
  - Third quarter 2010: Select a list of finalists.
    (BLAKE, Grøstl, JH, Keccak, Skein)
  - Encourage the public to study the finalists.
  - Second quarter 2012: Announce a winner: Keccak (Bertoni, Daemen, Peeters, Van Assche)
  - Standard published by NIST in 2015

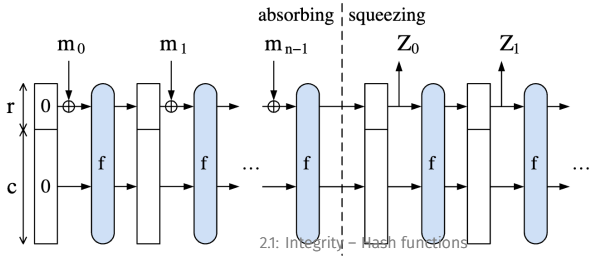
Sponge function

- Components:
  - state $S \in \{0,1\}^b$
  - function $f\colon \{0,1\}^b \to \{0,1\}^b$, often a permutation
  - a padding function
- The state S is divided into two parts: $R \in \{0,1\}^r$ and $C \in \{0,1\}^c$ where $b = r + c$; r is called the rate and c is the capacity

Sponge function

To hash a message m:

1. Initialize state $S = R \| C$ to zero
2. Pad message to break into r-bit blocks
3. [Absorb stage] For each r-bit block B:
   - 3.1 Replace $R \leftarrow R \oplus B$
   - 3.2 Replace $S \leftarrow f(S)$
4. [Squeeze stage] While more output bits are needed:
   - 4.1 Output R
   - 4.2 Replace $S \leftarrow f(S)$



SHA-3

- SHA-3 is a sponge construction using the Keccak permutation on a 1600-bit state
- Different levels of security by having a larger hidden capacity
- Modes for providing fixed and arbitrary length output

	Output size	Rate r	Capacity c	Security strength in bits	
				Collision	Pre/2nd-Pre
SHA3-224	224	1152	448	112	224
SHA3-256	256	1088	512	128	256
SHA3-384	384	832	768	192	384
SHA3-512	512	576	1024	256	512
SHAKE128	d	1344	256	$\min\{\frac{d}{2}, 128\}$	$\min\{d, 128\}$
SHAKE256	d	1088	512	$\min\{\frac{d}{2}, 256\}$	$\min\{d, 256\}$

NIST's Policy on Hash Functions

- December 2022
- `https://csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions`

- Should stop using SHA-1 for digital signatures and other applications that require collision resistance.
- May still use SHA-1 for HMAC, KDFs, and random number generators.
- May use SHA-2 for all applications that employ secure hash algorithms.
- SHA-3 may also be used, but this is not required.
- Withdraw SHA-1 from approved usage by end of 2030.