

Hash functions

Martin Stanek

Department of Computer Science
Comenius University
`stanek@dcs.fmph.uniba.sk`

Cryptology 1 (2022/23)

Content

Hash function properties

preimage / second preimage / collision resistance

Birthday attack

Constructions

hard problems

block cipher based

Merkle-Damgård construction

Examples of real-world hash functions

SHA-256

SHA-3 (Keccak)

Introduction

- ▶ hash function computes a fixed-length fingerprint/digest/hash from a message/document of (almost) arbitrary length
- ▶ $h : X \rightarrow Y$ function (deterministic computation)
- ▶ efficient (fast) & no key used
- ▶ usually $X = \{0, 1\}^*$, $X = \{0, 1\}^{\leq 2^{64}}$, $X = \{0, 1\}^{\leq 2^{128}}$, ...
 $Y = \{0, 1\}^{160}$ for SHA-1, $\{0, 1\}^{256}$ for SHA-256 and SHA3-256, ...
- ▶ various uses of h.f.:
 - ▶ digital signature schemes (digest of the message is signed)
 - ▶ padding in public-key encryption schemes
 - ▶ verifying integrity of data
 - ▶ instantiation of random oracles and pseudorandom functions
 - ▶ MAC constructions
 - ▶ password storing methods etc.

Basic requirements of hash functions (informally)

- ▶ preimage resistance (one-way)
 - ▶ It is infeasible to compute $x \in X$ given $y \in h(X)$ such that $h(x) = y$.
- ▶ second preimage resistance
 - ▶ It is infeasible to compute $x' \in X$ given $x \in X$ such that $x \neq x'$ & $h(x) = h(x')$.
- ▶ collision resistance
 - ▶ It is infeasible to compute $x, x' \in X$ such that $x \neq x'$ & $h(x) = h(x')$.
- ▶ remarks:
 - ▶ $|X| \gg |Y|$, otherwise the h.f. is useless \Rightarrow large number of collisions
 - ▶ Y is finite, h is deterministic \Rightarrow in theory, e.g. collisions can be found in $O(1)$ time (“hardcoded”)
 - ▶ formalizing the requirements is not straightforward (introduction of *hash function families*, multiple “flavors” of preimage and second preimage resistance) – however, above intuition satisfies our needs
 - ▶ Pre, Sec, Coll, (aPre, ePre, aSec, eSec), MAC, Prf, Pro, TCR, CTFP, ...

Basic requirements of hash functions (informally)

- ▶ preimage resistance (one-way)
 - ▶ It is infeasible to compute $x \in X$ given $y \in h(X)$ such that $h(x) = y$.
- ▶ second preimage resistance
 - ▶ It is infeasible to compute $x' \in X$ given $x \in X$ such that $x \neq x'$ & $h(x) = h(x')$.
- ▶ collision resistance
 - ▶ It is infeasible to compute $x, x' \in X$ such that $x \neq x'$ & $h(x) = h(x')$.
- ▶ remarks:
 - ▶ $|X| \gg |Y|$, otherwise the h.f. is useless \Rightarrow large number of collisions
 - ▶ Y is finite, h is deterministic \Rightarrow in theory, e.g. collisions can be found in $O(1)$ time (“hardcoded”)
 - ▶ formalizing the requirements is not straightforward (introduction of *hash function families*, multiple “flavors” of preimage and second preimage resistance) – however, above intuition satisfies our needs
 - ▶ Pre, Sec, Coll, (aPre, ePre, aSec, eSec), MAC, Prf, Pro, TCR, CTFP, ...

Properties of h.f. – discussion

- ▶ collision resistance \Rightarrow second preimage resistance
 - ▶ if you can find a second preimage, then you have a collision
- ▶ collision resistance \nRightarrow preimage resistance
 - ▶ identity: $X = Y, \forall x \in X : h(x) = x$ (Coll, \neg Pre)
 - ▶ let g with range $\{0, 1\}^n$ be collision and preimage resistant; then

$$h(x) = \begin{cases} 0 \parallel x & \text{if } |x| = n \\ 1 \parallel g(x) & \text{otherwise} \end{cases}$$

is collision resistant but not preimage resistant

- ▶ second preimage resistance \nRightarrow preimage resistance
 - ▶ identity again (Sec, \neg Pre)
- ▶ however, in a “normal” situation ...

Collision by inverting h.f.

- ▶ assumption: h can be inverted efficiently
- ▶ algorithm:
 1. $x \xleftarrow{\$} X$
 2. invert $h(x) \mapsto x'$
 3. if $x' \neq x$... collision found
- ▶ let us estimate the probability of success
- ▶ notation: $[x] = \{x' \in X; h(x') = h(x)\}$ equivalence class
- ▶ C – set of all equivalence classes

$$\begin{aligned}\Pr_{\text{succ}} &= \frac{1}{|X|} \sum_{x \in X} \frac{|[x]| - 1}{|[x]|} = \frac{1}{|X|} \sum_{c \in C} \sum_{x \in c} \frac{|c| - 1}{|c|} = \frac{1}{|X|} \sum_{c \in C} (|c| - 1) \\ &= \frac{1}{|X|} \left(\underbrace{\sum_{c \in C} |c|}_{|X|} - \underbrace{\sum_{c \in C} 1}_{\leq |Y|} \right) \geq 1 - \frac{|Y|}{|X|} \quad \dots \geq \underbrace{1 - \left(\frac{|Y|}{|X|}\right)^k}_{\text{after } k \text{ repetitions}}\end{aligned}$$

Generic attack for finding preimage/2nd preimage

- ▶ generic attack, finding a preimage for given $y \in h(X)$:
- ▶ algorithm:
 1. choose $x \in X$ (randomly or systematically)
 2. if $h(x) = y$ then the preimage is found, otherwise repeat
- ▶ expected complexity $O(2^n)$ for $Y = \{0, 1\}^n$
- ▶ similar generic attack for finding a second preimage

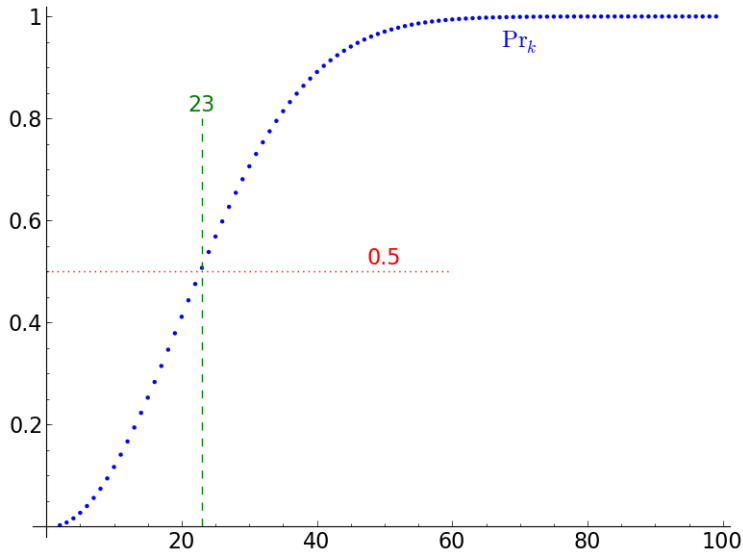
Birthday attack – introduction

- ▶ generic attack for finding collision(s)
- ▶ example: What is the probability that at least two people in a room share the same birthday?

$$\Pr_2 = 1 - \frac{365 \cdot 364}{365^2} \approx 0.0027; \quad \Pr_3 = 1 - \frac{365 \cdot 364 \cdot 363}{365^3} \approx 0.0082$$

- ▶ k people: $\Pr_k = 1 - 365^k/365^k$
- ▶ at least 23 people needed for probability $\geq 1/2$
- ▶ “hash function” maps people to dates; $|Y| = 365$

Birthday attack – introduction (2)



Birthday attack on h.f.

1. choose (distinct) $x_1, \dots, x_k \xleftarrow{\$} X$
 2. compute $h(x_1), \dots, h(x_k)$
 3. find collisions, for example by sorting $(h(x_i), x_i)$ and searching for collisions in adjacent elements, or by storing $(h(x_i), x_i)$ in a hash table using the hash value as a key
- ▶ linear time and memory complexity $O(k)$
 - ▶ we treat n as a constant (for $Y = \{0, 1\}^n$); also assuming constant time to evaluate h
 - ▶ time: using Radixsort for sorting in $O(k)$ or using a hash table with $k \times O(1)$ operations
 - ▶ memory complexity can be improved (see later)

What is the probability of success?

Birthday attack – analysis (1)

- ▶ trivial observations – the probability of success increases:
 - ▶ for increasing k
 - ▶ for unbalanced distribution of images
- ▶ assume the worst situation: h distributes the hash values uniformly, i.e.

$$\Pr[h(x) = y] = 1/|Y| \quad \forall y \in Y$$

- ▶ let y_1, \dots, y_k be random, independent and uniform elements from Y
- ▶ notation: $|Y| = N$
- ▶ probability that all y_i 's are distinct:

$$\Pr_{\text{dist}} = \frac{N(N-1) \dots (N-k+1)}{N^k} = \left(1 - \frac{1}{N}\right) \left(1 - \frac{2}{N}\right) \dots \left(1 - \frac{k-1}{N}\right)$$

Birthday attack – analysis (2)

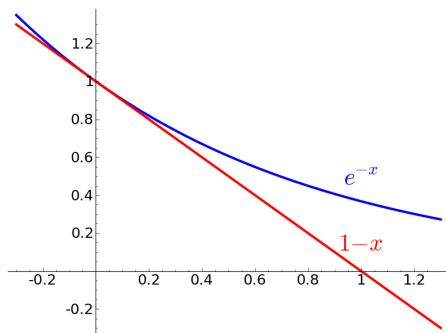
- ▶ probability of at least one collision: $\Pr_{\text{col}} = 1 - \Pr_{\text{dist}}$
- ▶ estimate \Pr_{col} :

$$\Pr_{\text{col}} = 1 - \prod_{i=1}^{k-1} \left(1 - \frac{i}{N}\right) \geq 1 - e^{-\frac{1}{N} - \frac{2}{N} - \dots - \frac{k-1}{N}} = 1 - e^{-\frac{k(k-1)}{2N}}$$

we use inequality $1 - x \leq e^{-x}$

it follows from Taylor series:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots$$



Birthday attack – analysis (3)

- find k such that $\Pr_{\text{col}} \geq \varepsilon$, for some constant $\varepsilon \in (0, 1)$

$$\Pr_{\text{col}} \geq 1 - e^{-k(k-1)/(2N)} \geq \varepsilon$$

$$1 - \varepsilon \geq e^{-k(k-1)/(2N)}$$

$$2N \ln(1 - \varepsilon) \geq -k^2 + k$$

$$k^2 - k + 2N \ln(1 - \varepsilon) \geq 0$$

$$k \geq \frac{1}{2} + \sqrt{\frac{1}{4} + 2N \ln \frac{1}{1 - \varepsilon}}$$

$$k \geq \sqrt{N} \cdot \sqrt{2 \ln \frac{1}{1 - \varepsilon}}$$

Birthday attack – results

- ▶ the complexity of b.a. for “reasonable” ε , e.g. $1/2$, $2/3$, is $O(N^{1/2})$
- ▶ for $Y = \{0, 1\}^n$ we get $\approx 2^{n/2}$ (e.g. for SHA-1 $\approx 2^{80}$)
- ▶ expected k for given success probability:

$$50\% \quad k \approx 1.177 \cdot 2^{n/2}$$

$$90\% \quad k \approx 2.146 \cdot 2^{n/2}$$

$$99\% \quad k \approx 3.035 \cdot 2^{n/2}$$

Implications of birthday attack

- ▶ generic attack, i.e. any h.f. can be attacked this way
 - ▶ recall: generic attack for symmetric encryption is brute-force, $O(2^k)$ for key length k
- ▶ the length of hash value (digest) should be twice the length of symmetric key used for encryption
- ▶ standardized parameters of AES and SHA-2 family:

AES key length		128	192	256
SHA-2 output length	$224^{(*)}$	256	384	512

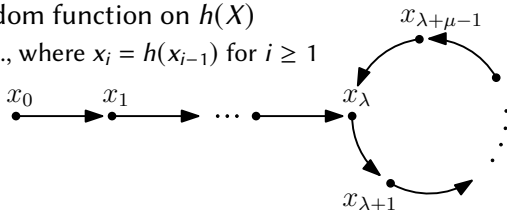
(*) this corresponds to the effective key length of 3DES (112 bits)

“Meaningful” collisions

- ▶ prepare documents m, m' with t places that can be changed without changing the meaning of the document
 - ▶ one space vs. two spaces, synonyms etc.
- ▶ 2^t variants of each document
- ▶ hash and find a collision between these two sets
- ▶ the same asymptotic time and memory complexity of b.a.

Improving memory complexity of birthday attack (1)

- ▶ assumption: h as a random function on $h(X)$
 - ▶ sequence: x_0, x_1, x_2, \dots , where $x_i = h(x_{i-1})$ for $i \geq 1$



- ▶ expected (as $N \rightarrow \infty$): $\rho = \lambda + \mu = \sqrt{\pi N/2}$
- ▶ finding collision in constant memory:
 1. $x_0 \xleftarrow{\$} X$ (using $X \setminus Y$ guarantees the existence of a collision, $\lambda \geq 1$)
 2. compute (x_i, x_{2i}) for $i \geq 1$: $x_i = h(x_{i-1})$, $x_{2i} = h(h(x_{2(i-1)}))$
 3. if $x_i = x_{2i}$ then $h^i(x_0) = h^{2i}(x_0)$, we found a point on the cycle, $\lambda \leq i$, and the collision can be computed as follows:
 - 3.1 compute (x_j, x_{i+j}) for $j = 0, 1, \dots, i$ starting with (x_0, x_i)
 - 3.2 check for situation when $x_j \neq x_{i+j}$ and $x_{j+1} = x_{i+j+1}$
 - 3.3 collision $h(x_j) = h(x_{i+j})$; remark: $\mu \mid (2i - i) \Rightarrow x_\lambda = x_{i+\lambda}$

Improving memory complexity of birthday attack (2)

- ▶ only a constant number of values (e.g. x_0 , and the recent pair of values (x_i, x_{2i}) or (x_j, x_{i+j})) should be stored
- ▶ complexity:
 - ▶ cycle is detected (point is found) if $i \geq \lambda$ and $\mu \mid i$
 - ▶ the difference $2i - i$ increases by 1 in each iteration, i.e. the cycle is detected with $\lambda + \mu$ iterations maximum
 - ▶ complexity $O(\lambda + \mu) = O(\sqrt{N})$
- ▶ this method does not change the asymptotic time complexity of b.a.
- ▶ no control over the colliding messages/inputs

Collision resistance in practice

- ▶ collision resistance is not easy
- ▶ MD5
 - ▶ designed by Ron Rivest (1991)
 - ▶ collision published in 2005
- ▶ SHA-1
 - ▶ designed by NSA, published as a standard in 1995
 - ▶ deprecated in major web browsers in 2017
 - ▶ first collision published in 2017; two pdf files, see <https://shattered.io/>
 - ▶ attack complexity: $2^{63.1}$ SHA-1 compressions
- ▶ hash functions in web servers certificates (for signatures):

	01/2015	01/2016	01/2017	01/2018	01/2019
SHA-1	66.7%	13.2%	1.5%	0.0%	0.0%
SHA-256	33.3%	86.8%	98.4%	99.8%	99.8%

source: SSL Pulse, <https://www.ssllabs.com/ssl-pulse/>

Hash functions based on hard problems

- ▶ provable properties (assuming the hardness of underlying problem)
- ▶ slow, impractical \Rightarrow not used in practice
- ▶ example based on discrete logarithm problem:
 - ▶ (G, \cdot) – group of prime order p ; let g be a generator of (G, \cdot)
 - ▶ $f \in G$, such that $\alpha = \log_g f$ is unknown
 - ▶ $h : \mathbb{Z}_p \times \mathbb{Z}_p \rightarrow G$ is defined as follows: $h(a, b) = g^a \cdot f^b$
 - ▶ h is collision resistant, otherwise we can find α :

$$h(a, b) = h(a', b') \quad \text{where } (a, b) \neq (a', b')$$

$$g^a \cdot f^b = g^{a'} \cdot f^{b'}$$

$$g^{a+\alpha b} = g^{a'+\alpha b'} \quad \Rightarrow \quad \alpha = \frac{a - a'}{b' - b} \bmod p$$

Hash functions based on block ciphers

- ▶ $m = m_1, m_2, \dots, m_k$ input divided into blocks
- ▶ h_0 – initialization vector; h_i – intermediate hash value ($1 \leq i \leq k$)
- ▶ iteration – sequential processing of input blocks
- ▶ examples:
 - ▶ Matyas, Meyer, Oseas: $h_i = E_{g(h_{i-1})}(m_i) \oplus m_i$
 - ▶ Davies, Meyer: $h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1}$
 - ▶ Miyaguchi, Preneel: $h_i = E_{g(h_{i-1})}(m_i) \oplus h_{i-1} \oplus m_i$
- ▶ $H(m) = h_k$ (the hash value is the output of the last iteration)
- ▶ problem: standard block ciphers have small block length
 - ▶ specific block ciphers (SHACAL for SHA-1, W cipher for Whirlpool etc.)
 - ▶ double block length constructions (MDC-4, Hirose, Tandem-DM etc.)

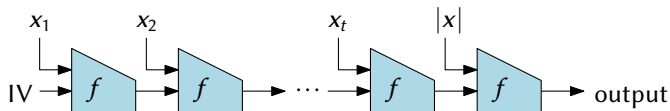
Dedicated constructions

- ▶ no proofs of security based on some “hard underlying problem”
- ▶ fast (usually one of the design goals)
- ▶ iterated construction (informally):
 - ▶ message padding and “slicing”
 - ▶ start with IV and sequentially process the slices
 - ▶ result is the output of the final iteration (sometimes after some additional processing)
- ▶ most common approaches
 - ▶ Merkle-Damgård: SHA-1, SHA-2 family
 - ▶ sponge: SHA-3 (Keccak)

Merkle-Damgård construction (1)

- ▶ collision resistance of compression function implies collision resistance of hash function
- ▶ fixed input length compression function $f : \{0, 1\}^{n+r} \rightarrow \{0, 1\}^n$
- ▶ hash function $H : \{0, 1\}^{\leq l} \rightarrow \{0, 1\}^n$
- ▶ input $x = x_1, x_2, \dots, x_t$ (block length r)
 - ▶ last block padded by 10 ... 0 (if needed)
 - ▶ additional block $x_{t+1} = |x|$; in binary, thus $l < 2^r$
- ▶ other variants of padding used in practice or proposed in the literature
- ▶ using the length in padding \sim MD strengthening
 - ▶ ensures suffix-free property of the padding:
for any $x \neq x'$, $\text{pad}(x)$ is not a suffix of $\text{pad}(x')$
 - ▶ suffix-free \sim necessary and sufficient condition for collision-preserving padding

Merkle-Damgård construction (2)



computation:

1. $h_0 = 0^n$ (initialization vector)
2. $h_i = f(h_{i-1} || x_i)$, for $i = 1, \dots, t + 1$
3. $H(x) = h_{t+1}$

Collision resistance of MD construction

let $x \neq x'$ be a collision in H : $H(x) = H(x')$, i.e. $h_{t+1} = h'_{t'+1}$

a. if $t \neq t'$ then $x_{t+1} \neq x'_{t'+1}$ and $f(h_t, x_{t+1}) = f(h'_{t'}, x'_{t'+1}) \dots$ collision in f

b. $t = t'$: $x = x_1, \dots, x_{t+1}$, $x' = x'_1, \dots, x'_{t+1}$

$f(h_t, x_{t+1}) = f(h'_t, x'_{t+1}) \dots$ either collision in f or

▶ $h_t = h'_t$ & $x_{t+1} = x'_{t+1}$

$f(h_{t-1}, x_t) = f(h'_{t-1}, x'_t) \dots$ either collision in f or

▶ $h_{t-1} = h'_{t-1}$ & $x_t = x'_t$

...

▶ either we get a collision in f or $x = x'$

Collision resistance of MD construction

let $x \neq x'$ be a collision in H : $H(x) = H(x')$, i.e. $h_{t+1} = h'_{t'+1}$

a. if $t \neq t'$ then $x_{t+1} \neq x'_{t'+1}$ and $f(h_t, x_{t+1}) = f(h'_{t'}, x'_{t'+1}) \dots$ collision in f

b. $t = t'$: $x = x_1, \dots, x_{t+1}$, $x' = x'_1, \dots, x'_{t+1}$

$f(h_t, x_{t+1}) = f(h'_t, x'_{t+1}) \dots$ either collision in f or

▶ $h_t = h'_t$ & $x_{t+1} = x'_{t+1}$

$f(h_{t-1}, x_t) = f(h'_{t-1}, x'_t) \dots$ either collision in f or

▶ $h_{t-1} = h'_{t-1}$ & $x_t = x'_t$

...

▶ either we get a collision in f or $x = x'$

Collision resistance of MD construction

let $x \neq x'$ be a collision in H : $H(x) = H(x')$, i.e. $h_{t+1} = h'_{t'+1}$

a. if $t \neq t'$ then $x_{t+1} \neq x'_{t'+1}$ and $f(h_t, x_{t+1}) = f(h'_{t'}, x'_{t'+1}) \dots$ collision in f

b. $t = t'$: $x = x_1, \dots, x_{t+1}$, $x' = x'_1, \dots, x'_{t+1}$

$f(h_t, x_{t+1}) = f(h'_t, x'_{t+1}) \dots$ either collision in f or

▶ $h_t = h'_t$ & $x_{t+1} = x'_{t+1}$

$f(h_{t-1}, x_t) = f(h'_{t-1}, x'_t) \dots$ either collision in f or

▶ $h_{t-1} = h'_{t-1}$ & $x_t = x'_t$

...

▶ either we get a collision in f or $x = x'$

Collision resistance of MD construction

let $x \neq x'$ be a collision in H : $H(x) = H(x')$, i.e. $h_{t+1} = h'_{t'+1}$

a. if $t \neq t'$ then $x_{t+1} \neq x'_{t'+1}$ and $f(h_t, x_{t+1}) = f(h'_{t'}, x'_{t'+1})$... collision in f

b. $t = t'$: $x = x_1, \dots, x_{t+1}$, $x' = x'_1, \dots, x'_{t+1}$

$f(h_t, x_{t+1}) = f(h'_t, x'_{t+1})$... either collision in f or

▶ $h_t = h'_t$ & $x_{t+1} = x'_{t+1}$

$f(h_{t-1}, x_t) = f(h'_{t-1}, x'_t)$... either collision in f or

▶ $h_{t-1} = h'_{t-1}$ & $x_t = x'_t$

...

▶ either we get a collision in f or $x = x'$

Collision resistance of MD construction

let $x \neq x'$ be a collision in H : $H(x) = H(x')$, i.e. $h_{t+1} = h'_{t'+1}$

- a. if $t \neq t'$ then $x_{t+1} \neq x'_{t'+1}$ and $f(h_t, x_{t+1}) = f(h'_{t'}, x'_{t'+1}) \dots$ collision in f
- b. $t = t'$: $x = x_1, \dots, x_{t+1}$, $x' = x'_1, \dots, x'_{t+1}$
 $f(h_t, x_{t+1}) = f(h'_t, x'_{t+1}) \dots$ either collision in f or
 - ▶ $h_t = h'_t$ & $x_{t+1} = x'_{t+1}$
 $f(h_{t-1}, x_t) = f(h'_{t-1}, x'_t) \dots$ either collision in f or
 - ▶ $h_{t-1} = h'_{t-1}$ & $x_t = x'_t$
 - ...
 - ▶ either we get a collision in f or $x = x'$

Parameters of real-world hash function

family	function	length [bits]		
		max. input	output	block
SHA-2	MD-5	$2^{64} - 1$	128	512
	SHA-1	$2^{64} - 1$	160	512
	Whirlpool	$2^{256} - 1$	512	512
	SHA-256	$2^{64} - 1$	256	512
	SHA-384	$2^{128} - 1$	384	1024
	SHA-512	$2^{128} - 1$	512	1024
SHA-3	SHA3-256	∞	256	1088
	SHA3-384	∞	384	832
	SHA3-512	∞	512	576

SHA-2

- ▶ SHA-2 family of hash function
(SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256)
- ▶ standard: FIPS PUB 180-4
- ▶ similar design of SHA-256 (32-bit words, block size 512 bits) and SHA-512 (64-bit words, block size 1024 bits)
- ▶ other variants are truncated versions with different initialization vectors
- ▶ Merkle-Damgård construction

Example: SHA-256

- ▶ input message M ; $l = |M|$ ($0 \leq l < 2^{64}$ bits)
- ▶ padding and parsing:
 - ▶ padding: $M \underbrace{100\dots0}_k \underbrace{(l)_2}_{64 \text{ bits}}$, where k is the smallest value such that the overall length is a multiple of 512
 - ▶ parsing into 512-bit blocks: $M^{(1)}, M^{(2)}, \dots, M^{(N)}$
 - ▶ each block consists of 16 32-bit words: $M^{(i)} = M_0^{(i)}, M_1^{(i)}, \dots, M_{15}^{(i)}$
- ▶ initialization vector (8 32-bit words): $H_0^{(0)}, H_1^{(0)}, \dots, H_7^{(0)}$
- ▶ intermediate hash values: $H_0^{(i)}, H_1^{(i)}, \dots, H_7^{(i)}$
- ▶ SHA-256 digest: $H_0^{(N)}, H_1^{(N)}, \dots, H_7^{(N)}$

SHA-256 compression function

compression function (for $i = 1, \dots, N$):

1. expanding a message block ($\mapsto W_0, \dots, W_{63}$)

$$W_i = \begin{cases} M_t^{(i)} & \text{for } 0 \leq t \leq 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} & \text{for } 16 \leq t \leq 63 \end{cases}$$

2. $(a, b, c, d, e, f, g, h) \leftarrow (H_0^{(i-1)}, H_1^{(i-1)}, \dots, H_7^{(i-1)})$

3. for $t = 0, \dots, 63$:

- 3.1 $T_1 = h + \Sigma_1(e) + \text{Ch}(e, f, g) + K_t + W_t$

- 3.2 $T_2 = \Sigma_0(a) + \text{Maj}(a, b, c)$

- 3.3 $(a, b, c, d, e, f, g, h) \leftarrow (T_1 + T_2, a, b, c, d + T_1, e, f, g)$

4. $(H_0^{(i)}, H_1^{(i)}, \dots, H_7^{(i)}) \leftarrow (a + H_0^{(i-1)}, b + H_1^{(i-1)}, \dots, h + H_7^{(i-1)})$

SHACAL-2 block cipher in Davies-Meyer mode

Functions used in SHA-256

- ▶ functions operate on 32-bit words
- ▶ addition is computed mod 2^{32}
- ▶ $\text{Ch}(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$
- ▶ $\text{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
- ▶ $\Sigma_0(x) = \text{ROTR}^2(x) \oplus \text{ROTR}^{13}(x) \oplus \text{ROTR}^{22}(x)$
- ▶ $\Sigma_1(x) = \text{ROTR}^6(x) \oplus \text{ROTR}^{11}(x) \oplus \text{ROTR}^{25}(x)$
- ▶ $\sigma_0(x) = \text{ROTR}^7(x) \oplus \text{ROTR}^{18}(x) \oplus \text{SHR}^3(x)$
- ▶ $\sigma_1(x) = \text{ROTR}^{17}(x) \oplus \text{ROTR}^{19}(x) \oplus \text{SHR}^{10}(x)$
- ▶ ROTR – circular shift rotation to the right
- ▶ SHR – shift to the right

Some performance numbers

	MB/s
MD5	687
SHA-1	738
SHA-256	323
SHA-512	417
SHA3-256	287
SHA2-512	154

block size: 8192 bytes, 1 thread

platform: i7-2600 @ 3.40 GHz (4 cores/8 threads)

implementation: openssl 1.1.1c

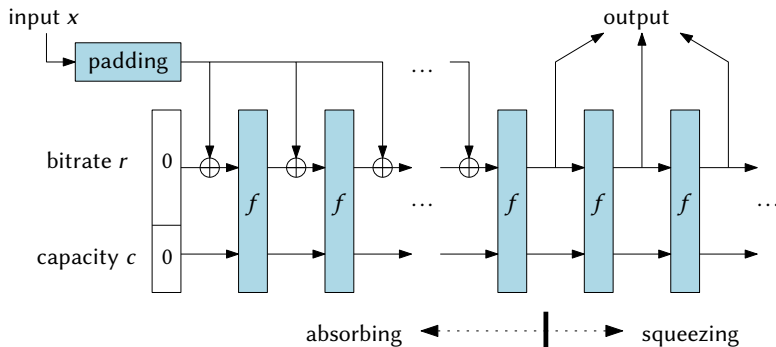
Remark: Intel SHA Extensions – instructions for improving performance of SHA-1 and SHA-256 hash functions (not used in above table); AMD Ryzen and some Intel processors.

SHA-3 overview

- ▶ Keccak – winner of SHA-3 competition (2012)
- ▶ standard: NIST FIPS 202 (2015)
 - ▶ 4 hash functions with fixed-length output: SHA3-224, SHA3-256, SHA3-384, SHA3-512
 - ▶ 2 functions with variable-length output (XOF – extendable-output functions): SHAKE128, SHAKE256
- ▶ different approach than SHA-1 or SHA-2 hash functions
- ▶ Keccak is not an MD-construction
- ▶ sponge construction
- ▶ other functions/variants/constructions proposed:
 - ▶ SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash (NIST SP 800-185, 2016)

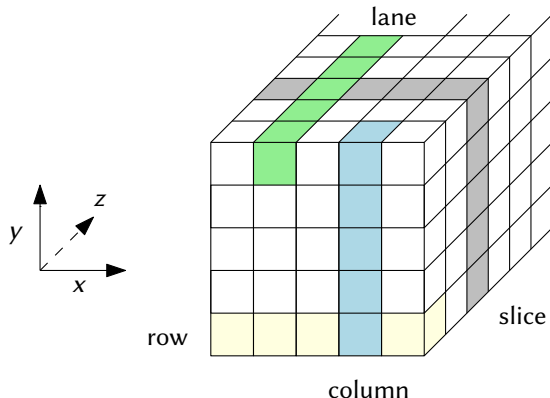
SHA-3 structure

- ▶ sponge construction – absorbing & squeezing
 - ▶ arbitrary output length
 - ▶ f – permutation on $\{0, 1\}^{r+c}$
 - ▶ r – bitrate (e.g. 1088 for SHA3-256)
 - ▶ c – capacity (e.g. 512 for SHA3-256)
 - ▶ padding for SHA3-256: $x || 01 || 10^*1$



SHA-3 inside permutation f (1)

- ▶ state: $5 \times 5 \times 2^l$ bits ($2^l = 64$ for SHA3-256)



- ▶ $12 + 2l$ rounds (24 rounds for SHA3-256)
- ▶ round function: $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$, (θ is applied first)

SHA-3 inside permutation f (2)

- ▶ θ (theta) – xor each bit of a column with parities of two neighboring columns
- ▶ ρ (rho) – rotate each lane by a constant value
- ▶ π (pi) – permute the positions of the lanes
- ▶ χ (chi) – flip bit if neighbors to the right are 0, 1
 - ▶ χ operates on rows (independently, in parallel)
- ▶ ι (iota) – xor a round specific constant to lane[0,0]
 - ▶ destroying symmetry

