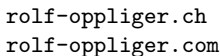


Chapter 6 – Cryptographic Hash Functions

March 1, 2022

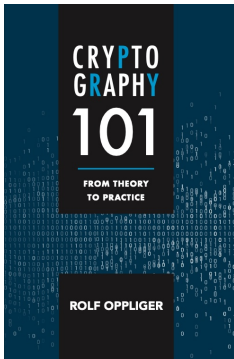
Terms of Use

- This work is published with a CC BY-ND 4.0 license (CC BY ND)
 - CC = Creative Commons (CC)
 - BY = Attribution (BY)
 - ND = No Derivatives (ND)



- Swiss National Cyber Security Centre
NCSC (scientific employee)
- eSECURITY Technologies Rolf Oppliger
(founder and owner)
- University of Zurich (adjunct professor)
- Artech House (author and series editor for
information security and privacy)

Reference Book



© Artech House, 2021
ISBN 978-1-63081-846-3

<https://books.esecurity.ch/crypto101.html>

Challenge Me



6. Cryptographic Hash Functions

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

6.1 Introduction

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

6.1 Introduction

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

6.1 Introduction

- Preimage resistance (one-wayness) and collision resistance are inherently different properties
- On the one hand, a preimage resistant function need not be (strong or weak) collision-resistant
 - If g is an n -bit preimage resistant hash function, then the function $h(x) = g(x \mid_n)$ is still preimage resistant but not collision-resistant
 - All $x \parallel y$ (with $|x| \geq n$ and y arbitrary) hash to the same value and yield collisions

6.1 Introduction

- On the other hand, a (strong or weak) collision-resistant hash function need not be preimage resistant (e.g., Maurer's counterexample)

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

- A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

6.1 Introduction

- The notion of a collision can be generalized to **multicollisions**
- More specifically, an **r -collision** is an r -tuple (x_1, \dots, x_r) with $h(x_1) = \dots = h(x_r)$
- For $r = 2$, a 2-collision is a “normal” collision
- Finding multicollisions is not substantially more difficult than finding “normal” collisions

6.1 Introduction

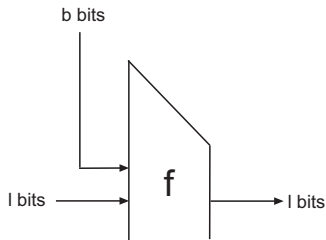
- In practice, Σ_{in} and Σ_{out} are often set to $\{0, 1\}$
- A respective hash function is a mapping from $\{0, 1\}^*$ to $\{0, 1\}^n$
- A practically relevant question is how large n should be
- There is a trade-off here, i.e., n should be as short as possible, but as long as needed
- A lower bound for n is obtained by the birthday attack that exploits the birthday paradox (e.g., $n \geq 256$ to achieve a 128-bit security level)

6.1 Introduction

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

6.2 Merkle-Damgård Construction

- In the late 1980s, Ralph C. Merkle and Ivan B. Damgård independently proposed a construction that can be used to turn a collision-resistant compression function $f : \Sigma^{b+l} \rightarrow \Sigma^l$ (with $b, l \in \mathbb{N}$) into an iterated hash function h



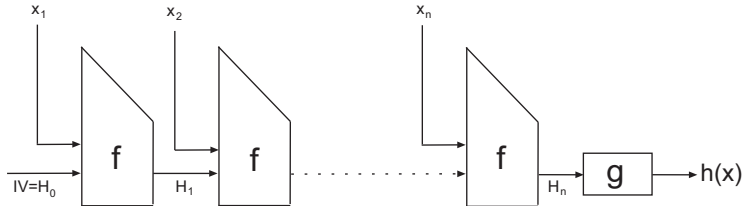
6.2 Merkle-Damgård Construction

- $$H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1} \quad \text{for } i = 1, \dots, n$$

6. Cryptographic Hash Functions

6.2 Merkle-Damgård Construction

- In a typical setting, l is 160 or 256 bits and b is 512 bits
- An iterated hash function looks as follows:



6. Cryptographic Hash Functions

6.2 Merkle-Damgård Construction

- Such a function h can be defined as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(H_{i-1}, x_i) \quad \text{for } i = 1, \dots, n \\ h(x) &= g(H_n) \end{aligned}$$

- The message x must be padded to a multiple of b bits
- The padding method of choice is to append (at the end of the message) a 1, a variable number of 0s, and the binary encoding of the message length

6.2 Merkle-Damgård Construction

- ## Theorem (Merkle-Damgård)

If the compression function f is collision-resistant, then the iterated hash function h that is built according to the Merkle-Damgård construction is also collision-resistant

- There are only a few cryptographic hash functions that don't employ the Merkle-Damgård construction (e.g., KECCAK)

6.3 Historical Perspective

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

6.3 Historical Perspective

- In 1993, the U.S. NIST proposed the **Secure Hash Algorithm (SHA)**, which is similar to MD5, but more strengthened and a little bit slower
- Probably after discovering a never-published weakness in the original SHA proposal, the U.S. NIST released **SHA-1**
- In 1995, SHA-1 was specified in FIPS PUB 180 (later in RFC 4634) and has been revised multiple times since then
- The latest revision is FIPS PUB 180-4 (August 2015)
- It also introduces the **SHA-2 family**

6. Cryptographic Hash Functions

6.3 Historical Perspective

Table 6.1
Secure Hash Algorithms as Specified in FIPS 180-4

Algorithm	Message Size	Block Size	Word Size	Hash Value Size
SHA-1	$< 2^{64}$ bits	512 bits	32 bits	160 bits
SHA-224	$< 2^{64}$ bits	512 bits	32 bits	224 bits
SHA-256	$< 2^{64}$ bits	512 bits	32 bits	256 bits
SHA-384	$< 2^{128}$ bits	1,024 bits	64 bits	384 bits
SHA-512	$< 2^{128}$ bits	1,024 bits	64 bits	512 bits
SHA-512/224	$< 2^{128}$ bits	1,024 bits	64 bits	224 bits
SHA-512/256	$< 2^{128}$ bits	1,024 bits	64 bits	256 bits

6.3 Historical Perspective

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ▶ ↺ 🔍 ↻

6. Cryptographic Hash Functions

6.3 Historical Perspective

- In 2005, Wang et al. also presented collisions for SHA-1
- The original attack required 2^{69} (instead of 2^{80}) hash operations to find a collision, but it can be improved to 2^{63}
- The attack was widely discussed in the media and led to a better adoption of SHA-2
- Also, a NIST competition for **SHA-3** was initiated
- In 2012, **KECCAK** was announced as the winner of the competition
- **RIPEMD** and **RIPEMD-160** are European versions of MD5 and SHA-1 (not further addressed)

6.4 Exemplary Hash Functions — MD4

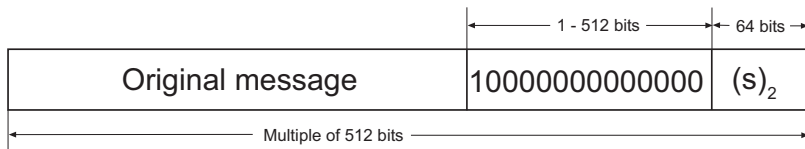
- **MD4** follows the Merkle-Damgård construction and uses a Davies-Meyer compression function with $b = 512$ and $l = 128$
- The output length is 128 bits
- The function was designed to be efficiently executed on 32-bit processors with a little-endian architecture
- This means that a 4-byte word $a_1a_2a_3a_4$ is stored as $a_4a_3a_2a_1$, representing the integer $a_42^{24} + a_32^{16} + a_22^8 + a_1$

6.4 Exemplary Hash Functions — MD4

- $$w[n-1] = m_{s-32} m_{s-31} \dots m_{s-1}$$

6.4 Exemplary Hash Functions — MD4

- More specifically, w is constructed in two steps:
 - First, m is padded (with a 1 and variable number of 0s) so that the bitlength is congruent to 448 modulo 512 (i.e., 64 bits short of being a multiple of 512 bits)
 - Second, a 64-bit binary representation of s is appended (to form the last two words of w)



6.4 Exemplary Hash Functions — MD4

- $W \overset{\curvearrowright}{\leftarrow} c$ refers to the c -bit left rotation (circular left shift) of word w ($0 \leq c \leq 31$)

X	Y	Z	f	g	h
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	1	1

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — MD4

Round 2:

1. $A = (A + g(B, C, D) + X[0] + c_1) \xleftrightarrow{\quad} 3$
2. $D = (D + g(A, B, C) + X[4] + c_1) \xleftrightarrow{\quad} 5$
3. $C = (C + g(D, A, B) + X[8] + c_1) \xleftrightarrow{\quad} 9$
4. $B = (B + g(C, D, A) + X[12] + c_1) \xleftrightarrow{\quad} 13$
5. $A = (A + g(B, C, D) + X[1] + c_1) \xleftrightarrow{\quad} 3$
6. $D = (D + g(A, B, C) + X[5] + c_1) \xleftrightarrow{\quad} 5$
7. $C = (C + g(D, A, B) + X[9] + c_1) \xleftrightarrow{\quad} 9$
8. $B = (B + g(C, D, A) + X[13] + c_1) \xleftrightarrow{\quad} 13$
9. $A = (A + g(B, C, D) + X[2] + c_1) \xleftrightarrow{\quad} 3$
10. $D = (D + g(A, B, C) + X[6] + c_1) \xleftrightarrow{\quad} 5$
11. $C = (C + g(D, A, B) + X[10] + c_1) \xleftrightarrow{\quad} 9$
12. $B = (B + g(C, D, A) + X[14] + c_1) \xleftrightarrow{\quad} 13$
13. $A = (A + g(B, C, D) + X[3] + c_1) \xleftrightarrow{\quad} 3$
14. $D = (D + g(A, B, C) + X[7] + c_1) \xleftrightarrow{\quad} 5$
15. $C = (C + g(D, A, B) + X[11] + c_1) \xleftrightarrow{\quad} 9$
16. $B = (B + g(C, D, A) + X[15] + c_1) \xleftrightarrow{\quad} 13$

Round 3:

1. $A = (A + h(B, C, D) + X[0] + c_2) \xleftrightarrow{\quad} 3$
2. $D = (D + h(A, B, C) + X[8] + c_2) \xleftrightarrow{\quad} 9$
3. $C = (C + h(D, A, B) + X[4] + c_2) \xleftrightarrow{\quad} 11$
4. $B = (B + h(C, D, A) + X[12] + c_2) \xleftrightarrow{\quad} 15$
5. $A = (A + h(B, C, D) + X[2] + c_2) \xleftrightarrow{\quad} 3$
6. $D = (D + h(A, B, C) + X[10] + c_2) \xleftrightarrow{\quad} 9$
7. $C = (C + h(D, A, B) + X[6] + c_2) \xleftrightarrow{\quad} 11$
8. $B = (B + h(C, D, A) + X[14] + c_2) \xleftrightarrow{\quad} 15$
9. $A = (A + h(B, C, D) + X[1] + c_2) \xleftrightarrow{\quad} 3$
10. $D = (D + h(A, B, C) + X[9] + c_2) \xleftrightarrow{\quad} 9$
11. $C = (C + h(D, A, B) + X[5] + c_2) \xleftrightarrow{\quad} 11$
12. $B = (B + h(C, D, A) + X[13] + c_2) \xleftrightarrow{\quad} 15$
13. $A = (A + h(B, C, D) + X[3] + c_2) \xleftrightarrow{\quad} 3$
14. $D = (D + h(A, B, C) + X[11] + c_2) \xleftrightarrow{\quad} 9$
15. $C = (C + h(D, A, B) + X[7] + c_2) \xleftrightarrow{\quad} 11$
16. $B = (B + h(C, D, A) + X[15] + c_2) \xleftrightarrow{\quad} 15$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — MD5

- **MD5** is a strengthened version of MD4
- It is conceptually and structurally similar to MD4
- The main difference is that MD5 invokes 4 rounds (instead of only 3)
- This is advantageous from a security viewpoint, but it is disadvantageous from a performance viewpoint (i.e., performance decreases one third)
- MD5 uses a slightly modified function f , an additional function i , and a word table T with 64 entries that is constructed from the sine function (instead of c_1 and c_2)

6.4 Exemplary Hash Functions — MD5

(*m*)

$A = 0x67452301$

$$B = 0_{\text{x}}\text{EFCDAB89}$$

$C = 0x98BADCFE$

$D = 0x10325476$

```
for  $i = 0$  to  $n/16 - 1$  do
```

```
for  $j = 0$  to 15 do  $X[j] = w[i \cdot 16 + j]$ 
```

$$A' = A$$
$$B' = B$$
$$C' = C$$
$$D' = D$$

Round 1

Round 2

Round 3

Round 4

$$A = A + A'$$
$$B = B + B'$$
$$C = C + C'$$
$$D = D + D'$$
$$(h(m) = A \parallel B \parallel C \parallel D)$$

1. $A = (A + f(B, C, D) + X[0] + T[1])$ $\leftarrow 7$
2. $D = (D + f(A, B, C) + X[1] + T[2])$ $\leftarrow 12$
3. $C = (C + f(D, A, B) + X[2] + T[3])$ $\leftarrow 17$
4. $B = (B + f(C, D, A) + X[3] + T[4])$ $\leftarrow 22$
5. $A = (A + f(B, C, D) + X[4] + T[5])$ $\leftarrow 7$
6. $D = (D + f(A, B, C) + X[5] + T[6])$ $\leftarrow 12$
7. $C = (C + f(D, A, B) + X[6] + T[7])$ $\leftarrow 17$
8. $B = (B + f(C, D, A) + X[7] + T[8])$ $\leftarrow 22$
9. $A = (A + f(B, C, D) + X[8] + T[9])$ $\leftarrow 7$
10. $D = (D + f(A, B, C) + X[9] + T[10])$ $\leftarrow 12$
11. $C = (C + f(D, A, B) + X[10] + T[11])$ $\leftarrow 17$
12. $B = (B + f(C, D, A) + X[11] + T[12])$ $\leftarrow 22$
13. $A = (A + f(B, C, D) + X[12] + T[13])$ $\leftarrow 7$
14. $D = (D + f(A, B, C) + X[13] + T[14])$ $\leftarrow 12$
15. $C = (C + f(D, A, B) + X[14] + T[15])$ $\leftarrow 17$
16. $B = (B + f(C, D, A) + X[15] + T[16])$ $\leftarrow 22$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — MD5

Round 2:

1. $A = (A + g(B, C, D) + X[1] + T[17]) \xleftrightarrow{5}$
2. $D = (D + g(A, B, C) + X[6] + T[18]) \xleftrightarrow{9}$
3. $C = (C + g(D, A, B) + X[11] + T[19]) \xleftrightarrow{14}$
4. $B = (B + g(C, D, A) + X[0] + T[20]) \xleftrightarrow{20}$
5. $A = (A + g(B, C, D) + X[5] + T[21]) \xleftrightarrow{5}$
6. $D = (D + g(A, B, C) + X[10] + T[22]) \xleftrightarrow{9}$
7. $C = (C + g(D, A, B) + X[15] + T[23]) \xleftrightarrow{14}$
8. $B = (B + g(C, D, A) + X[4] + T[24]) \xleftrightarrow{20}$
9. $A = (A + g(B, C, D) + X[9] + T[25]) \xleftrightarrow{5}$
10. $D = (D + g(A, B, C) + X[14] + T[26]) \xleftrightarrow{9}$
11. $C = (C + g(D, A, B) + X[3] + T[27]) \xleftrightarrow{14}$
12. $B = (B + g(C, D, A) + X[8] + T[28]) \xleftrightarrow{20}$
13. $A = (A + g(B, C, D) + X[13] + T[29]) \xleftrightarrow{5}$
14. $D = (D + g(A, B, C) + X[2] + T[30]) \xleftrightarrow{9}$
15. $C = (C + g(D, A, B) + X[7] + T[31]) \xleftrightarrow{14}$
16. $B = (B + g(C, D, A) + X[12] + T[32]) \xleftrightarrow{20}$

Round 3:

1. $A = (A + h(B, C, D) + X[5] + T[33]) \xleftrightarrow{4}$
2. $D = (D + h(A, B, C) + X[8] + T[34]) \xleftrightarrow{11}$
3. $C = (C + h(D, A, B) + X[11] + T[35]) \xleftrightarrow{16}$
4. $B = (B + h(C, D, A) + X[14] + T[36]) \xleftrightarrow{23}$
5. $A = (A + h(B, C, D) + X[1] + T[37]) \xleftrightarrow{4}$
6. $D = (D + h(A, B, C) + X[4] + T[38]) \xleftrightarrow{11}$
7. $C = (C + h(D, A, B) + X[7] + T[39]) \xleftrightarrow{16}$
8. $B = (B + h(C, D, A) + X[10] + T[40]) \xleftrightarrow{23}$
9. $A = (A + h(B, C, D) + X[13] + T[41]) \xleftrightarrow{4}$
10. $D = (D + h(A, B, C) + X[0] + T[42]) \xleftrightarrow{11}$
11. $C = (C + h(D, A, B) + X[3] + T[43]) \xleftrightarrow{16}$
12. $B = (B + h(C, D, A) + X[6] + T[44]) \xleftrightarrow{23}$
13. $A = (A + h(B, C, D) + X[9] + T[45]) \xleftrightarrow{4}$
14. $D = (D + h(A, B, C) + X[12] + T[46]) \xleftrightarrow{11}$
15. $C = (C + h(D, A, B) + X[15] + T[47]) \xleftrightarrow{16}$
16. $B = (B + h(C, D, A) + X[2] + T[48]) \xleftrightarrow{23}$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — MD5

Round 4:

1. $A = (A + i(B, C, D) + X[0] + T[49]) \xleftrightarrow{6}$
2. $D = (D + i(A, B, C) + X[7] + T[50]) \xleftrightarrow{10}$
3. $C = (C + i(D, A, B) + X[14] + T[51]) \xleftrightarrow{15}$
4. $B = (B + i(C, D, A) + X[5] + T[52]) \xleftrightarrow{21}$
5. $A = (A + i(B, C, D) + X[12] + T[53]) \xleftrightarrow{6}$
6. $D = (D + i(A, B, C) + X[3] + T[54]) \xleftrightarrow{10}$
7. $C = (C + i(D, A, B) + X[10] + T[55]) \xleftrightarrow{15}$
8. $B = (B + i(C, D, A) + X[1] + T[56]) \xleftrightarrow{21}$
9. $A = (A + i(B, C, D) + X[8] + T[57]) \xleftrightarrow{6}$
10. $D = (D + i(A, B, C) + X[15] + T[58]) \xleftrightarrow{10}$
11. $C = (C + i(D, A, B) + X[6] + T[59]) \xleftrightarrow{15}$
12. $B = (B + i(C, D, A) + X[13] + T[60]) \xleftrightarrow{21}$
13. $A = (A + i(B, C, D) + X[4] + T[61]) \xleftrightarrow{6}$
14. $D = (D + i(A, B, C) + X[11] + T[62]) \xleftrightarrow{10}$
15. $C = (C + i(D, A, B) + X[2] + T[63]) \xleftrightarrow{15}$
16. $B = (B + i(C, D, A) + X[9] + T[64]) \xleftrightarrow{21}$

- MD5 is susceptible to collision attacks
- While a “normal” attack requires 2^{64} hash computations, the collision attack of Wang et al. requires 2^{39} and the best-known attack 2^{32}
- This value is so small that MD5 must not be used anymore

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-1

- **SHA-1** was specified by the U.S. NIST in FIPS PUB 180 (currently FIPS PUB 180-4)
- Again, it is conceptually and structurally similar to MD5
- Major differences
 - SHA-1 is optimized for computer systems with a big-endian architecture (instead of a little-endian architecture)
 - SHA-1 employs 5 registers *A*, *B*, *C*, *D*, and *E* (instead of 4)
 - SHA-1 yields 160-bit hash values (instead of 128-bit hash values)

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-1

- Instead of f , g , h , and i , SHA-1 uses a sequence of 80 logical functions f_0, f_1, \dots, f_{79} :

$$f_t(X, Y, Z) = \begin{cases} Ch(X, Y, Z) = (X \wedge Y) \oplus ((\neg X) \wedge Z) & 0 \leq t \leq 19 \\ Parity(X, Y, Z) = X \oplus Y \oplus Z & 20 \leq t \leq 39 \\ Maj(X, Y, Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z) & 40 \leq t \leq 59 \\ Parity(X, Y, Z) = X \oplus Y \oplus Z & 60 \leq t \leq 79 \end{cases}$$

- Note that the *Parity* function occurs twice ($20 \leq t \leq 39$ and $60 \leq t \leq 79$), and that *Ch* and *Maj* are similar to f and g (\vee is replaced with \oplus , but this doesn't change the result)

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-1

- Instead of c_1 and c_2 (MD4) or the 64 words of table T (MD5), SHA-1 uses 4 constant 32-bit words that are used to build a sequence of 80 words K_0, K_1, \dots, K_{79} :

$$K_t = \begin{cases} \lfloor 2^{30} \sqrt{2} \rfloor = \text{0x5A827999} & 0 \leq t \leq 19 \\ \lfloor 2^{30} \sqrt{3} \rfloor = \text{0x6ED9EBA1} & 20 \leq t \leq 39 \\ \lfloor 2^{30} \sqrt{5} \rfloor = \text{0x8F1BBCDC} & 40 \leq t \leq 59 \\ \lfloor 2^{30} \sqrt{10} \rfloor = \text{0xCA62C1D6} & 60 \leq t \leq 79 \end{cases}$$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-1

- While w is an array of 32-bit words in MD4 and MD5, SHA-1 uses an array b of 16-word blocks instead
- Hence, $b[i]$ ($i = 0, 1, \dots, n - 1$) refers to a 16-word block that is $16 \cdot 32 = 512$ bits long
- SHA-1 uses each 16-word block b to recursively derive an 80-word message schedule W :

$$W_t = \begin{cases} b_t & 0 \leq t \leq 15 \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \ll 1 & 16 \leq t \leq 79 \end{cases}$$

- The 16 words of b become the first 16 words of W , and the remaining $80 - 16 = 64$ words of W are generated according to the formula

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-1

Overview:

(m)

Construct $b = b[0] \parallel b[1] \parallel \dots \parallel b[n-1]$

$A = 0x67452301$

$B = 0xEFCDAB89$

$C = 0x98BADCFE$

$D = 0x10325476$

$E = 0xC3D2E1F0$

for $i = 0$ to $n - 1$ do

 Derive message schedule W from $b[i]$

$A' = A$

$B' = B$

$C' = C$

$D' = D$

$E' = E$

 |

|

for $t = 0$ to 79 do

$T = (A \stackrel{\curvearrowright}{\leftarrow} 5) + f_t(B, C, D) + E + K_t + W_t$

$E = D$

$D = C$

$C = B \stackrel{\curvearrowright}{\leftarrow} 30$

$B = A$

$A = T$

$A = A + A'$

$B = B + B'$

$C = C + C'$

$D = D + D'$

$E = E + E'$

$(h(m) = A \parallel B \parallel C \parallel D \parallel E)$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-1

- SHA-1 was first broken in 2005 (2^{69} instead of 2^{80} hash computations)
- The attack was later improved (2^{63} hash computations)
- In 2011, the U.S. NIST deprecated SHA-1, and disallowed its use for digital signatures by the end of 2013
- Two recent attacks have brought SHA-1 to the end of its life cycle
 - SHAttered (2017)
 - SHA-1 is a Shambles (2019)

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-2 family

- The functions of the **SHA-2 family** are listed in Table 6.1
- The functions employ the *Ch* and *Maj* functions from SHA-1 (applied to 32-bit or 64-bit words)
- In the case of SHA-224 and SHA-256, these functions are complemented by 4 32-bit functions:

$$\Sigma_0^{\{256\}}(X) = (X \curvearrowright 2) \oplus (X \curvearrowright 13) \oplus (X \curvearrowright 22)$$

$$\Sigma_1^{\{256\}}(X) = (X \curvearrowright 6) \oplus (X \curvearrowright 11) \oplus (X \curvearrowright 25)$$

$$\sigma_0^{\{256\}}(X) = (X \curvearrowright 7) \oplus (X \curvearrowright 18) \oplus (X \curvearrowleft 3)$$

$$\sigma_1^{\{256\}}(X) = (X \curvearrowright 17) \oplus (X \curvearrowright 19) \oplus (X \curvearrowleft 10)$$

- Note that \curvearrowleft refers to the *c*-bit right shift operator

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-2 family

- All other hash functions from the SHA-2 family use similar 64-bit functions:

$$\Sigma_0^{\{512\}}(X) = (X \curvearrowright 28) \oplus (X \curvearrowright 34) \oplus (X \curvearrowright 39)$$

$$\Sigma_1^{\{512\}}(X) = (X \curvearrowright 14) \oplus (X \curvearrowright 18) \oplus (X \curvearrowright 41)$$

$$\sigma_0^{\{512\}}(X) = (X \curvearrowright 1) \oplus (X \curvearrowright 8) \oplus (X \curvearrowleft 7)$$

$$\sigma_1^{\{512\}}(X) = (X \curvearrowright 19) \oplus (X \curvearrowright 61) \oplus (X \curvearrowleft 6)$$

- All Σ -functions are used in the round functions, whereas all σ -functions are used to derive the message schedule W

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-2 family

- While SHA-1 uses four 32-bit words to represent the constants K_0, K_1, \dots, K_{79} , SHA-224 and SHA-256 use a sequence of 64 distinct 32-bit words that serve as constants

$$K_0^{\{256\}}, K_1^{\{256\}}, \dots, K_{63}^{\{256\}}$$

- The 64 words are generated by taking the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers (not addressed here)

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-2 family

- Similarly, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 use a sequence of 80 distinct 64-bit words that serve as constants

$$K_0^{\{512\}}, K_1^{\{512\}}, \dots, K_{79}^{\{512\}}$$

- The 80 words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers (so the first 32 bits of the first 64 values are the same as with SHA-224 and SHA-256)

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-2 family

- While SHA-224 and SHA-256 require messages to be padded to a multiple of 512 bits, all other SHA-2 hash functions require messages to be padded to a multiple of 1024 bits
- In this case, the length of the original message is encoded in the final two 64-bit words (instead of two 32-bit words)
- All functions from the SHA-2 family operate on 8 32- or 64-bit registers A , B , C , D , E , F , G , and H
- The registers are initialized in a particular way (not addressed here)

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — SHA-2 family

- As of this writing, the cryptographic hash functions from the SHA-2 family are considered to be secure
- They are used in many applications, such as Bitcoin (double SHA-2) and many other cryptocurrencies
- There is no need to replace them in the short term
- If one is worried about quantum computers, then SHA-384 and SHA-512 can be used

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- KECCAK is the algorithm selected by the U.S. NIST as the winner of the public SHA-3 competition in 2012
- FIPS PUB 202 complements FIPS PUB 180-4
- It specifies 4 cryptographic hash functions and 2 extendable-output functions (XOFs)
 - SHA3-224, SHA3-256, SHA3-384, and SHA3-512
 - SHAKE128 and SHAKE256 (where SHAKE stands for “Secure Hash Algorithm with Keccak”)
- KECCAK/SHA-3 relies on the **sponge construction** (instead of the Merkle-Damgård construction)

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- In December 2016, NIST released SP 800-18540 that specifies complementary functions derived from KECCAK/SHA-3
 - **Customizable SHAKE (cSHAKE)** is a SHAKE XOF that can be customized with a particular bit string to provide domain separation (conceptually similar to a “salt”)
 - **KMAC** is a keyed MAC construction that is based on KECCAK
 - **TupleHash** is a SHA-3-derived function that can be used to hash a tuple of input strings (that are uniquely serialized)
 - **ParallelHash** takes advantage of the parallelism available in some modern processors

6.4 Exemplary Hash Functions — KECCAK/SHA-3

-
- The diagram illustrates the hierarchical structure of a 3D data cube. At the top is a 4x4x4 cube labeled "state". Below it, three 4x4 grids are shown, labeled "plane", "slice", and "sheet", each with a 3D coordinate system (x, y, z). Below these, three 4x1 grids are shown, labeled "row", "column", and "lane", each with a 3D coordinate system (x, y, z). At the bottom, a single cube is labeled "bit", with a 3D coordinate system (x, y, z) below it.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- In the case of SHA-3, $b = 1600$, $0 \leq x, y < 5$, and $0 \leq z < w$ (where $w = 2^l = 64$ for $l = 6$)
- Consequently, the state is either a 1600-bit string S or a $(5 \times 5 \times 64)$ -array \mathbf{A} of 1600 bits
- For all $0 \leq x, y < 5$ and $0 \leq z < w$, the relationship between S and \mathbf{A} is as follows:

$$\mathbf{A}[x, y, z] = S[w(5y + x) + z]$$

- $\mathbf{A}[0, 0, 0]$ translates to $S[0]$, whereas $\mathbf{A}[4, 4, 63]$ translates to $S[64((5 \cdot 4) + 4) + 63] = S[64 \cdot 24 + 63] = S[1599]$

6. Cryptographic Hash Functions

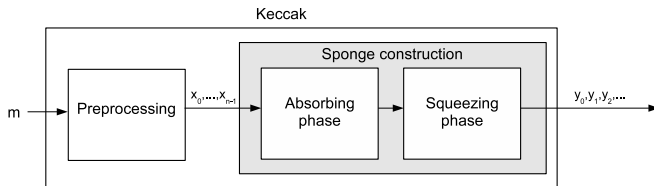
6.4 Exemplary Hash Functions — KECCAK/SHA-3

$$\begin{aligned}
 S &= \mathbf{A} = plane[0] \parallel plane[1] \parallel \dots \parallel plane[4] \\
 &= lane[0, 0] \parallel lane[1, 0] \parallel \dots \parallel lane[4, 0] \parallel \\
 &\quad lane[0, 1] \parallel lane[1, 1] \parallel \dots \parallel lane[4, 1] \parallel \\
 &\quad lane[0, 2] \parallel lane[1, 2] \parallel \dots \parallel lane[4, 2] \parallel \\
 &\quad lane[0, 3] \parallel lane[1, 3] \parallel \dots \parallel lane[4, 3] \parallel \\
 &\quad lane[0, 4] \parallel lane[1, 4] \parallel \dots \parallel lane[4, 4] \\
 &= bit[0, 0, 0] \parallel bit[0, 0, 1] \parallel bit[0, 0, 2] \parallel \dots \parallel bit[0, 0, 63] \parallel \\
 &\quad bit[1, 0, 0] \parallel bit[1, 0, 1] \parallel bit[1, 0, 2] \parallel \dots \parallel bit[1, 0, 63] \parallel \\
 &\quad bit[2, 0, 0] \parallel bit[2, 0, 1] \parallel bit[2, 0, 2] \parallel \dots \parallel bit[2, 0, 63] \parallel \\
 &\quad \dots \\
 &\quad bit[3, 4, 0] \parallel bit[3, 4, 1] \parallel bit[3, 4, 2] \parallel \dots \parallel bit[3, 4, 63] \parallel \\
 &\quad bit[4, 4, 0] \parallel bit[4, 4, 1] \parallel bit[4, 4, 2] \parallel \dots \parallel bit[4, 4, 63]
 \end{aligned}$$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- The sponge construction operates in 2 phases:
 - In the **absorbing** or **input phase**, the n message blocks x_0, x_1, \dots, x_{n-1} are consumed and read into the state
 - In the **squeezing** or **output phase**, an output y_0, y_1, y_2, \dots of configurable length is generated from the state



6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- There are a few parameters to configure the input and output sizes as well as the security of KECCAK
 - The **state width** b can take any value $b = 5 \cdot 5 \cdot 2^l = 25 \cdot 2^l$ for $l = 0, 1, \dots, 6$ (i.e., 25, 50, 100, 200, 400, 800, or 1600 bits)
 - The **bit rate** r determines the number of input bits that are processed simultaneously
 - The **capacity** c refers to the double security level of the construction
- In either case, $b = r + c$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

Table 6.6

The KECCAK Parameter Values for the SHA-3 Hash Functions

Hash Function	n	b	r	c	w
SHA3-224	224	1600	1152	448	64
SHA3-256	256	1600	1088	512	64
SHA3-384	384	1600	832	768	64
SHA3-512	512	1600	576	1024	64

- Note that $b = 1600$ and $w = 64$ in all versions of SHA-3

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- Before a message m can be processed, it must be padded properly (to make sure that the input is a multiple of r bits long)
- It uses a padding scheme known as **multirate padding**

$$\text{Padding}(m) = \underbrace{m \parallel p \parallel 10^*1}_{\text{multiple of } r}$$

- The value of bit string p depends on the mode
 - 2-bit string 01 for hashing
 - 4-bit string 1111 for generating a variable-length output

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- The sponge construction is based on a permutation of the state (called f -function or f -permutation)
- The same f -function is used in the absorbing and squeezing phases
- It takes $b = r + c$ bits as input and generates an output of the same length
- Internally, the f -function consists of n_r round functions with the same input and output behavior

6. Cryptographic Hash Functions

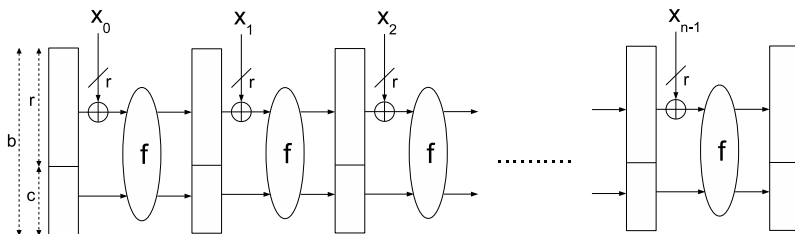
6.4 Exemplary Hash Functions — KECCAK/SHA-3

- Remember that l determines the state width according to $b = 25 \cdot 2^l$ (SHA-3 uses the fixed values $l = 6$ and hence $b = 1600$)
- The value l also determines n_r , i.e., the number of rounds, according to $n_r = 12 + 2l$
- So the possible state widths 25, 50, 100, 200, 400, 800, and 1600 come along with respective numbers of rounds, i.e., 12, 14, 16, 18, 20, 22, and 24
- As SHA-3 fixes the state width to 1600 bits, the number of rounds is also fixed to 24, i.e., $n_r = 24$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

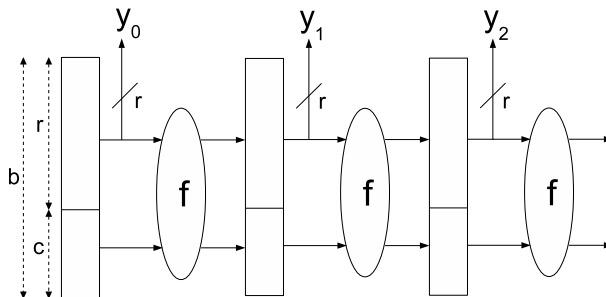
■ KECCAK absorbing phase



6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

■ KECCAK squeezing phase



6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- In each round, a sequence of five step mappings is executed, where each mapping operates on the b bits of the state
- Each step mapping takes a state array \mathbf{A} as input and returns an updated state array \mathbf{A}' as output
- The five step mappings are denoted by Greek letters, i.e., theta (θ), rho (ρ), pi (π), chi (χ), and iota (ι)
- While θ must be applied first, the order of the other mappings is arbitrary and does not matter (and ρ and π are often applied simultaneously)

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- The step mappings are relatively simple to capture visually, but more difficult to capture mathematically
- The x - and y -axes are labeled in an unusual manner

Table 6.7

The (x, y) -Coordinates of the Bits in a Slice

(3,2)	(4,2)	(0,2)	(1,2)	(2,2)
(3,1)	(4,1)	(0,1)	(1,1)	(2,1)
(3,0)	(4,0)	(0,0)	(1,0)	(2,0)
(3,4)	(4,4)	(0,4)	(1,4)	(2,4)
(3,3)	(4,3)	(0,3)	(1,3)	(2,3)

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

6.4 Exemplary Hash Functions — KECCAK/SHA-3

A 3D schematic diagram of a quantum system. A rectangular prism represents a volume with a grid of points. A central column of four cubes is highlighted. Above this column, two circles represent particles, with dashed arrows indicating their interaction with the central cubes and each other. A coordinate system (x , y , z) is shown at the bottom left.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺ ↻

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

$$\begin{aligned}
 \mathbf{A}'[x_0, y_0, z_0] = \mathbf{A}[x_0, y_0, z_0] &\oplus \bigoplus_{y=0}^4 \mathbf{A}[(x_0 - 1) \bmod 5, y, z_0] \\
 &\oplus \bigoplus_{y=0}^4 \mathbf{A}[(x_0 + 1) \bmod 5, y, (z_0 - 1) \bmod w]
 \end{aligned}$$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

■ Algorithm to compute θ (theta)

(A)

```

for x = 0 to 4 do
  for z = 0 to w - 1 do
    C[x, z] = A[x, 0, z] ⊕ A[x, 1, z] ⊕ A[x, 2, z] ⊕ A[x, 3, z] ⊕ A[x, 4, z]
  for x = 0 to 4 do
    for z = 0 to w - 1 do
      D[x, z] = C[(x - 1) mod 5, z] ⊕ C[(x + 1) mod 5, (z - 1) mod w]
    for x = 0 to 4 do
      for y = 0 to 4 do
        for z = 0 to w - 1 do
          A'[x, y, z] = A[x, y, z] ⊕ D[x, z]

```

(A')

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- **Step mapping ρ (rho)** rotates the bits in each lane for a certain amount of bits (offset), while **step mapping π (pi)** permutes the position of the lanes
- Both mappings can be combined and expressed as

$$\text{lane}[y, 2x + 3y] = \text{lane}[x, y] \xrightarrow{r} r[x, y]$$

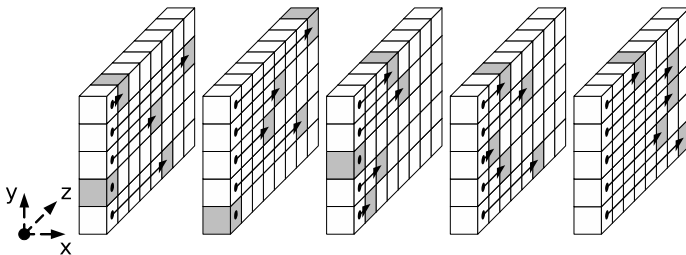
or

$$\mathbf{A}'[y, 2x + 3y, \cdot] = \mathbf{A}[x, y, \cdot] \xrightarrow{r} r[x, y]$$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

■ Step mapping ρ (rho)



© keccak.team

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

Table 6.8 (new)

The Offset Values Used by the Step Mapping ρ

	$x = 3$	$x = 4$	$x = 0$	$x = 1$	$x = 2$
$y = 2$	25	39	3	10	43
$y = 1$	55	20	36	44	6
$y = 0$	28	27	0	1	62
$y = 4$	56	14	18	2	61
$y = 3$	21	8	41	45	15

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

■ Algorithm to compute ρ (rho)

(A)

for $z = 0$ to $w - 1$ do $\mathbf{A}'[0, 0, z] = \mathbf{A}[0, 0, z]$

$(x, y) = (1, 0)$

for $t = 0$ to 23 do

 for $z = 0$ to $w - 1$ do $\mathbf{A}'[x, y, z] = \mathbf{A}[x, y, (z - (t + 1)(t + 2)/2) \bmod w]$

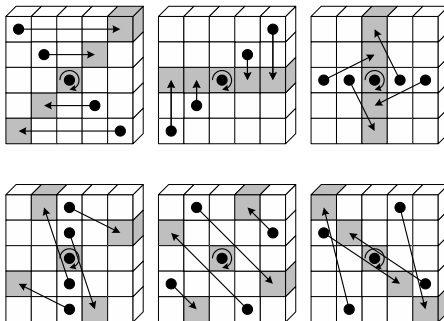
$(x, y) = (y, (2x + 3y) \bmod 5)$

(A')

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

■ Step mapping π (pi)



© keccak.team

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

■ Algorithm to compute π (pi)

(A)

for $x = 0$ to 4 do

 for $y = 1$ to 4 do

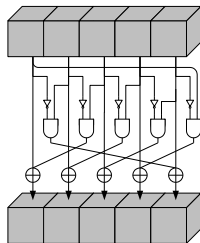
 for $z = 0$ to $w - 1$ do $\mathbf{A}'[x, y, z] = \mathbf{A}[(x + 3y) \bmod 5, x, z]$

(A')

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- Step mapping χ (chi) operates on lanes
- It combines $lane[x, y]$ with $lane[x + 1, y]$ and $lane[x + 2, y]$ with the Boolean NOT (\neg) XOR (\oplus), and AND (\wedge) operators
- It is the only nonlinear step mapping in the round function of KECCAK



© keccak.team

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- Algorithm to compute χ (chi)

```

for x = 0 to 4 do
  for y = 1 to 4 do
    for z = 0 to w - 1 do
       $\mathbf{A}'[x, y, z] = \mathbf{A}[x, y, z] \oplus ((\mathbf{A}[(x + 1) \bmod 5, y, z] \oplus 1) \cdot \mathbf{A}[(x + 2) \bmod 5, y, z])$ 

```

A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- $$\mathbf{A}'[0, 0, \cdot] = \mathbf{A}[0, 0, \cdot] \oplus \text{RC}[i_r]$$

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

Table 6.9 (new)

The 24 Round Constants $RC[i_r]$ Employed by SHA-3

$RC[0]$	0x0000000000000001	$RC[12]$	0x000000008000808B
$RC[1]$	0x0000000000008082	$RC[13]$	0x800000000000008B
$RC[2]$	0x800000000000808A	$RC[14]$	0x8000000000008089
$RC[3]$	0x8000000080008000	$RC[15]$	0x8000000000008003
$RC[4]$	0x000000000000808B	$RC[16]$	0x8000000000008002
$RC[5]$	0x0000000080000001	$RC[17]$	0x8000000000000080
$RC[6]$	0x8000000080008081	$RC[18]$	0x000000000000800A
$RC[7]$	0x8000000000008009	$RC[19]$	0x800000008000000A
$RC[8]$	0x000000000000008A	$RC[20]$	0x8000000080008081
$RC[9]$	0x0000000000000088	$RC[21]$	0x8000000000008080
$RC[10]$	0x0000000080008009	$RC[22]$	0x0000000080000001
$RC[11]$	0x000000008000000A	$RC[23]$	0x8000000080008008

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- Given a state \mathbf{A} and round index i_r , the round function Rnd is defined as

$$\text{Rnd}(\mathbf{A}, i_r) = \iota(\chi(\pi(\rho(\theta(\mathbf{A}))))), i_r)$$

- The KECCAK- $p[b, n_r]$ permutation consists of n_r iterations of Rnd:

(S, n_r)

convert S into state \mathbf{A}

for $i_r = 2l + 12 - n_r, \dots, 2l + 12 - 1$ do $\mathbf{A} = \text{Rnd}(\mathbf{A}, i_r)$

convert \mathbf{A} into b -bit string S'

(S')

6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions — KECCAK/SHA-3

- The KECCAK- f family of permutations refers to the specialization of the KECCAK- p family with $n_r = 12 + 12l$:

$$\text{KECCAK-}f[b] = \text{KECCAK-}p[b, 12 + 12l]$$

- The KECCAK- $p[1600, 24]$ permutation that underlies the six SHA-3 functions is equivalent to KECCAK- $f[1600]$
- There is no known attack against KECCAK/SHA-3
- But KECCAK/SHA-3 is still not widely deployed in the field

6.5 Exemplary Hash Functions — Final Remarks

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

6.5 Exemplary Hash Functions — Final Remarks

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

6. Cryptographic Hash Functions

6.5 Exemplary Hash Functions — Final Remarks

- An alternative design for cryptographic hash functions was proposed by Larry Carter and Mark Wegman in late 1970s
- Instead of using a single hash function, it uses families of such functions from which a specific function is randomly selected
- Such a family H consists of all hash functions $h : X \rightarrow Y$ that map values from X to values from Y
- H is called **two-universal**, if for every $x, y \in X$ with $x \neq y$

$$\Pr_{h \leftarrow H}[h(x) = h(y)] \leq \frac{1}{|Y|}$$

6.5 Exemplary Hash Functions — Final Remarks

- This suggests that the images are uniformly distributed in Y , and that the probability of having two images collide is as small as possible (given the size of Y)
- This notion of universality can be generalized
- Using (two-)universal families of hash functions is referred to as **universal hashing**
- Universal hashing is the basic ingredient for **Carter-Wegman MACs** (as further addressed in Chapter 10)

Questions and Answers



