## Cryptography 101: From Theory to Practice

## Chapter 6 - Cryptographic Hash Functions

Rolf Oppliger

March 1, 2022

## Terms of Use

- This work is published with a CC BY-ND 4.0 license (@(i) ()
- CC = Creative Commons (@)
- BY = Attribution (©)
- ND = No Derivatives ( $\Theta$ )


## whoami



$$
\begin{aligned}
& \text { rolf-oppliger.ch } \\
& \text { rolf-oppliger.com }
\end{aligned}
$$

■ 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


(c) Artech House, 2021 ISBN 978-1-63081-846-3
https://books.esecurity.ch/crypto101.html

## Challenge Me



## Outline

## 6. Cryptographic Hash Functions

1 Introduction
2 Cryptographic Systems
3 Random Generators
4 Random Functions
5 One-Way Functions
7 Pseudorandom Generators
8 Pseudorandom Functions
9 Symmetric Encryption
10 Message Authentication
11 Authenticated Encryption
12 Key Establishment
13 Asymmetric Encryption
14 Digital Signatures
15 Zero-Knowledge Proofs of Knowledge
16 Key Management
17 Summary
18 Outlook

## 6. Cryptographic Hash Functions

6.1 Introduction
6.2 Merkle-Damgård Construction
6.3 Historical Perspective
6.4 Exemplary Hash Functions
6.5 Final Remarks

## 6. Cryptographic Hash Functions

### 6.1 Introduction

- According to Definition 2.5, a hash function is an efficiently computable function $h: \Sigma_{\text {in }}^{*} \rightarrow \Sigma_{\text {out }}^{n}$ that maps $x \in \Sigma_{\text {in }}^{*}$ to $y \in \sum_{\text {out }}^{n}$ (for a fixed output length $n$ )
- Additional properties
- $H$ is one-way (preimage resistant) if it is computationally infeasible to find $x \in \sum_{\text {in }}^{*}$ with $h(x)=y$ for $y \in_{R} \sum_{\text {out }}^{n}$
- $H$ is second-preimage resistant (weak collision resistant) if it is computationally infeasible to find $x^{\prime} \in \Sigma_{i n}^{*}$ with $x^{\prime} \neq x$ and $h\left(x^{\prime}\right)=h(x)$ for $x \in_{R} \sum_{\text {in }}^{*}$
- $H$ is collision-resistant (strong collision resistant) if it is computationally infeasible to find $x, x^{\prime} \in \sum_{i n}^{*}$ with $x^{\prime} \neq x$ and $h\left(x^{\prime}\right)=h(x)$


## 6. Cryptographic Hash Functions

### 6.1 Introduction

■ According to Definition 2.6, a hash function $h$ is cryptographic if it is one-way and either second-preimage resistant or collision-resistant

- Remarks (1)
- Due to the pigeonhole principle, the term "collision free" is wrong and should not be used here
- If one wants to use complexity-theoretic arguments, then one must consider families of (cryptographic) hash functions
- Collision resistance implies second-preimage resistance, but not vice versa (this is why the terms "weak collision resistant" and "strong collision resistant" are used in the first place)


## 6. Cryptographic Hash Functions

### 6.1 Introduction

- Remarks (2)
- 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 $h(x)=g\left(\left.x\right|_{n}\right)$ is still preimage resistant but not collision-resistant

- All $x \| y$ (with $|x| \geq n$ and $y$ arbitrary) hash to the same value and yield collisions


## 6. Cryptographic Hash Functions

### 6.1 Introduction

- Remarks (3)
- On the other hand, a (strong or weak) collision-resistant hash function need not be preimage resistant (e.g., Maurer's counterexample)
- If $g$ is an $n$-bit collision-resistant hash function, then the ( $n+1$ )-bit hash function

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

■ is still collision-resistant but not preimage resistant

- For all $h(x)$ that begin with 1 , it is trivial to find a preimage (just drop the 1)


## 6. Cryptographic Hash Functions

### 6.1 Introduction

- The notion of a collision can be generalized to multicollisions

■ More specifically, an $r$-collision is an $r$-tuple $\left(x_{1}, \ldots, x_{r}\right)$ with $h\left(x_{1}\right)=\ldots=h\left(x_{r}\right)$
■ For $r=2$, a 2-collision is a "normal" collision

- Finding multicollisions is not substantially more difficult than finding "normal" collisions


## 6. Cryptographic Hash Functions

### 6.1 Introduction

■ In practice, $\Sigma_{\text {in }}$ and $\Sigma_{\text {out }}$ 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. Cryptographic Hash Functions

### 6.1 Introduction

■ In general, there are many ways to construct a cryptographic hash function
■ According to ISO/IEC 10118-1

- Hash functions that employ block ciphers (ISO/IEC 10118-2)
- Dedicated hash functions (ISO/IEC 10118-3)
- Hash functions based on modular arithmetic (ISO/IEC 10118-4)
■ Mainly due to their performance advantages, dedicated hash functions are usually the preferred choice
- Most functions employ the Merkle-Damgård construction and yield iterated hash functions


## 6. Cryptographic Hash Functions

### 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 $f: \Sigma^{b+\prime} \longrightarrow \Sigma^{\prime}$ (with $b, I \in \mathbb{N}$ ) into an iterated hash function $h$


## 6. Cryptographic Hash Functions

### 6.2 Merkle-Damgård Construction

- There are many possibilities to design and come up with a compression function $f$
■ A popular (i.e., widely used) possibility is a Davies-Meyer compression function
- It applies a block cipher $E$ on a chaining value $H_{i}$, where the respective message block $x_{i}$ serves as the key:

$$
H_{i}=E_{x_{i}}\left(H_{i-1}\right) \oplus H_{i-1} \quad \text { for } \quad i=1, \ldots, n
$$

## 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:

$$
\begin{aligned}
H_{0} & =I V \\
H_{i} & =f\left(H_{i-1}, x_{i}\right) \text { for } i=1, \ldots, n \\
h(x) & =g\left(H_{n}\right)
\end{aligned}
$$

- 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 0 s , and the binary encoding of the message length


## 6. Cryptographic Hash Functions

### 6.2 Merkle-Damgård Construction

- Merkle and Damgård showed that finding a collision for $h$ is at least as hard as finding a collision for $f$

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. Cryptographic Hash Functions

### 6.3 Historical Perspective

- The first cryptographic hash function was developed in the 1980s by RSA Security (acronymed MD for "message digest")
- It was proprietary and never published

■ MD2 (RFC 1319) was the first cryptographic hash function that was published and used in the field

- After the announcemnet of SNEFRU by Ralph C. Merkle, RSA Security released MD4 (RFC 1320)
- In 1991, SNEFRU and some other hash functions were broken and weaknesses were found in MD4

■ As a result, RSA Security came up with MD5 (RFC 1321) and released RFCs 1319-1321 in April 1992

## 6. Cryptographic Hash Functions

### 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. Cryptographic Hash Functions

### 6.3 Historical Perspective

■ During the 1990s, a series of results showed that MD4 was insecure and MD5 was partially broken
■ In the early 2000s, a group of Chinese researchers (headed by Xiaoyun Wang) published collisions for MD4, MD5, and a few other cryptographic hash functions

- In 2008, a Dutch research group (headed by Arjen Lenstra) found a way to exploit an MD5 collision to create a rogue CA certificate
- Consequently, MD4, MD5, and a few other cryptographic hash functions should no longer be used (they may still serve as study objects)


## 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. Cryptographic Hash Functions <br> 6.4 Exemplary Hash Functions - MD4

- MD4 follows the Merkle-Damgård construction and uses a Davies-Meyer compression function with $b=512$ and $I=128$
- 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 $a_{1} a_{2} a_{3} a_{4}$ is stored as $a_{4} a_{3} a_{2} a_{1}$, representing the integer $a_{4} 2^{24}+a_{3} 2^{16}+a_{2} 2^{8}+a_{1}$


## 6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions - MD4

- Let $m=m_{0} m_{1} \ldots m_{s-1}$ be an $s$-bit message

■ MD4 first generates an array $w$ of $n$ 32-bit words

$$
w=w[0]\|[1]\| \ldots \| w[n-1]
$$

where $n$ is a multiple of 16 , i.e., $n \equiv 0(\bmod 16)$

- Hence, the bitlength of $w$ is a multiple of $32 \cdot 16=512$ bits

$$
\begin{aligned}
w[0] & =m_{0} m_{1} \ldots m_{31} \\
w[1] & =m_{32} m_{33} \ldots m_{63} \\
& \ldots \\
w[n-1] & =m_{s-32} m_{s-31} \ldots m_{s-1}
\end{aligned}
$$

## 6. Cryptographic Hash Functions

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 $0 s$ ) 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 $w$ )

| Original message | 10000000000000 | $(\mathrm{~S})_{2}$ |
| :---: | :--- | :--- |
|  |  |  |

## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - MD4

- $F, g$, and $h$ are logical functions
$f(X, Y, Z)=(X \wedge Y) \vee((\neg X) \wedge Z)$
$g(X, Y, Z)=(X \wedge Y) \vee(X \wedge Z) \vee(Y \wedge Z)$
$h(X, Y, Z)=X \oplus Y \oplus Z$
- $C_{1}$ and $c_{2}$ are constants
- $W \stackrel{\curvearrowright}{\hookleftarrow} c$ 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

## Overview:

(m)

Construct $w=w[0]\|w[1]\| \ldots \| w[n-1]$
$A=0 \times 67452301$
$B=0 \times E F C D A B 89$
$C=0 \times 98 \mathrm{BADCFE}$
$D=0 \times 10325476$
for $i=0$ to $n / 16-1$ do

$$
\begin{aligned}
& \text { for } j=0 \text { to } 15 \text { do } X[j]=w[i \cdot 16+j] \\
& A^{\prime}=A \\
& B^{\prime}=B \\
& C^{\prime}=C \\
& D^{\prime}=D
\end{aligned}
$$

Round 1
Round 2
Round 3

$$
\begin{aligned}
& A=A+A^{\prime} \\
& B=B+B^{\prime} \\
& C=C+C^{\prime} \\
& D=D+D^{\prime}
\end{aligned}
$$

$(h(m)=A\|B\| C \| D)$

## Round 1:

1. $A=(A+f(B, C, D)+X[0]) \stackrel{\curvearrowright}{\rightleftarrows} 3$
2. $D=(D+f(A, B, C)+X[1]) \stackrel{\rightleftarrows}{\rightleftarrows}$
3. $C=(C+f(D, A, B)+X[2]) \stackrel{\subsetneq}{\rightleftarrows} 11$
4. $B=(B+f(C, D, A)+X[3]) \stackrel{\smile}{\hookleftarrow} 19$
5. $A=(A+f(B, C, D)+X[4]) \stackrel{\curvearrowright}{\rightleftarrows} 3$
6. $D=(D+f(A, B, C)+X[5]) \stackrel{\rightleftarrows}{\rightleftarrows}$
7. $C=(C+f(D, A, B)+X[6]) \stackrel{\curvearrowright}{\leftrightarrows} 11$
8. $B=(B+f(C, D, A)+X[7]) \stackrel{\curvearrowright}{\hookleftarrow} 19$
9. $A=(A+f(B, C, D)+X[8]) \stackrel{\curvearrowright}{\rightleftarrows} 3$
10. $D=(D+f(A, B, C)+X[9]) \stackrel{\curvearrowright}{\rightleftarrows}$
11. $C=(C+f(D, A, B)+X[10]) \stackrel{\curvearrowright}{\hookleftarrow} 11$
12. $B=(B+f(C, D, A)+X[11]) \stackrel{\curvearrowright}{\rightleftarrows} 19$
13. $A=(A+f(B, C, D)+X[12]) \stackrel{\curvearrowright}{\rightleftarrows}$
14. $D=(D+f(A, B, C)+X[13]) \stackrel{\curvearrowright}{\hookleftarrow}$
15. $C=(C+f(D, A, B)+X[14]) \stackrel{\curvearrowright}{\hookleftarrow} 11$
16. $B=(B+f(C, D, A)+X[15]) \stackrel{\rightleftharpoons}{\rightleftarrows} 19$

## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - MD4

## Round 2:

1. $A=\left(A+g(B, C, D)+X[0]+c_{1}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
2. $D=\left(D+g(A, B, C)+X[4]+c_{1}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
3. $C=\left(C+g(D, A, B)+X[8]+c_{1}\right) \stackrel{\rightleftarrows}{\rightleftarrows} 9$
4. $B=\left(B+g(C, D, A)+X[12]+c_{1}\right) \stackrel{\curvearrowright}{\rightleftarrows} 13$
5. $A=\left(A+g(B, C, D)+X[1]+c_{1}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
6. $D=\left(D+g(A, B, C)+X[5]+c_{1}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
7. $C=\left(C+g(D, A, B)+X[9]+c_{1}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
8. $B=\left(B+g(C, D, A)+X[13]+c_{1}\right) \stackrel{\curvearrowright}{\hookleftarrow} 13$
9. $A=\left(A+g(B, C, D)+X[2]+c_{1}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
10. $D=\left(D+g(A, B, C)+X[6]+c_{1}\right) \stackrel{\curvearrowright}{\rightleftarrows} 5$
11. $C=\left(C+g(D, A, B)+X[10]+c_{1}\right) \stackrel{\curvearrowright}{\rightleftarrows} 9$
12. $B=\left(B+g(C, D, A)+X[14]+c_{1}\right) \stackrel{\hookleftarrow}{\hookleftarrow} 13$
13. $A=\left(A+g(B, C, D)+X[3]+c_{1}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
14. $D=\left(D+g(A, B, C)+X[7]+c_{1}\right) \stackrel{\curvearrowright}{\hookleftarrow} 5$
15. $C=\left(C+g(D, A, B)+X[11]+c_{1}\right) \stackrel{\curvearrowright}{\rightleftarrows} 9$
16. $B=\left(B+g(C, D, A)+X[15]+c_{1}\right) \stackrel{\rightleftharpoons}{\rightleftarrows} 13$

## Round 3:

1. $A=\left(A+h(B, C, D)+X[0]+c_{2}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
2. $D=\left(D+h(A, B, C)+X[8]+c_{2}\right) \stackrel{\curvearrowright}{\rightleftarrows} 9$
3. $C=\left(C+h(D, A, B)+X[4]+c_{2}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
4. $B=\left(B+h(C, D, A)+X[12]+c_{2}\right) \stackrel{\curvearrowright}{\rightleftarrows}$
5. $A=\left(A+h(B, C, D)+X[2]+c_{2}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
6. $D=\left(D+h(A, B, C)+X[10]+c_{2}\right) \stackrel{\lessgtr}{\hookleftarrow} 9$
7. $C=\left(C+h(D, A, B)+X[6]+c_{2}\right) \stackrel{\curvearrowright}{\hookleftarrow} 11$
8. $B=\left(B+h(C, D, A)+X[14]+c_{2}\right) \stackrel{\curvearrowright}{\hookleftarrow} 15$
9. $A=\left(A+h(B, C, D)+X[1]+c_{2}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
10. $D=\left(D+h(A, B, C)+X[9]+c_{2}\right) \stackrel{\curvearrowright}{\rightleftarrows} 9$
11. $C=\left(C+h(D, A, B)+X[5]+c_{2}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
12. $B=\left(B+h(C, D, A)+X[13]+c_{2}\right) \stackrel{\rightleftharpoons}{\rightleftarrows} 15$
13. $A=\left(A+h(B, C, D)+X[3]+c_{2}\right) \stackrel{\rightleftarrows}{\rightleftarrows}$
14. $D=\left(D+h(A, B, C)+X[11]+c_{2}\right) \stackrel{\curvearrowright}{\rightleftarrows} 9$
15. $C=\left(C+h(D, A, B)+X[7]+c_{2}\right) \stackrel{\curvearrowright}{\hookleftarrow} 11$
16. $B=\left(B+h(C, D, A)+X[15]+c_{2}\right) \stackrel{\rightleftharpoons}{\hookleftarrow} 15$

## 6. Cryptographic Hash Functions <br> 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. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - MD5

## Overview:

$$
\begin{aligned}
& (m) \\
& \hline \text { Construct } w=w[0]\|w[1]\| \ldots \|[n-1] \\
& A=0 \times 67452301 \\
& B=0 \times E F C D A B 89 \\
& C=0 \times 98 B A D C F E \\
& D=0 \times 10325476 \\
& \text { for } i=0 \text { to } n / 16-1 \text { do } \\
& \text { for } j=0 \text { to } 15 \text { do } X[j]=w[i \cdot 16+j] \\
& A^{\prime}=A \\
& B^{\prime}=B \\
& C^{\prime}=C \\
& D^{\prime}=D \\
& \text { Round } 1 \\
& \text { Round } 2 \\
& \text { Round } 3 \\
& \text { Round } 4 \\
& A=A+A^{\prime} \\
& B=B+B^{\prime} \\
& C=C+C^{\prime} \\
& D=D+D^{\prime}
\end{aligned}
$$

## Round 1:

1. $A=(A+f(B, C, D)+X[0]+T[1]) \stackrel{\curvearrowright}{\rightleftarrows}$
2. $D=(D+f(A, B, C)+X[1]+T[2]) \stackrel{\leftarrow}{\hookleftarrow} 12$
3. $C=(C+f(D, A, B)+X[2]+T[3]) \stackrel{\rightleftarrows}{\rightleftarrows} 17$
4. $B=(B+f(C, D, A)+X[3]+T[4]) \stackrel{\rightleftarrows}{\rightleftarrows} 22$
5. $A=(A+f(B, C, D)+X[4]+T[5]) \stackrel{\curvearrowright}{\rightleftarrows}$
6. $D=(D+f(A, B, C)+X[5]+T[6]) \stackrel{\rightleftarrows}{\rightleftarrows} 12$
7. $C=(C+f(D, A, B)+X[6]+T[7]) \stackrel{\rightleftarrows}{\rightleftarrows}$
8. $B=(B+f(C, D, A)+X[7]+T[8]) \stackrel{\rightleftarrows}{\rightleftarrows} 22$
9. $A=(A+f(B, C, D)+X[8]+T[9]) \stackrel{\smile}{\rightleftarrows}$
10. $D=(D+f(A, B, C)+X[9]+T[10]) \stackrel{\curvearrowright}{\rightleftarrows} 12$
11. $C=(C+f(D, A, B)+X[10]+T[11]) \stackrel{(17}{\rightleftarrows}$
12. $B=(B+f(C, D, A)+X[11]+T[12]) \stackrel{\curvearrowright}{\rightleftarrows} 22$
13. $A=(A+f(B, C, D)+X[12]+T[13]) \stackrel{ }{\rightleftarrows} 7$
14. $D=(D+f(A, B, C)+X[13]+T[14]) \stackrel{\curvearrowright}{\rightleftarrows} 12$
15. $C=(C+f(D, A, B)+X[14]+T[15]) \stackrel{\curvearrowright}{\rightleftarrows} 17$
16. $B=(B+f(C, D, A)+X[15]+T[16]) \stackrel{\curvearrowright}{\rightleftarrows} 22$
$(h(m)=A\|B\| C \| D)$

## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - MD5

## Round 2:

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

## Round 3:

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

## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - MD5

## Round 4:

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

- 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 $f_{0}, f_{1}, \ldots, f_{79}$ :

$$
f_{t}(X, Y, Z)= \begin{cases}\operatorname{Ch}(X, Y, Z)=(X \wedge Y) \oplus((\neg X) \wedge Z) & 0 \leq t \leq 19 \\ P a r i t y \\ M, Y, Z)=X \oplus Y \oplus Z & 20 \leq t \leq 39 \\ \operatorname{Maj}(X, Y, Z)=(X \wedge Y) \oplus(X \wedge Z) \oplus(Y \wedge Z) & 40 \leq t \leq 59 \\ \operatorname{Parity}(X, Y, Z)=X \oplus Y \oplus Z & 60 \leq t \leq 79\end{cases}
$$

- Note that the Parity function occurs twice $(20 \leq t \leq 39$ and $60 \leq t \leq 79)$, and that $C h$ and $M a j$ are similar to $f$ and $g(V$ 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 $K_{0}, K_{1}, \ldots, K_{79}$ :

$$
K_{t}= \begin{cases}\left\lfloor 2^{30} \sqrt{2}\right\rfloor=0 \times 5 \text { A827999 } & 0 \leq t \leq 19 \\ \left\lfloor 2^{30} \sqrt{3}\right\rfloor=0 \times 6 \text { ED9EBA1 } & 20 \leq t \leq 39 \\ \left\lfloor 2^{30} \sqrt{5}\right\rfloor=0 \times 8 \text { F1BBCDC } & 40 \leq t \leq 59 \\ \left\lfloor 2^{30} \sqrt{10}\right\rfloor=0 \times \text { CA62C1D6 } & 60 \leq t \leq 79\end{cases}
$$

## 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, $b[i](i=0,1, \ldots, n-1)$ 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 $W$ :

$$
W_{t}= \begin{cases}b_{t} & 0 \leq t \leq 15 \\ \left(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}\right) \stackrel{\curvearrowright}{\hookleftarrow} & 16 \leq t \leq 79\end{cases}
$$

- 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 $b=b[0]\|b[1]\| \ldots \| b[n-1]$
$A=0 \times 67452301$
$B=0 \times E F C D A B 89$
$C=0 \times 98 \mathrm{BADCFE}$
$D=0 \times 10325476$
$E=0 \times C 3 D 2 E 1 F 0$
for $i=0$ to $n-1$ do
Derive message schedule $W$ from $b[i]$
$A^{\prime}=A$
$B^{\prime}=B$
$C^{\prime}=C$
$D^{\prime}=D$
$E^{\prime}=E$
1

$$
\begin{aligned}
& \text { for } t=0 \text { to } 79 \text { do } \\
& T=(A \stackrel{\sim}{\hookleftarrow} 5)+f_{t}(B, C, D)+E+K_{t}+W_{t} \\
& E=D \\
& D=C \\
& C=B \rightleftarrows 30 \\
& B=A \\
& A=T \\
& A=A+A^{\prime} \\
& B=B+B^{\prime} \\
& C=C+C^{\prime} \\
& D=D+D^{\prime} \\
& E=E+E^{\prime} \\
& \hline(h(m)=A\|B\| C\|D\| E)
\end{aligned}
$$

## 6. Cryptographic Hash Functions <br> 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:

$$
\begin{aligned}
& \Sigma_{0}^{\{256\}}(X)=(X \bumpeq 2) \oplus(X \bumpeq 13) \oplus(X \bumpeq 22) \\
& \Sigma_{1}^{\{256\}}(X)=(X \bumpeq 6) \oplus(X \bumpeq 11) \oplus(X \leadsto 25) \\
& \sigma_{0}^{\{256\}}(X)=(X \leadsto 7) \oplus(X \stackrel{\leftrightharpoons}{\hookrightarrow} 18) \oplus(X \hookrightarrow 3) \\
& \sigma_{1}^{\{256\}}(X)=(X \xlongequal[\hookrightarrow]{\leftrightharpoons} 17) \oplus(X \bumpeq 19) \oplus(X \hookrightarrow 10)
\end{aligned}
$$

■ Note that $\hookrightarrow$ 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:

$$
\begin{aligned}
& \sum_{0}^{\{512\}}(X)=(X \xlongequal[\hookrightarrow]{\bumpeq} 28) \oplus(X \bumpeq 34) \oplus(X \bumpeq 39) \\
& \sum_{1}^{\{512\}}(X)=(X \xlongequal[\hookrightarrow]{\leftrightharpoons} 14) \oplus(X \bumpeq 18) \oplus(X \hookrightarrow 41) \\
& \sigma_{0}^{\{512\}}(X)=(X \bumpeq 1) \oplus(X \bumpeq 8) \oplus(X \hookrightarrow 7) \\
& \sigma_{1}^{\{512\}}(X)=(X \leadsto 19) \oplus(X \leadsto 61) \oplus(X \hookrightarrow 6)
\end{aligned}
$$

- All $\sum$-functions are used in the round functions, whereas all $\sigma$-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 $K_{0}, K_{1}, \ldots, K_{79}$, SHA-224 and SHA-256 use a sequence of 64 distinct 32-bit words that serve as constants

$$
K_{0}^{\{256\}}, K_{1}^{\{256\}}, \ldots, K_{63}^{\{256\}}
$$

- 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

$$
K_{0}^{\{512\}}, K_{1}^{\{512\}}, \ldots, K_{79}^{\{512\}}
$$

- 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

■ SHA-256 uses message schedule $W$ (with 64 words) and two temporary variables $T_{1}$ and $T_{2}$ ( $\rightarrow$ animation)
■ SHA-224 uses different initialization values and truncates the output to 224 bits
(m)

```
    Construct \(b=b[0]\|b[1]\| \ldots \| b[n-1]\)
    \(A=0 \times 6\) A09E667 \(\quad B=0 \times B B 67\) AE85
    \(C=0 \times 3\) C6EF372 \(\quad D=0 \times 454 F F 53 A\)
    \(E=0 \times 510 \mathrm{E} 527 \mathrm{~F} \quad F=0 \times 9 \mathrm{~B} 05688 \mathrm{C}\)
    \(G=0 \times 1\) F83D9AB \(\quad H=0 \times 5 B E 0 C D 19\)
    for \(i=0\) to \(n-1\) do
        Derive message schedule \(W\) from \(b[i]\)
        \(A^{\prime}=A \quad B^{\prime}=B \quad C^{\prime}=C \quad D^{\prime}=D\)
        \(E^{\prime}=E \quad F^{\prime}=F \quad G^{\prime}=G \quad H^{\prime}=H\)
        for \(t=0\) to 63 do
            \(T_{1}=H+\Sigma_{1}^{\{256\}}(E)+C h(E, F, G)+K_{t}^{\{256\}}+W_{t}\)
            \(T_{2}=\Sigma_{0}^{\{256\}}(A)+\operatorname{Maj}(A, B, C)\)
            \(H=G \quad G=F\)
            \(E=D+T_{1} \quad D=C\)
            \(C=B \quad B=A\)
            \(A=T_{1}+T_{2}\)
            \(A=A+A^{\prime} \quad B=B+B^{\prime} \quad C=C+C^{\prime}\)
            \(D=D+D^{\prime} \quad E=E+E^{\prime} \quad F=F+F^{\prime}\)
            \(G=G+G^{\prime} \quad H=H+H^{\prime}\)
                \((h(m)=A\|B\| C\|D\| E\|F\| G \| H)\)
```


## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - SHA-2 family

- SHA-512 uses 64-bit words, the same message schedule $W$ (with 80 words), and $T_{1}$ and $T_{2}$
■ SHA-384, SHA-512/ 224, and SHA-512/ 256 use different initialization values and truncate the output
(m)

Construct $b=b[0]\|b[1]\| \ldots \| b[n-1]$
$A=0 \times 6 A 09 E 667 \mathrm{~F} 3 \mathrm{BCC} 908 \quad B=0 \times B B 67$ AE8584CAA73B
$C=0 \times 3 C 6 E F 372 \mathrm{FE} 94 \mathrm{~F} 82 \mathrm{~B} \quad D=0 \times \mathrm{A} 54 \mathrm{FF} 53 \mathrm{~A} 5 \mathrm{~F} 1 \mathrm{D} 36 \mathrm{~F} 1$
$E=0 \times 510$ E527FADE682D $1 \quad F=0 \times 9 B 05688 \mathrm{C} 2 \mathrm{~B} 3 \mathrm{E} 6 \mathrm{C} 1 \mathrm{~F}$
$G=0 \times 1$ F83D9ABFB41BD6B $\quad H=0 \times 5 B E 0 C D 19137 E 2179$
for $i=0$ to $n-1$ do
Derive message schedule $W$ from $b[i]$

$$
\begin{array}{llll}
A^{\prime}=A & B^{\prime}=B & C^{\prime}=C & D^{\prime}=D \\
E^{\prime}=E & F^{\prime}=F & G^{\prime}=G & H^{\prime}=H
\end{array}
$$

$$
\text { for } t=0 \text { to } 79 \text { do }
$$

$$
T_{1}=H+\sum_{1}^{\{512\}}(E)+C h(E, F, G)+K_{t}^{\{512\}}+W_{t}
$$

$$
T_{2}=\Sigma_{0}^{\{512\}}(A)+\operatorname{Maj}(A, B, C)
$$

$$
H=G \quad G=F
$$

$$
E=D+T_{1} \quad D=C
$$

$$
C=B \quad B=A
$$

$$
A=T_{1}+T_{2}
$$

$$
\begin{array}{lll}
A=A+A^{\prime} & B=B+B^{\prime} & C=C+C^{\prime}
\end{array} \quad D=D+D^{\prime},
$$

$$
E=E+E^{\prime} \quad F=F+F^{\prime} \quad G=G+G^{\prime} \quad H=H+H^{\prime}
$$

$$
(h(m)=A\|B\| C\|D\| E\|F\| G \| H)
$$

## 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. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

- The sponge construction is based on a permutation operating on a data structure known as the state

■ The state can either be seen as a (one-dimensional) $b$-bit string $S$ or a three-dimensional array $\mathbf{A}[x, y, z]$ of bits with appropriate values for $x, y$, and $z$ (i.e., $x y z \leq b$ )

(C) keccak.team

## 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 $w=2^{\prime}=64$ for $I=6$ )

- 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:

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

- $\mathbf{A}[0,0,0]$ 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

$$
\begin{aligned}
S= & \mathbf{A}=\text { plane }[0] \| \text { plane }[1]\|\ldots\| \text { plane }[4] \\
= & \text { lane }[0,0] \| \text { lane }[1,0]\|\ldots\| \text { lane }[4,0] \| \\
& \text { lane }[0,1] \| \text { lane }[1,1]\|\ldots\| \text { lane } 4,1] \| \\
& \text { lane }[0,2] \| \text { lane }[1,2]\|\ldots\| \text { lane } 4,2] \| \\
& \text { lane }[0,3] \| \text { lane }[1,3]\|\ldots\| \text { lane }[4,3] \| \\
& \text { lane }[0,4] \| \text { lane }[1,4]\|\ldots\| \text { lane }[4,4] \\
= & \operatorname{bit}[0,0,0]\|\operatorname{bit}[0,0,1]\| \operatorname{bit}[0,0,2]\|\ldots\| \operatorname{bit}[0,0,63] \| \\
& \operatorname{bit}[1,0,0]\|\operatorname{bit}[1,0,1]\| \operatorname{bit}[1,0,2]\|\ldots\| \operatorname{bit}[1,0,63] \| \\
& \operatorname{bit}[2,0,0]\|\operatorname{bit}[2,0,1]\| \operatorname{bit}[2,0,2]\|\ldots\| \operatorname{bit}[2,0,63] \| \\
& \ldots \\
& \operatorname{bit}[3,4,0]\|\operatorname{bit}[3,4,1]\| \operatorname{bit}[3,4,2]\|\ldots\| \operatorname{bit}[3,4,63] \| \\
& \operatorname{bit}[4,4,0]\|\operatorname{bit}[4,4,1]\| \operatorname{bit}[4,4,2]\|\ldots\| \operatorname{bit}[4,4,63]
\end{aligned}
$$

## 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 $x_{0}, x_{1}, \ldots, x_{n-1}$ are consumed and read into the state
■ In the squeezing or output phase, an output $y_{0}, y_{1}, y_{2}, \ldots$ of configurable length is generated from the state

Keccak

(C) keccak.team

## 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 $b=5 \cdot 5 \cdot 2^{\prime}=25 \cdot 2^{\prime}$ for $I=0,1, \ldots, 6$ (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

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

- 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 <br> 6.4 Exemplary Hash Functions - Keccak/SHA-3

■ Remember that I determines the state width according to $b=25 \cdot 2^{\prime}$ (SHA-3 uses the fixed values $I=6$ and hence $b=1600$ )
■ The value $I$ also determines $n_{r}$, i.e., the number of rounds, according to $n_{r}=12+2 l$
■ 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 $\mathbf{A}^{\prime}$ as output
- The five step mappings are denoted by Greek letters, i.e., theta $(\theta)$, rho $(\rho)$, pi $(\pi)$, chi $(\chi)$, and iota $(\iota)$
- While $\theta$ must be applied first, the order of the other mappings is arbitrary and does not matter (and $\rho$ and $\pi$ 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. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

- The step mappings mainly operate on lanes, i.e., $w$-bit words that can be processed in a register on a modern processor (lane $[x, y]$ refers to $\mathbf{A}[x, y, \cdot]$ )
- The (mathematical) operations include the addition and multiplication modulo 2, i.e., the bitwise addition and multiplication in GF(2)
- This suggests that the addition is equal to the Boolean XOR operation $(\oplus)$ and the multiplication is equal to the Boolean AND operation ( $\wedge$ )
- With the exception of the round constants $\mathrm{RC}\left[i_{r}\right]$ used in $\iota$ (iota), the step mappings are the same in all rounds


## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

■ Step mapping $\theta$ (theta)

© keccak.team

[^0]
## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

$$
\begin{aligned}
\mathbf{A}^{\prime}\left[x_{0}, y_{0}, z_{0}\right]=\mathbf{A}\left[x_{0}, y_{0}, z_{0}\right] & \oplus \bigoplus_{y=0}^{4} \mathbf{A}\left[\left(x_{0}-1\right) \bmod 5, y, z_{0}\right] \\
& \oplus \bigoplus_{y=0}^{4} \mathbf{A}\left[\left(x_{0}+1\right) \bmod 5, y,\left(z_{0}-1\right) \bmod w\right]
\end{aligned}
$$

## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

- Algorithm to compute $\theta$ (theta)
(A)

$$
\begin{aligned}
& \text { for } x=0 \text { to } 4 \text { do } \\
& \text { for } z=0 \text { to } w-1 \text { do } \\
& \mathrm{C}[x, z]=\mathbf{A}[x, 0, z] \oplus \mathbf{A}[x, 1, z] \oplus \mathbf{A}[x, 2, z] \oplus \mathbf{A}[x, 3, z] \oplus \mathbf{A}[x, 4, z] \\
& \text { for } x=0 \text { to } 4 \text { do } \\
& \text { for } z=0 \text { to } w-1 \text { do } \\
& \mathrm{D}[x, z]=\mathrm{C}[(x-1) \bmod 5, z] \oplus \mathrm{C}[(x+1) \bmod 5,(z-1) \bmod w] \\
& \text { for } x=0 \text { to } 4 \text { do } \\
& \text { for } y=0 \text { to } 4 \text { do } \\
& \text { for } z=0 \text { to } w-1 \text { do } \\
& \mathbf{A}^{\prime}[x, y, z]=\mathbf{A}[x, y, z] \oplus \mathrm{D}[x, z]
\end{aligned}
$$

( $A^{\prime}$ )

## 6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions - Keccak/SHA-3

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

$$
\text { lane }[y, 2 x+3 y]=\operatorname{lane}[x, y] \stackrel{\curvearrowleft}{\hookrightarrow} r[x, y]
$$

or

$$
\mathbf{A}^{\prime}[y, 2 x+3 y, \cdot]=\mathbf{A}[x, y, \cdot] \stackrel{\curvearrowleft}{\hookrightarrow} r[x, y]
$$

## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

- Step mapping $\rho$ (rho)

(c) 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 $\rho$

|  | $x=3$ | $x=4$ | $x=0$ | $x=1$ | $x=2$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| $y=2$ | 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 $\rho$ (rho)
(A)

$$
\begin{aligned}
& \text { for } z=0 \text { to } w-1 \text { do } \mathbf{A}^{\prime}[0,0, z]=\mathbf{A}[0,0, z] \\
& (x, y)=(1,0) \\
& \text { for } t=0 \text { to } 23 \text { do } \\
& \quad \text { for } z=0 \text { to } w-1 \text { do } \mathbf{A}^{\prime}[x, y, z]=\mathbf{A}[x, y,(z-(t+1)(t+2) / 2) \bmod w] \\
& \quad(x, y)=(y,(2 x+3 y) \bmod 5)
\end{aligned}
$$

## ( $\mathbf{A}^{\prime}$ )

## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

■ Step mapping $\pi$ (pi)

© keccak.team

Cryptography 101: From Theory to Practice

## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

- Algorithm to compute $\pi$ (pi)
(A)
for $x=0$ to 4 do

$$
\text { for } y=1 \text { to } 4 \text { do }
$$

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

( $\mathbf{A}^{\prime}$ )

## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

- Step mapping $\chi$ (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

(C) keccak.team


## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

$$
\mathbf{A}^{\prime}[x, y, \cdot]=\mathbf{A}[x, y, \cdot] \oplus((\neg \mathbf{A}[x+1, y, \cdot]) \wedge \mathbf{A}[x+2, y, \cdot])
$$

- Algorithm to compute $\chi$ (chi)
(A)

$$
\begin{aligned}
& \text { for } x=0 \text { to } 4 \text { do } \\
& \text { for } y=1 \text { to } 4 \text { do } \\
& \text { for } z=0 \text { to } w-1 \text { do } \\
& \qquad \mathbf{A}^{\prime}[x, y, z]=\mathbf{A}[x, y, z] \oplus((\mathbf{A}[(x+1) \bmod 5, y, z] \oplus 1) \cdot \mathbf{A}[(x+2) \bmod 5, y, z])
\end{aligned}
$$

$$
\left(\mathbf{A}^{\prime}\right)
$$

## 6. Cryptographic Hash Functions

6.4 Exemplary Hash Functions - Keccak/SHA-3

- Step mapping $\iota$ (iota) adds modulo 2 a round-dependent constant $R C\left[i_{r}\right]$ to lane[0,0] and leaves all other 24 lanes unchanged
■ The round constants $R C\left[i_{r}\right]$ (for $i_{r}=0, \ldots, 23$ ) are constructed in a particular way (not addressed here)

$$
\mathbf{A}^{\prime}[0,0, \cdot]=\mathbf{A}[0,0, \cdot] \oplus \mathrm{RC}\left[i_{r}\right]
$$

## 6. Cryptographic Hash Functions

### 6.4 Exemplary Hash Functions - Keccak/SHA-3

## Table 6.9 (new) <br> The 24 Round Constants $R C\left[i_{r}\right]$ Employed by SHA-3

| $R C[0]$ | $0 \times 0000000000000001$ | $R C[12]$ | $0 \times 000000008000808 \mathrm{~B}$ |
| :--- | :--- | :--- | :--- |
| $R C[1]$ | $0 \times 0000000000008082$ | $R C[13]$ | $0 \times 800000000000008 \mathrm{~B}$ |
| $R C[2]$ | $0 \times 800000000000808 \mathrm{~A}$ | $R C[14]$ | $0 \times 8000000000008089$ |
| $R C[3]$ | $0 \times 8000000080008000$ | $R C[15]$ | $0 \times 8000000000008003$ |
| $R C[4]$ | $0 \times 000000000000808 B$ | $R C[16]$ | $0 \times 8000000000008002$ |
| $R C[5]$ | $0 \times 0000000080000001$ | $R C[17]$ | $0 \times 8000000000000080$ |
| $R C[6]$ | $0 \times 8000000080008081$ | $R C[18]$ | $0 \times 000000000000800 \mathrm{~A}$ |
| $R C[7]$ | $0 \times 8000000000008009$ | $R C[19]$ | $0 \times 800000008000000 \mathrm{~A}$ |
| $R C[8]$ | $0 \times 000000000000008 \mathrm{~A}$ | $R C[20]$ | $0 \times 8000000080008081$ |
| $R C[9]$ | $0 \times 0000000000000088$ | $R C[21]$ | $0 \times 8000000000008080$ |
| $R C[10]$ | $0 \times 0000000080008009$ | $R C[22]$ | $0 \times 0000000080000001$ |
| $R C[11]$ | $0 \times 000000008000000 \mathrm{~A}$ | $R C[23]$ | $0 \times 8000000080008008$ |

## 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

$$
\operatorname{Rnd}\left(\mathbf{A}, i_{r}\right)=\iota\left(\chi(\pi(\rho(\theta(\mathbf{A})))), i_{r}\right)
$$

- The KECCAK- $p\left[b, n_{r}\right]$ permutation consists of $n_{r}$ iterations of Rnd:

$$
\begin{aligned}
& \left(S, n_{r}\right) \\
& \text { convert } S \text { into state } \mathbf{A} \\
& \text { for } i_{r}=2 I+12-n_{r}, \ldots, 2 I+12-1 \text { do } \mathbf{A}=\operatorname{Rnd}\left(\mathbf{A}, i_{r}\right) \\
& \text { convert } \mathbf{A} \text { into } b \text {-bit string } S^{\prime} \\
& \hline\left(S^{\prime}\right)
\end{aligned}
$$

## 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 $n_{r}=12+12 /$ :

$$
\operatorname{KECCAK}-f[b]=\text { KECCAK }-p[b, 12+2 /]
$$

- 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. Cryptographic Hash Functions

■ Most cryptographic hash functions in use today follow the Merkle-Damgård construction and are iterated

- Consequences
- Since each iteration can only start if the preceding iteration has finished, the hash function may become a performance bottleneck
- The design of compression functions that are collision-resistant is still more of an art than a science (i.e., it lacks theoretical foundations)
- Against this background, people come up with ad hoc designs


## 6. Cryptographic Hash Functions

■ Sometimes, people try to improve collision resistance by concatenating two (or more) hash functions

- For example, instead of using MD5 or SHA-1 alone, they may apply one function after the other (e.g., SSL 3.0)
- Intuition suggests that the resulting (concatenated) hash function is more collision-resistant than each function applied individually
- In 2004, it was shown that intuition is illusive and wrong
- Since then, concatenating different hash funstions is no longer used in the field


## 6. Cryptographic Hash Functions

- 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 $x \neq y$

$$
\underset{\substack{\operatorname{Pr} \\ h \leftarrow H}}{\underset{\leftarrow}{\leftarrow}}[h(x)=h(y)] \leq \frac{1}{|Y|}
$$

## 6. Cryptographic Hash Functions

■ 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



## Thank you for your attention




[^0]:    Cryptography 101: From Theory to Practice

