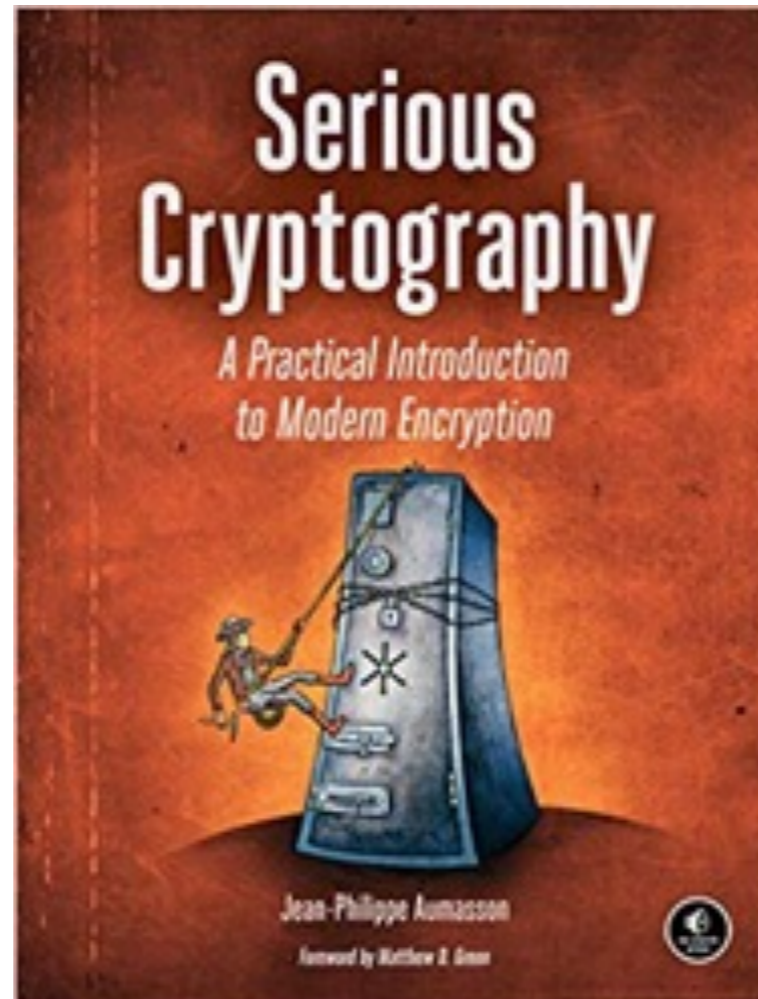


CNIT 141

Cryptography for Computer Networks



6. Hash Functions

Updated 10-2-2023

Topics

- Secure Hash Functions
- Building Hash Functions
- The SHA Family
- BLAKE2
- How Things Can Go Wrong

Use Cases of Hash Functions

- Digital signatures
- Public-key encryption
- Message authentication
- Integrity verification
- Password storage

Fingerprinting

- Hash function creates a fixed-length "fingerprint"
 - From input of any length
 - Not encryption
 - No key; cannot be reversed

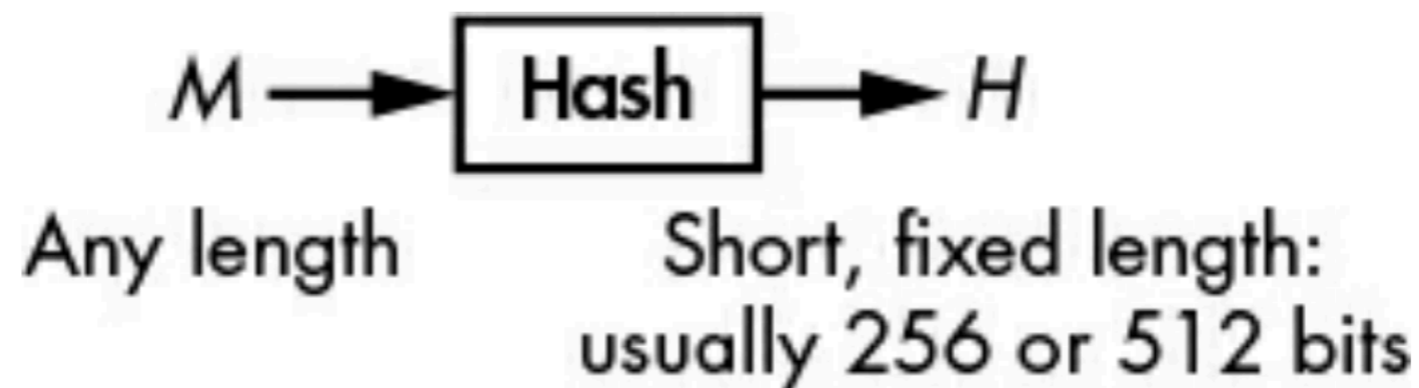


Figure 6-1: A hash function's input and output

Non-Cryptographic Hash Functions

- Provide no security; easily fooled by crafted input
- Used in hash tables
 - To make table lookups faster
- Also used to detect accidental errors
 - Ex: CRC (Cyclic Redundancy Check)
 - Used at layer 2 by Ethernet and Wi-Fi

Secure Hash Functions

Digital Signature

- Most common use case for hashing
- Message **M** hashed, then the hash is signed
- Hash is smaller so the signature computation is faster
- But the hash function must be secure

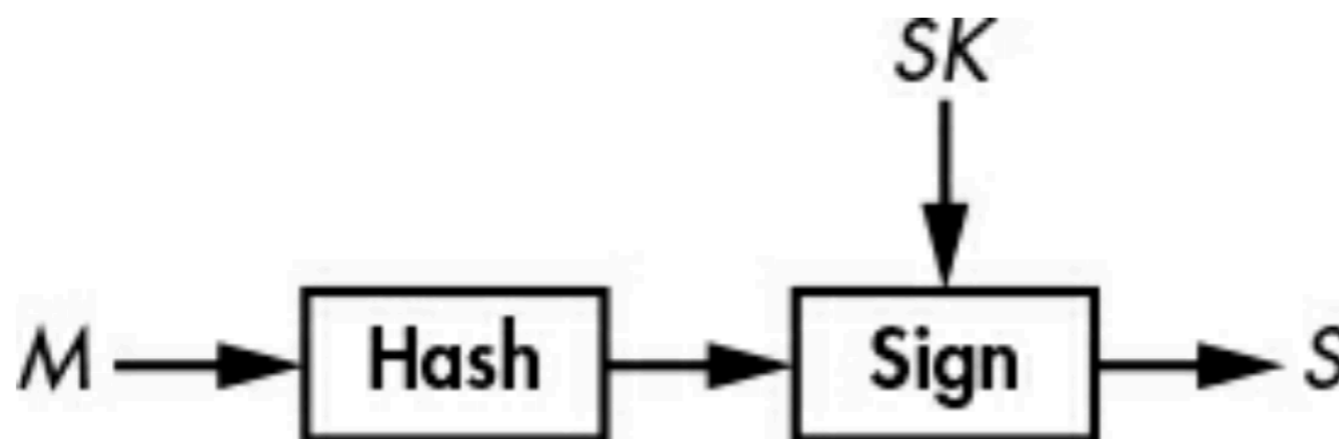


Figure 6-2: A hash function in a digital signature scheme. The hash acts as a proxy for the message.

Unpredictability

- Two different inputs must always have different hashes
- Changing one bit in the input makes the hash totally different

```
SHA-256("a") = 87428fc522803d31065e7bce3cf03fe475096631e5e07bbd7a0fde60c4cf25c7  
SHA-256("b") = a63d8014dba891345b30174df2b2a57efbb65b4f9f09b98f245d1b3192277ece  
SHA-256("c") = edeaaff3f1774ad2888673770c6d64097e391bc362d7d6fb34982ddf0efd18cb
```


Non-Uniqueness

- Consider an input message 1024 bits long
- And a hash 256 bits long
 - There are 2^{1024} different messages
 - But only are 2^{256} different hashes
- So many different messages have the same hash



One-Wayness

- Hashing cannot be reversed
 - An attacker cannot find **M** from **H**
- In principle, always true
 - Many different messages have the same hash



Preimage Resistance

- An attacker given **H** cannot find any **M** with that hash
- This is *preimage resistance*
 - Also called *first preimage resistance*



The Cost of Preimages

- Attacker hashes many messages
- Hunting for a match
- Will take 2^N guesses for hash of length **N** bits
- For $n = 256$, that's forever

```
find-preimage(H) {  
    repeat {  
        M = random_message()  
        if Hash(M) == H then return M  
    }  
}
```

Second Preimage

- Given M_1 and H_1
- Attacker tries to find another message M_2
 - That hashes to the same H
- Essentially the same as first preimage resistance
 - Unless the hash function has a certain type of mathematical defect
 - Like length extension attacks

Collision Resistance

- Every hash function has **collisions**
 - Two messages with the same hash
- Because there are more possible messages than hashes
- ***Pigeonhole principle***
 - If you have 10 holes and 11 pigeons, at least one hole contains more than one pigeon

Collision Resistance

- Collisions should be hard to find
 - In practice, impossible

Birthday Attack

- Get each person in a group to say the month and day they were born
- With 23 people, the probability of two having the same birthday is 51%

$$P(A') = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \frac{362}{365} \times \dots \times \frac{343}{365}$$

The terms of equation (1) can be collected to arrive at:

$$P(A') = \left(\frac{1}{365}\right)^{23} \times (365 \times 364 \times 363 \times \dots \times 343)$$

Evaluating equation (2) gives $P(A') \approx 0.492703$

Therefore, $P(A) \approx 1 - 0.492703 = 0.507297$ (50.7297%).

Finding Collisions

- If you search for collisions
 - And have enough RAM to remember each hash you calculate
- It only takes $2^{N/2}$ hashes to find a collision
 - **N** is the number of bits in the hash

Naive Birthday Attack

1. Compute $2^{n/2}$ hashes of $2^{n/2}$ arbitrarily chosen messages and store all the message/hash pairs in a list.
 2. Sort the list with respect to the hash value to move any identical hash values next to each other.
 3. Search the sorted list to find two consecutive entries with the same hash value.
- Requires $2^{N/2}$ calculations to create the list
 - And $2^{N/2}$ units of storage to hold it
 - Sorting the list takes even more calculations

MD5 Example (Not in book)

Preimage Attack

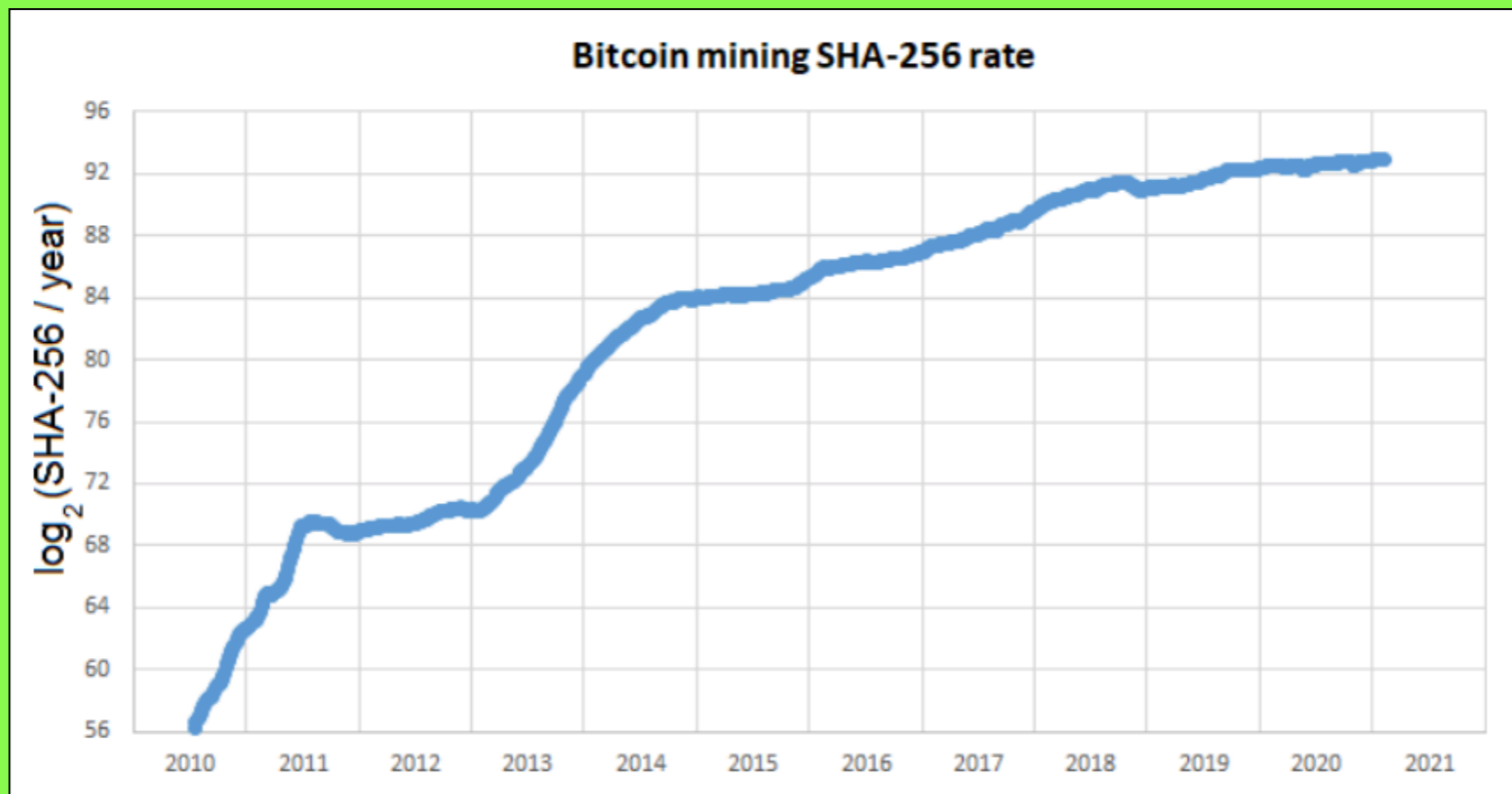
- MD5 is 128 bits long
- Given an MD5 hash, can we find an input with that hash value?
- Calculating every possible hash (nearly) would require 2^{128} calculations
- But we don't need to store them: we just compare each one with the target
- How hard is that?

72 Bit Brute-Force

- RSA Security made several challenges to test brute-force attacks (on the RC5 symmetric cipher)
- *Distributed.net* has been attacking these challenges since 1997
 - Cracked a 56-bit key in 250 days in 1997
 - Cracked a 64-bit key in 5 years in 2002
 - 72-bit key attack is in progress
 - So far they've tested 5.6% of the keyspace
 - It will take 127 years to test all keys

Is 80 bits of key size considered safe against brute force attacks?

- Not really--the Bitcoin hash rate is 2^{93} hashes per year
- So apparently a 93-bit key could be brute-forced



- [Link Ch 6g](#)

MD5 Preimage Attack Safety Margin

- Max. computing available to any attacker is 2^{96} calculations in a year
- Preimage attack requires 2^{128} calculations
- Safety margin is 2^{30} units time (calculations)
 - 2^{10} is 1024
 - 2^{30} is $1024 \times 1024 \times 1024 = 1$ billion years

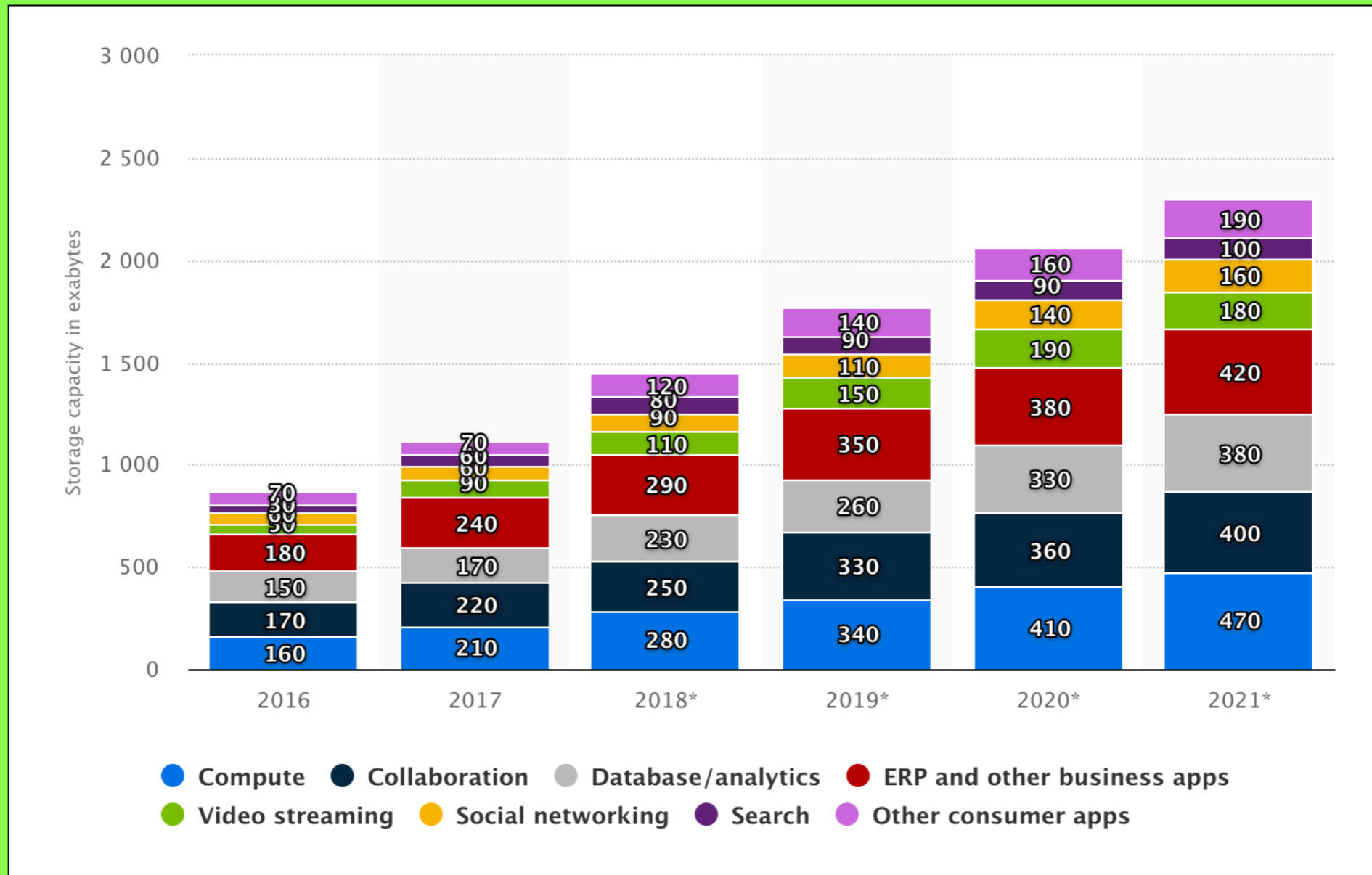
Naive Birthday Attack on MD5

- Take 2^{64} inputs, such as counting numbers from 1 to 2^{64}
- Calculate the hashes, requiring 2^{64} calculations
- Store them, using 2^{64} units of storage
- What's the safety margin?

Requirements for Naive Birthday Attack on MD5

- 2^{64} calculations is not impossible
- But 2^{64} bytes of storage?
 - $1\text{GB} = 10^9 = 2^{30}$
 - $1\text{TB} = 2^{40}$
 - $1\text{ million TB} = 2^{60}$
- 2^{64} bytes = 16 million TB = 16,000 petabytes
= 16 exabytes
- The world's storage is about 2500 exabytes
(see next slide)

Worldwide Data Center Storage Capacity



Requirements for Naive Birthday Attack on MD5

- If a large portion of all the world's processing power
- And a large portion of all the world's storage
- Were used, it's possible
- But that just finds a collision

Preimage Attack

- Given an MD5 hash, can we find an input with that hash value?
- MD5 is 128 bits long
- Calculating every possible hash (nearly) would require 2^{128} calculations
- Storing them would take 2^{128} units of RAM
- Totally out of reach

Low-Memory Collision Search: The Rho Method

- Don't just choose sequential inputs to hash
- Calculate the hash of the previous hash
- Start with any random seed \mathbf{s}
 - $\mathbf{H}_1 = \text{hash}(\mathbf{s})$
 - $\mathbf{H}_2 = \text{hash}(\mathbf{H}_1)$
 - $\mathbf{H}_3 = \text{hash}(\mathbf{H}_2)$
 - etc...

Rho Method

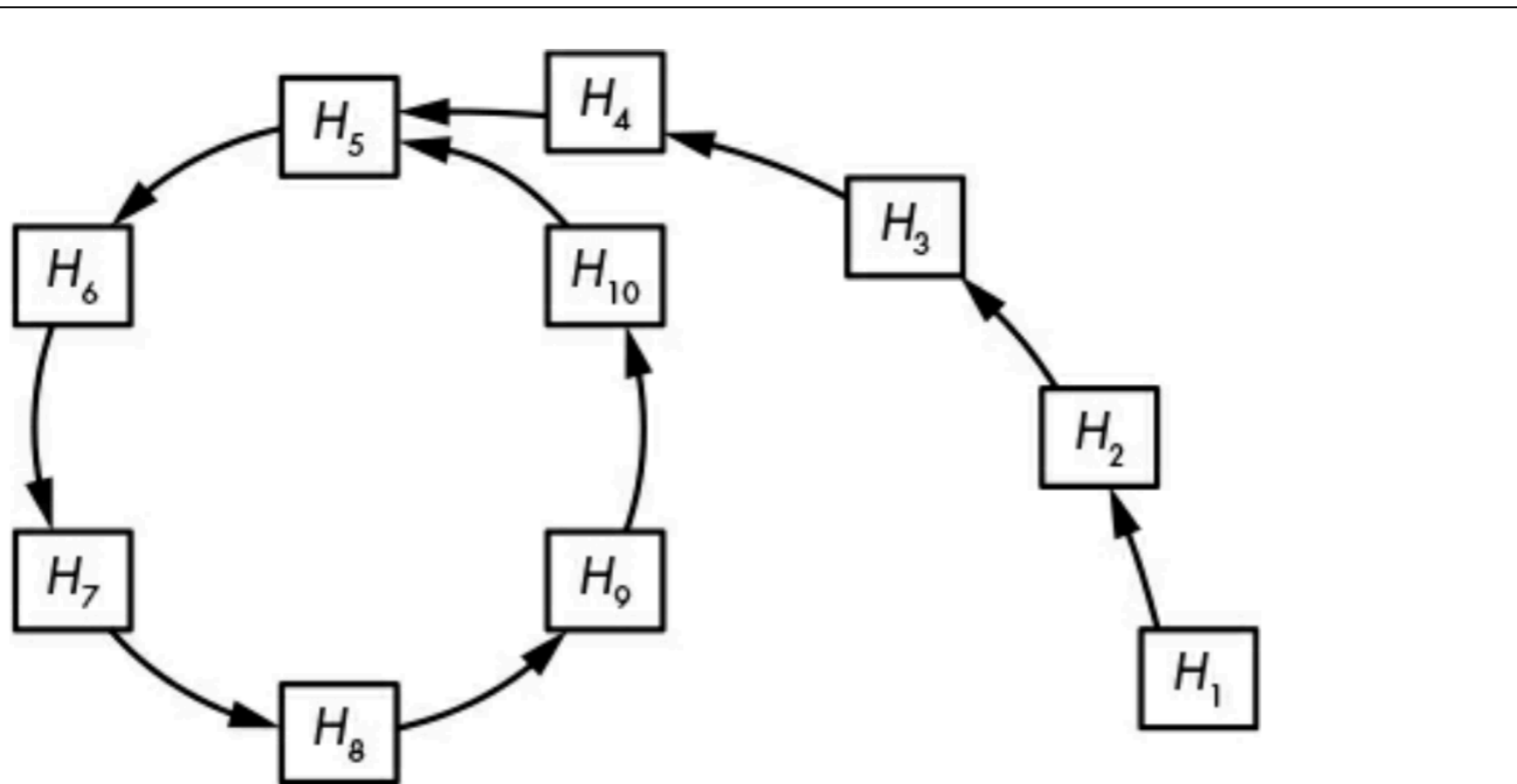


Figure 6-3: The structure of the Rho hash function. Each arrow represents an evaluation of the hash function. The cycle beginning at H_5 corresponds to a collision, **Hash**(H_4) = **Hash**(H_{10}) = H_5 .

Rho Method

- Tail and circle are approximately $2^{N/2}$ long
- Since the hashes are going in a circle
 - You don't need to store all the hash values in memory to detect a collision
 - You store only hashes starting with many zeroes ("Distinguished Points")
- This requires more calculations but less storage -- ***time-memory trade-off***

Rho Method

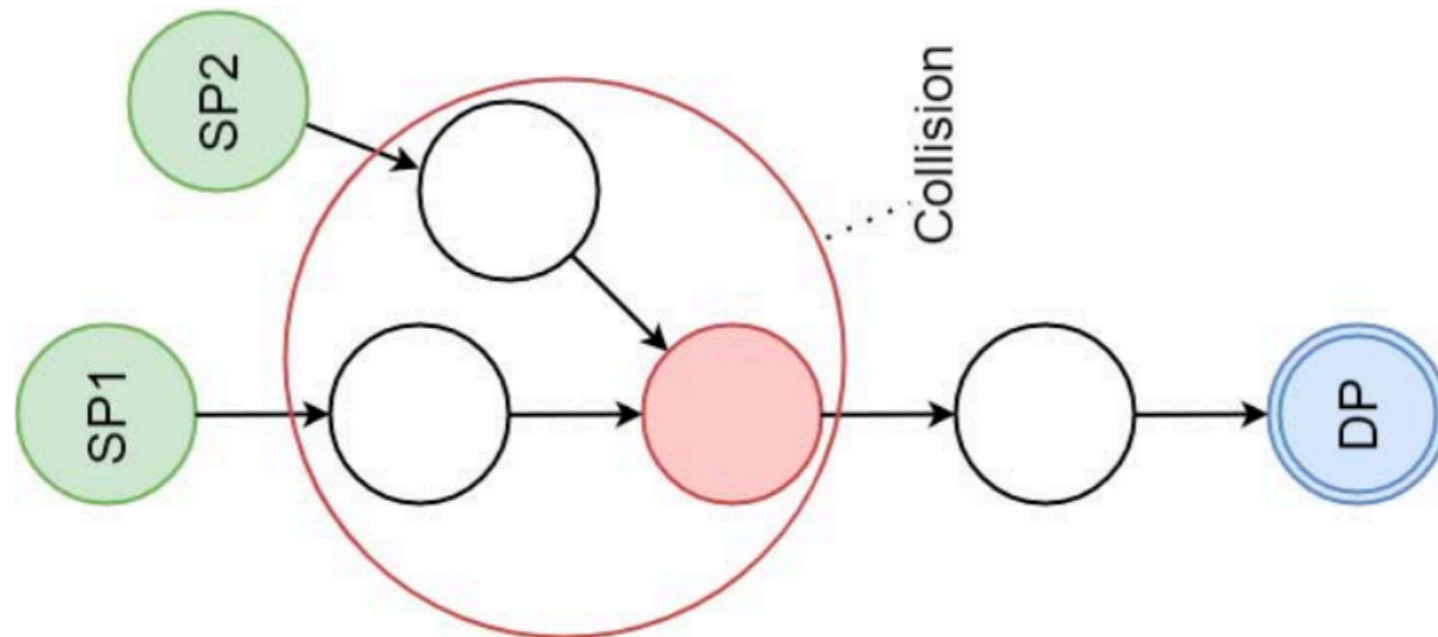


Figure 2: DPs lead to collision diagram (SP: starting point, DP: distinguished point)

- Link Ch 6k

Rho Method for 16-bit Hash

```
>>> import hashlib
>>> h = "a"
>>>
>>> for i in range(50):
...     h = hashlib.new('md5', h).hexdigest(
...     )[-2:]
...     print h,
...
61 91 94 ef 43 8b 7b d4 31 f5 dc 8f e0 87
6d 8a 19 c4 7a ee 1e 21 79 d7 98 ca d3 59
a2 c0 a8 a5 db 1b 1e 21 79 d7 98 ca d3 59
a2 c0 a8 a5 db 1b 1e 21
```

Kahoot!

6a

Building Hash Functions

Iterative Hashing

- A long messages is broken into blocks
- Each block is processed consecutively using ***compression*** or ***permutation***

Compression-Based Hash Functions: the Merkle-Damgård Construction

- Used in MD4, MD5, SHA-1, and SHA-2
 - Also RIPEMD and Whirlpool
- H_0 is an initial value (IV)
- M_1, M_2, \dots are blocks of message data
- The final H is the output

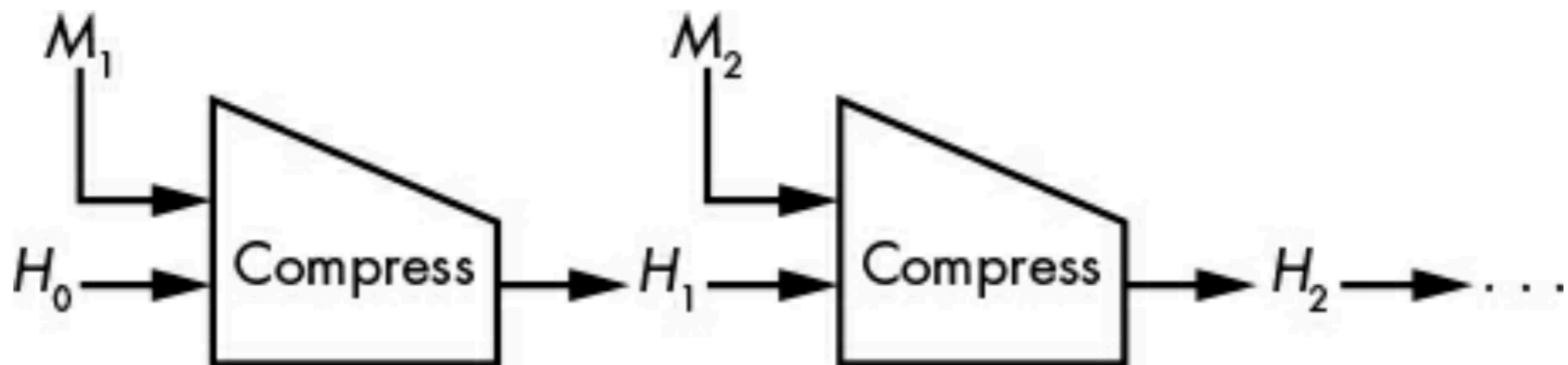


Figure 6-4: The Merkle-Damgård construction using a compression function called Compress

Padding the Last Block

- Pad with 1, then zeroes, finally the message length
- SHA-256 uses 512-bit blocks
- To pad the eight-bit message **10101010**

10101010100000000000000000000000()000000000000001000

Building Compression Functions: The Davies-Meyer Construction

- Uses a block cipher to build a compression function
- Use message blocks as keys
- The XOR feedback makes it secure against decryption

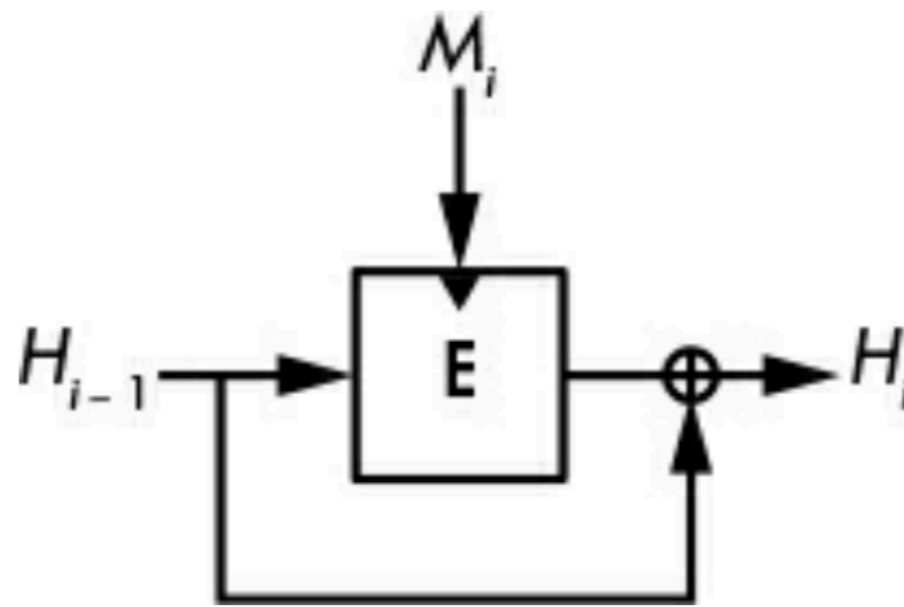


Figure 6-5: The Davies-Meyer construction. $\bar{1}$

Other Compression Functions

- Less popular because
 - They are more complex, or
 - Require message block to be same length as chaining value H_i

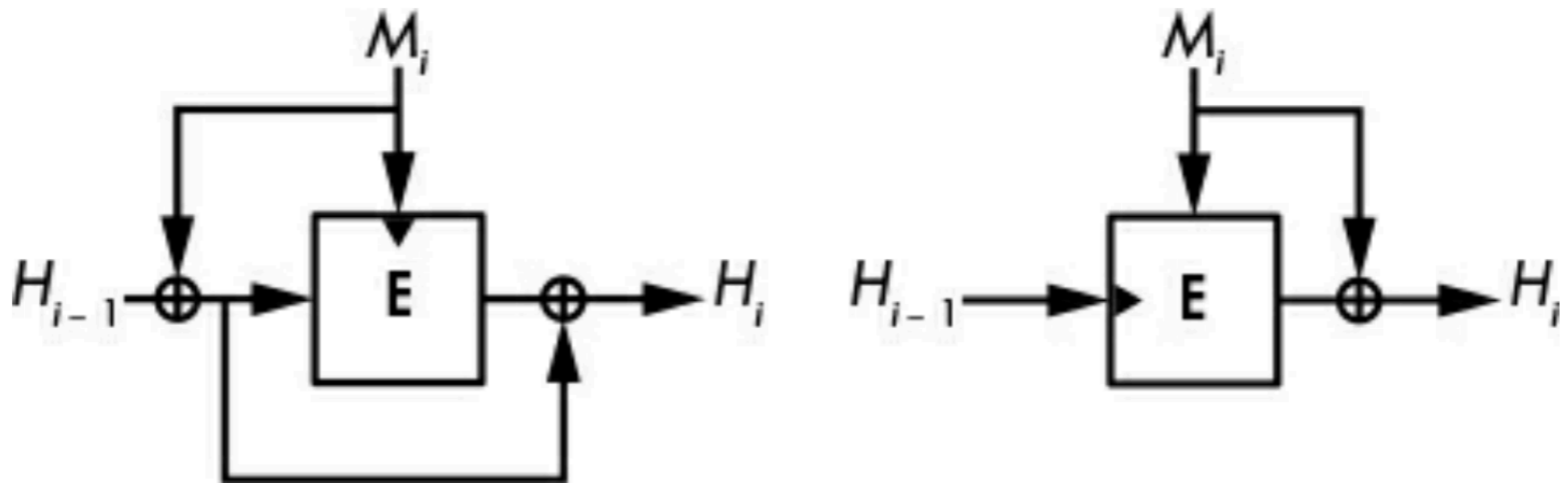
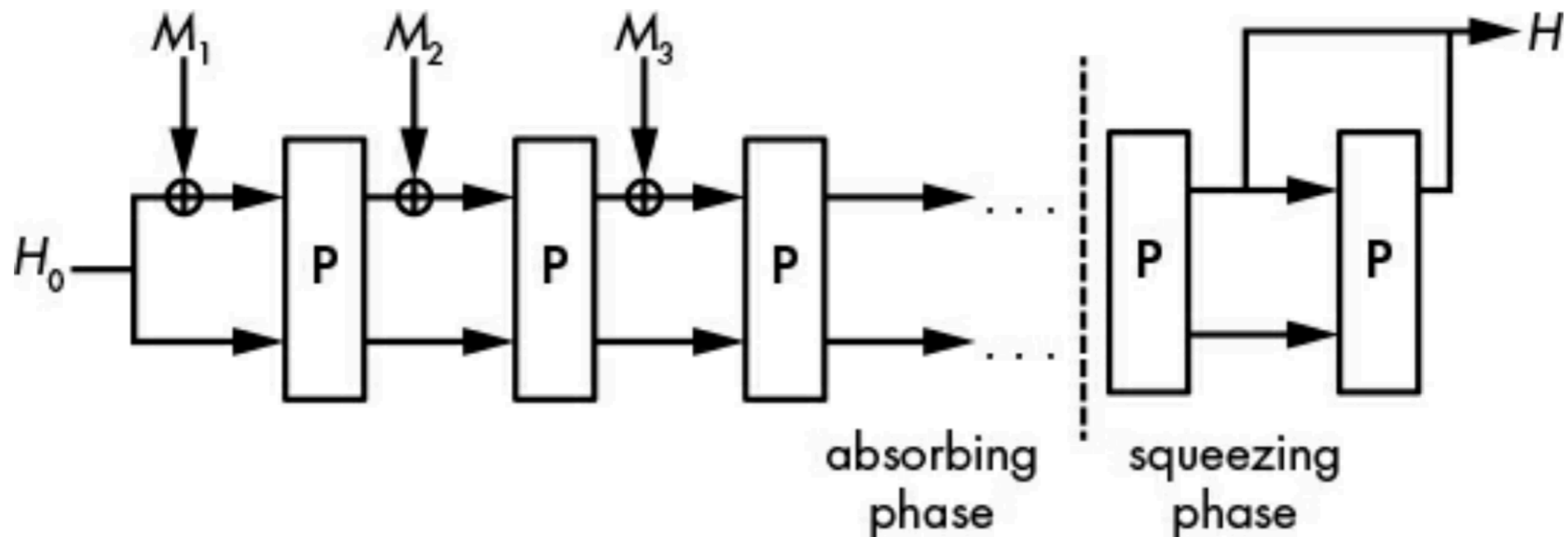


Figure 6-6: Other secure block cipher-based compression function constructions

Permutation-Based Hash Functions: Sponge Functions

- Simpler than using a cipher
- No key
- Keccak is the most famous sponge function
 - Also known as SHA-3

1. It XORs the first message block, M_1 , to H_0 , a predefined initial value of the internal state (for example, the all-zero string). Message blocks are all the same size and smaller than the internal state.
2. A permutation, P , transforms the internal state to another value of the same size.
3. It XORs block M_2 and applies P again, and then repeats this for the message blocks M_3, M_4 , and so on. This is called the *absorbing phase*.
4. After injecting all the message blocks, it applies P again and extracts a block of bits from the state to form the hash. (If you need a longer hash, apply P again and extract a block.) This is called the *squeezing phase*.



The SHA Family

MD5

- Broken in 2005
- Fast attacks now to generate MD5 collisions
- It still cannot be reversed in general

SHA-0

- Approved by NIST in 1993
- Replaced in 1995 by SHA-1
 - To fix an unidentified security issue
- 160 bits long, so it should take 2^{80} calculations to find a collision
 - In 1998 an attack was found requiring only 2^{60} calculations to find a SHA-0 collision
 - Later attacks work in 2^{33} calculations--less than an hour of computing (for SHA-0)

SHA-1

- Uses an encryption function called SHACAL
- Repeats this calculation for every 512-bit block in the message **M**

$$**H = E(M, H) + H**$$

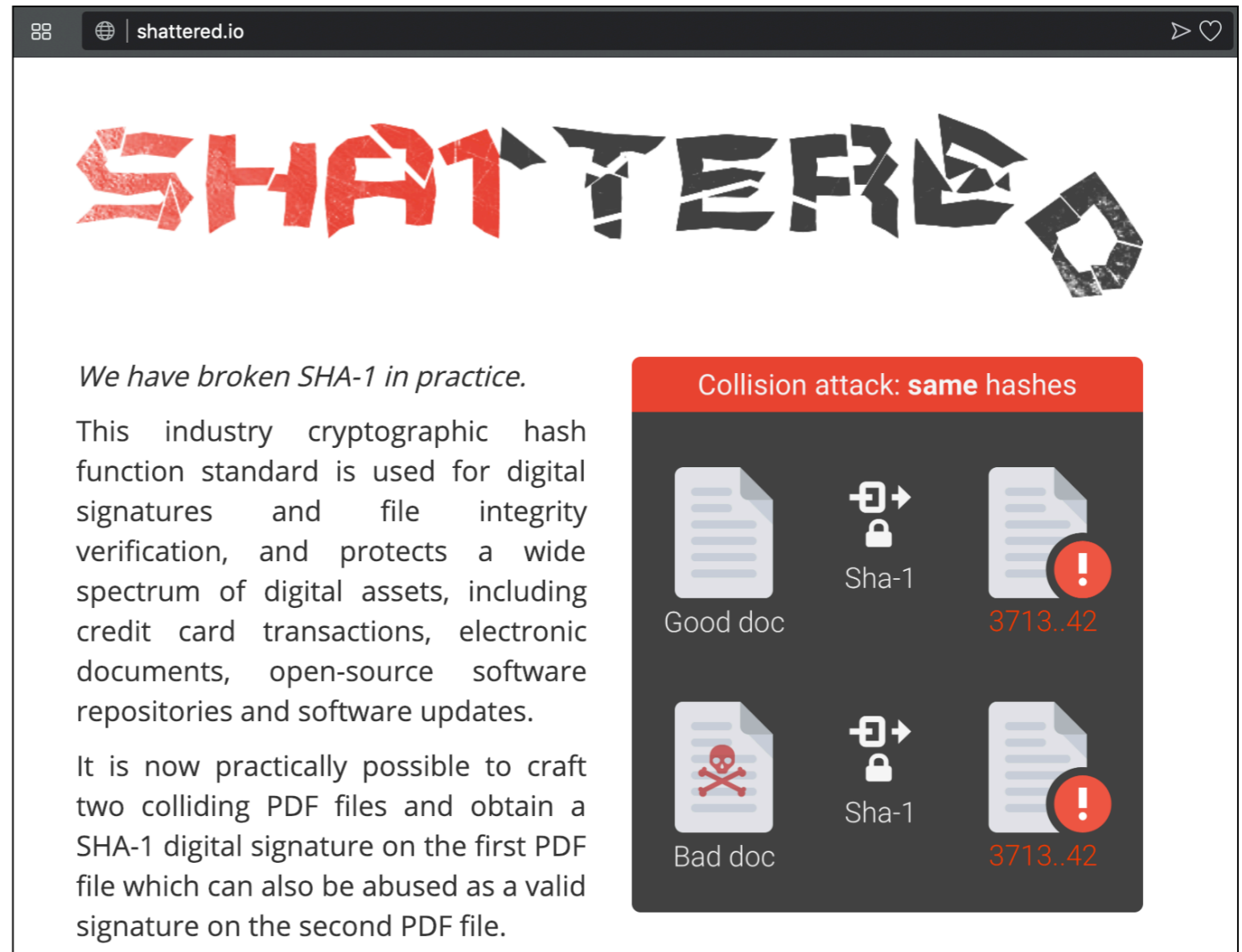
Compression Function

- 160-bit chaining value broken into 5 32-bit words **a b c d e**

```
SHA1-compress(H, M) {  
    (a0, b0, c0, d0, e0) = H // parsing H as five 32-bit big endian words  
    (a, b, c, d, e) = SHA1-blockcipher(a0, b0, c0, d0, e0, M)  
    return (a + a0, b + b0, c + c0, d + d0, e + e0)  
}
```

Attacks on SHA-1

- In 2005 an attack was found that would find a collision in 2^{63} calculations instead of 2^{80}
- In 2017 a SHA-1 collision was found
- No longer trusted



The screenshot shows a web browser window with the URL "shattered.io". The main heading is "SHATTERED" in a stylized, fragmented font. Below the heading, the text reads: "We have broken SHA-1 in practice. This industry cryptographic hash function standard is used for digital signatures and file integrity verification, and protects a wide spectrum of digital assets, including credit card transactions, electronic documents, open-source software repositories and software updates. It is now practically possible to craft two colliding PDF files and obtain a SHA-1 digital signature on the first PDF file which can also be abused as a valid signature on the second PDF file."

To the right of the text is a graphic titled "Collision attack: same hashes". It shows two rows of document icons. The top row is labeled "Good doc" and shows a document icon with a lock icon and the text "Sha-1" and "3713..42". The bottom row is labeled "Bad doc" and shows a document icon with a skull and crossbones, a lock icon, and the text "Sha-1" and "3713..42". Both document icons in the bottom row have a red exclamation mark in a circle, indicating a collision.

SHA-2

- Four versions
 - SHA-224, SHA-256, SHA-384, SHA-512
- SHA-256 uses
 - Eight 32-bit words
 - 64 rounds of calculation with a more complex expand function

Security of SHA-2

- All four SHA-2 algorithms are still considered strong
- But researchers and NIST grew concerned because its algorithm is close to SHA-1's

SHA-3 Finalists

- BLAKE
- Grøstl
- JH
- Keccak
- Skein

Keccak (SHA-3)

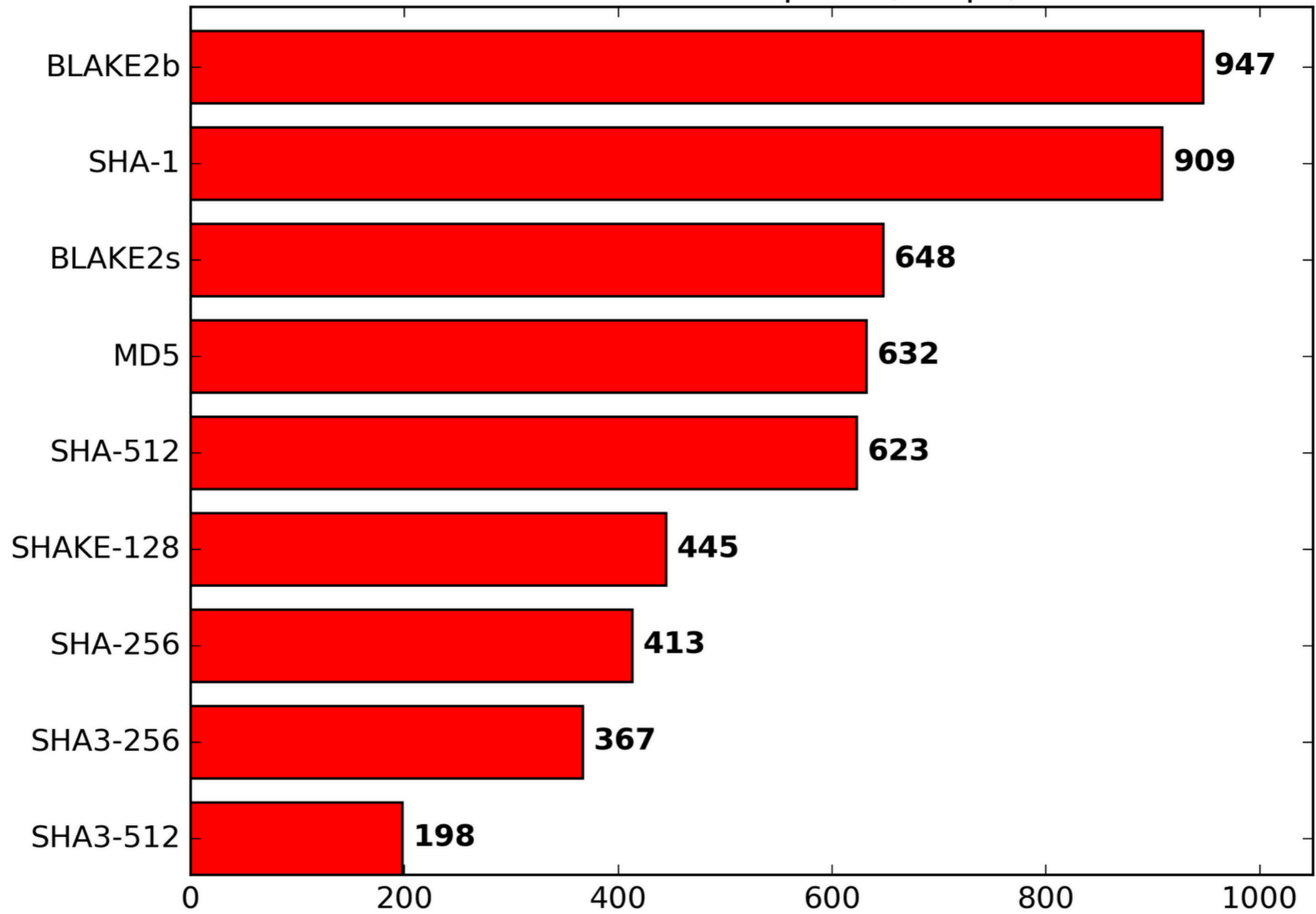
- Completely different from SHA-1 and SHA-2
- Sponge function with a 1600-bit state
- Includes four hashes:
 - SHA3-224, SHA3-256, SHA3-384, SHA3-512
- And two ***extendable-output functions***
 - Can produce hashes of any length
 - Called SHAKE128 and SHAKE256

BLAKE2

Speed

- BLAKE2 is as secure as SHA-3
- But much faster to compute
 - Faster than MD5 or SHA-1
- Two functions
 - BLAKE2 (also called BLAKE2b) optimized for 64-bit platforms, produces digests from 1 to 64 bytes
 - BLAKE2s optimized for 8-bit to 32-bit platforms, produces digests from 1 to 32 bytes

Hash functions speed (MiBps)



Usage

- BLAKE2 is the fastest secure hash available today
- The most popular non-NIST-standard hash
- Used in many apps and in major libraries such as openSSL and Sodium

BLAKE's Compression Function

- Parameters: a counter and a flag
- Block cipher is based on ChaCha
- Which is based on Salsa20

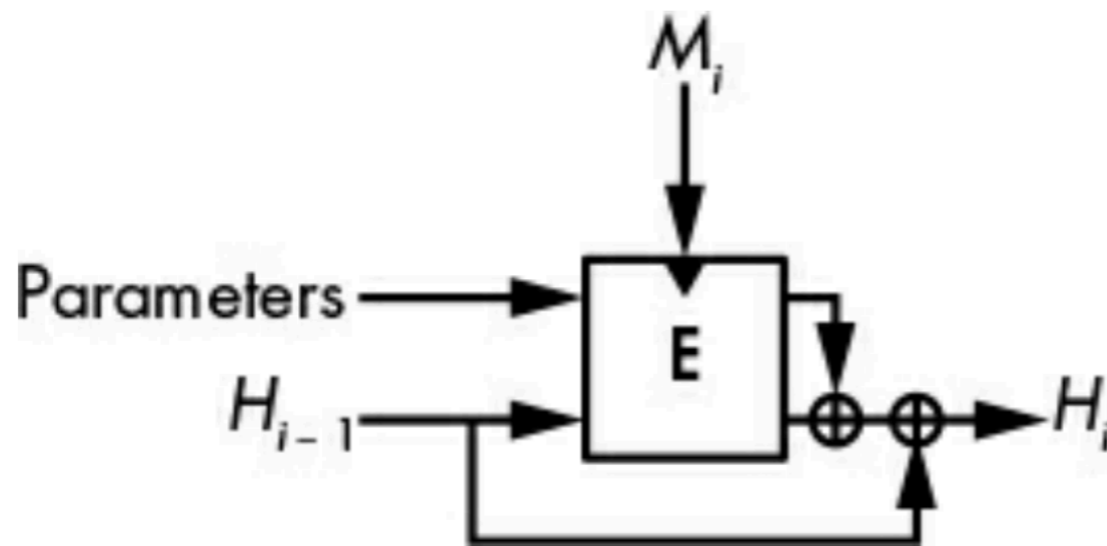


Figure 6-8: BLAKE2's compression function. The two halves of the state are XORed together after the block cipher.

BLAKE2b's Core Operations

- Transforms the state of four 64-bit words
 - **a b c d**
- Using two message words
 - **M_i M_j**

$$a = a + b + M_i$$

$$d = ((d \oplus a) \ggg 32)$$

$$c = c + d$$

$$b = ((b \oplus c) \ggg 24)$$

$$a = a + b + M_j$$

$$d = ((d \oplus a) \ggg 16)$$

$$c = c + d$$

$$b = ((b \oplus c) \ggg 63)$$

How Things Can Go Wrong

Misuse

- Using weak checksum algorithms like CRC32 for file integrity
- Instead of a cryptographic hash algorithm

The Length-Extension Attack

- If you know the hash $H(M)$
- You can add more data to the right and calculate the hash of the longer message
- Without knowing the original message

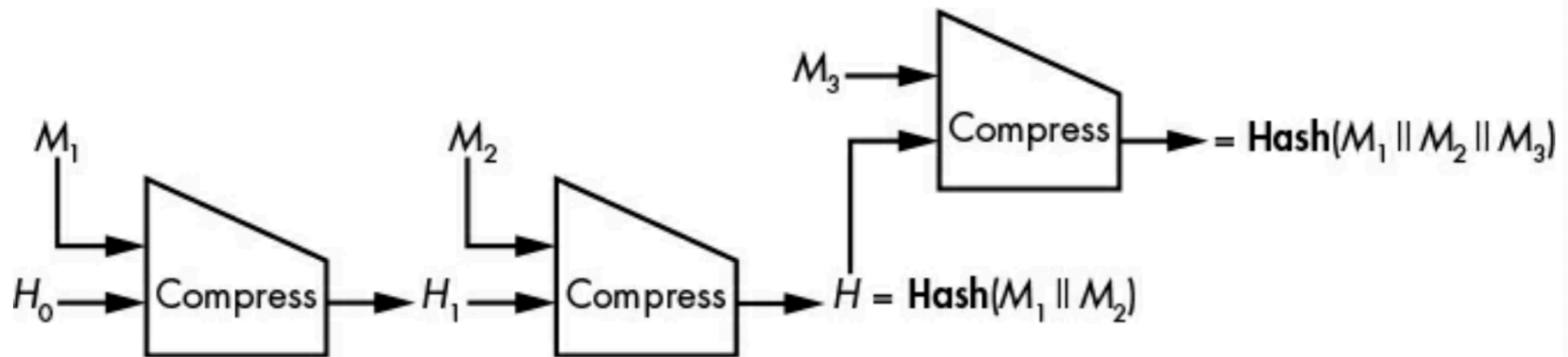


Figure 6-9: The length-extension attack

Effects

- Attacker can use $H(M_1 || M_2)$ to generate
 - $H(M_1 || M_2 || M_3)$
- The effect depends on how applications use hashes
 - But some applications do get fooled

SHA-2 Length Extension

- SHA-2 is vulnerable
 - Could have easily been avoided by making the last compression function different
 - BLAKE2 does that

Fooling Proof-of-Storage Protocols

- A cloud server verifies that it has stored a message **M**
 1. The client picks a random value, C , as a *challenge*.
 2. The server computes $\text{Hash}(M \parallel C)$ as a *response* and sends the result to the client.
 3. The client also computes $\text{Hash}(M \parallel C)$ and checks that it matches the value received from the server.

Fooling Proof-of-Storage Protocols

- Server can cheat by keeping only the chain value from hashing the message
 - And discarding the message
- Server can still calculate the response by *length extension*
- Works for SHA-1, SHA-2, SHA-3, and BLAKE
- Cure: hash **C||M** instead of **M||C**

Kahoot!

6b