

# Cryptography

## Hash Functions

Prof. Dr. Heiko Knospe

TH Köln – University of Applied Sciences

May 10, 2020

# Hash Functions

In general, a *cryptographic hash function* consists of a polynomial-time key generator (that takes a security parameter  $1^n$  as input) and a hash algorithm. A *keyed* hash function

$$H_k : \{0, 1\}^* \rightarrow \{0, 1\}^{l(n)}$$

takes a key and a binary string as input and outputs a hash value of length  $l(n)$ .

If the input length is restricted to  $l'(n) > l(n)$ , then  $H_k$  is called a *compression function*.

# Collisions and Collision Resistance

Since hash values are used as *message digests* or unique *identifiers*, their main requirement is *collision resistance*. A collision is given by two input values  $x \neq x'$  with

$$H(x) = H(x').$$

## Definition

A function  $H = H_k : D \rightarrow R$ , where  $H$ ,  $k$ ,  $D$  and  $R$  depend on a security parameter  $n$ , is called *collision resistant* if the probability that a probabilistic polynomial-time adversary finds a collision  $H(x) = H(x')$ , where  $x, x' \in D$  and  $x \neq x'$ , is negligible in  $n$ .

# Weak Collision Resistance

There are two related requirements, which are weaker than collision resistance:

- *Second-preimage resistance* or *weak collision resistance* means that an adversary, who is given a uniform  $x \in D$ , is not able to find a second preimage  $x' \in D$  with  $x \neq x'$  such that  $H(x) = H(x')$ .
- *Preimage resistance* or *one-wayness* means that an adversary, who is given a uniform  $y \in R$ , is not able to find a preimage  $x \in D$  such that  $H(x) = y$ .

# Unkeyed Hash Functions and the Random Oracle Model

In practice, hash functions are *unkeyed* or the key is fixed. Unkeyed hash functions

$$H : \{0, 1\}^* \rightarrow \{0, 1\}^l$$

have a theoretical disadvantage: they are fixed functions and a collision can be found in constant time.

An ideal unkeyed hash function is called a *random oracle*. The output of a random oracle is uniformly random, unless the same input is queried twice, in which case the oracle returns the same output.

The random oracle model is used in some security proofs. However, concrete implementations of a random oracle are impossible.

# Birthday Paradox

Hash values should not be too short. In fact, the *Birthday Paradox* demonstrates that collisions occur surprisingly often:

## Theorem

*Let  $k$  be the number of independent samples drawn from a uniform distribution on a set of size  $N$ . If  $k \approx 1.2\sqrt{N}$ , then the probability of a collision is around 50%.*

If we consider hash values of length  $l$  and assume that they have a uniform distribution, then collisions occur after hashing around  $\sqrt{2^l} = 2^{l/2}$  messages.

In order to minimize the risk of random collisions, hash values should be at least 200 bits long.

# Integrity Protection and Applications of Hash Functions

Firstly, the hash value can be used as a short *identifier* of data.

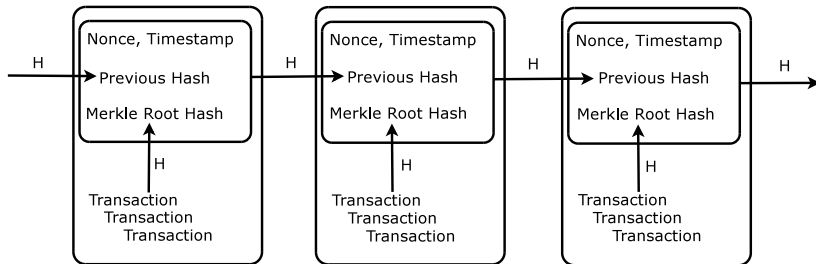
$$m \longrightarrow H(m)$$

The identifier is unique as long as the hash function is collision resistant. Hashes can be used to verify the *integrity* of messages and files. Note that the verifier needs access to the *authentic* message digest  $H(m)$ .

Hashes are also used in the construction of *message authentication codes* (HMAC). *Signature schemes* first hash the message. Furthermore, *pseudorandom functions* can be constructed from hash functions.

# Blockchain

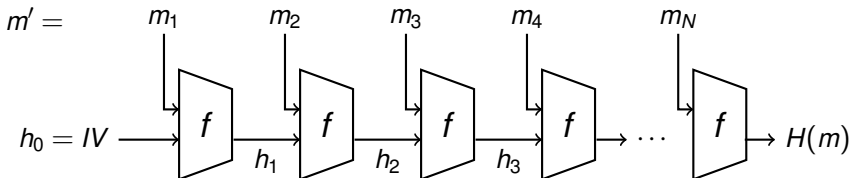
A *blockchain* is a sequence of linked blocks. Each block contains the hash value of the previous block. A blockchain can be used as a *distributed ledger*, which records transactions in an efficient and verifiable way. Hash values protect the integrity of the blockchain: transactions in a block cannot be modified without changing all subsequent hash values.





# Merkle-Damgård Construction

The *Merkle-Damgård construction* has found widespread use, including the MD-SHA family. The Merke-Damgård transform is based on a *compression function*  $f : \{0, 1\}^{n+l} \rightarrow \{0, 1\}^n$ , which maps  $n+l$  input bits to  $n$  output bits. The compression function is applied recursively. A message  $m$  is padded and its length is appended, giving  $m'$ . In each step, one block of  $m'$  is processed and the last output defines the hash value.



# SHA-1

SHA-1 is a Merkle-Damgård hash function based on a compression function

$$f : \{0, 1\}^{160+512} \rightarrow \{0, 1\}^{160}.$$

A 512-bit message block  $m = W_0 \| W_1 \| \dots \| W_{15}$  is subdivided into 16 words of length 32 bits. By XOR operations and a circular left shift by one position, 64 additional words  $W_{16}, \dots, W_{79}$  are generated:

$$W_j = (W_{j-16} \oplus W_{j-14} \oplus W_{j-8} \oplus W_{j-3}) \lll 1 \text{ for } 16 \leq j \leq 79$$

The 160-bit input vector  $h = H_1 \| H_2 \| H_3 \| H_4 \| H_5$  is subdivided into five 32-bit words and copied to the initial status vector:

$$A \| B \| C \| D \| E \leftarrow H_1 \| H_2 \| H_3 \| H_4 \| H_5$$

# SHA-1 Compression Function

Then, 80 rounds of the SHA-1 compression function are performed, which update the status words  $A||B||C||D||E$ . In round  $j$ , the 32-bit message word  $W_j$  is processed. A nonlinear bit-function  $F$  (defined by AND, OR, NOT and XOR operations) and a constant  $K$  are used. The function  $F$  and the constant  $K$  change every 20 rounds. The following function is used for the first 20 rounds:

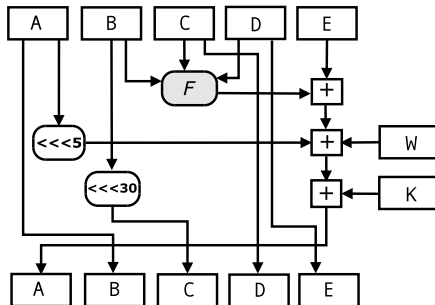
$$F(B, C, D) = (B \wedge C) \oplus (\neg B \wedge D)$$

After completing 80 rounds, the compression function outputs

$$f(h, m) = (A + H_1 \parallel B + H_2 \parallel C + H_3 \parallel D + H_4 \parallel E + H_5),$$

where  $+$  denotes addition modulo  $2^{32}$ .

# SHA1 Compression Function



One round of the SHA-1 compression function  $f$  which updates the 160-bit state.  $W$  is a 32-bit message chunk,  $F$  a bit function and  $K$  a constant.  $\lll$  denotes left-rotations and  $+$  addition modulo  $2^{32}$ .

# SHA-1 Collision

In February 2017, a SHA-1 collision was found. The attack required  $2^{63}$  SHA-1 calls and took approximately 6500 CPU years and 100 GPU years. A prefix  $P$  was chosen and two different 1024-bit messages  $M^{(1)}$  and  $M^{(2)}$  were found such that

$$H(P\|M^{(1)}) = H(P\|M^{(2)}).$$

Since  $P$  is a valid preamble for PDF documents, the collision makes it possible to fabricate two different PDF files with the same SHA-1 hash value and impressive examples have been published.

Here is the prefix  $P$  in ASCII characters:

```
%PDF-1.3.%.....1 0 obj.<</Width 2 0 R/Height 3 0 R/Type 4 0 R/  
Subtype 5 0 R/Filter 6 0 R/ColorSpace 7 0 R/Length 8 0 R/BitsPer  
Component 8>>.stream.....$SHA-1 is dead!!!!!!./...#9u.9...<L.....
```

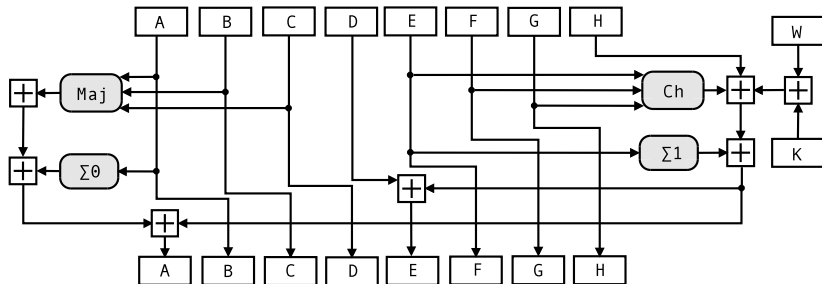
# SHA-2

The SHA-2 hash functions SHA-224, SHA-256, SHA-384 and SHA-512 are constructed in a similar way to SHA-1, but use an extended internal state and larger digests. It is assumed that SHA-2 offers better protection against collision-finding attacks and at the time of writing SHA-2 is widely used in security protocols and applications.

In the following, we describe the SHA-256 variant. The SHA-2 compression function  $f$  takes as input a 256-bit status vector and a 512-bit message block and outputs an updated 256-bit status:

$$f : \{0, 1\}^{256+512} \rightarrow \{0, 1\}^{256}$$

# SHA-2 Compression Function



*One round of the SHA-2 compression function  $f$ . The functions  $Maj$ ,  $Ch$ ,  $\Sigma_0$  and  $\Sigma_1$  are defined in a similar way as the function  $F$  used in SHA-1.*

The 32-bit status words  $A$ ,  $B$ ,  $C$ ,  $D$ ,  $E$ ,  $F$ ,  $G$ ,  $H$  are updated. In each of the 64 rounds, a 32-bit chunk  $W$  derived from the message block is processed.  $K$  is a 32-bit constant that depends on the round number.

# SHA-3

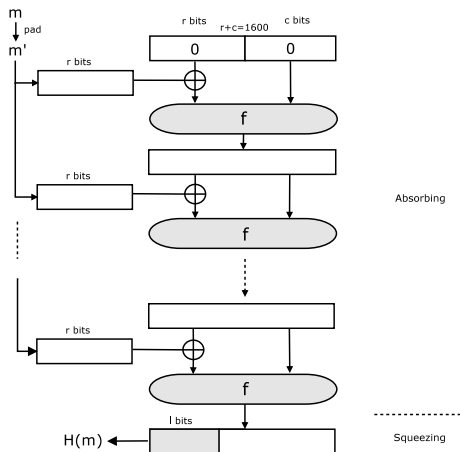
After a competition to construct a new secure hash function called SHA-3, *Keccak* was selected as the winner. Keccak is not of Merkle-Damgård type, but rather based on a *sponge* construction.

In 2015, the Keccak variants SHA3-224, SHA3-256, SHA3-384, SHA3-512 with output lengths between 224 and 512 bits were standardized. The SHA-3 instance of Keccak uses a three-dimensional state array of  $5 \times 5 \times 64 = 1600$  bits. The unkeyed Keccak- $f[1600]$  permutation operates on the 1600-bit state array and it is assumed that  $f$  behaves like a *random permutation*.

$$f : \{0, 1\}^{1600} \rightarrow \{0, 1\}^{1600}$$



# Keccak Operation



*Absorbing message blocks of length  $r$  and squeezing out the hash value.*

# SHA-3 Family

## Definition

The SHA-3 family of hash functions

$$H : \{0,1\}^* \rightarrow \{0,1\}^l$$

supports output lengths  $l \in \{224, 256, 384, 512\}$ . Depending on  $l$ , the rate  $r$  and the capacity  $c$  are fixed such that  $r + c = 1600$ . The input message  $m$  is padded such that the length of  $m'$  is a multiple of  $r$ . The padded message  $m'$  is split into blocks  $m_1, m_2, \dots, m_N$  of length  $r$ :

$$m' = m \parallel 0110 \dots 01 = m_1 \parallel m_2 \parallel \dots \parallel m_N$$

The state  $s = s_1 \parallel s_2$  is initialized by the zero vector  $0^r \parallel 0^c$ .

# SHA-3 Family

## Definition

During the *absorbing* phase, the state is recursively updated by the message blocks:

$$s_1 \parallel s_2 \leftarrow f(s_1 \oplus m_i \parallel s_2) \text{ for } 1 \leq i \leq N$$

Finally, the SHA-3 hash value is computed using a single *squeezing* operation;  $H(m)$  is defined by the leftmost  $l$  bits.

# SHA-3 Family

An advantage of the sponge construction – in comparison to the Merkle-Damgård transform – is that the hash value does *not reveal the full state*, which prevents length extension attacks.

The SHA-3 standard also defines two *extendable-output functions* (XOF) called SHAKE128 and SHAKE256, with which the output can be extended to any desired length. In this case, the Keccak- $f$  function is applied multiple times during the squeezing phase to obtain the required number of output bits.

There are also a keyed hash functions based on Keccak (KMAC128, KMAC256).