Network Security and Cryptography
Symmetric-key cryptography

Lecture 8: Integrity and hash functions

Mark Ryan

Integrity of messages

Goal: Ensure change of message by attacker can be detected
Key tool: *Cryptographic hash function*

Definition

A *cryptographic hash function* is a function from bitstrings of [almost] arbitrary length to bitstring of a small, fixed length (e.g. 160 bits or 256 bits) such that

- For any message or string x, $h(x)$ is easy to compute.
- h is *one-way* (sometimes called *pre-image resistant*): it is hard to invert. For any y it is computationally infeasible to find an x such that $y = h(x)$.
- h is *weakly collision-resistant* (sometimes called *second preimage resistant*): for any x it is computationally infeasible to find an $x' \neq x$ such as $h(x) = h(x')$.
- h is *collision-resistant*: it is computationally infeasible to find x and x' such that $x \neq x'$ and $h(x) = h(x')$.

Example uses of hash functions

1. Verifying the integrity of messages and files. Idea is to confirm the hash by a secure channel (phone? another website?).
2. Signature generation and verification. Almost all digital signature algorithms require taking the hash first.
3. Password verification. To avoid storing plaintext passwords on a web server, store the hashes instead. Even better, to make brute force attacks harder: use *password stretching* techniques like PBKDF2, and use *salted passwords*.
4. Proof-of-work. Hash functions allow us to define tasks requiring measureable amounts of work, as used in the Bitcoin blockchain. The work is generally useless except to show that you have done it.
5. File or data identifier. Several source code management systems, including Git, use the hash of various types of content to name the content uniquely.

Collision-free vs collision-resistant

- A hash function necessarily has loads of collisions. This can be seen just by considering the size of the input message space and the size of the output message space.
  - For SHA-1, the input is a message up to length $2^{64} - 1$ bits.
  - The output is a message of length 160 bits.

  So, on average, each possible output corresponds to a vast number of inputs!

- Collision-resistance means that it is computationally hard to find a single collision. If a single collision is found, the hash function is considered broken.

Collision-resistance

Collision resistance is achieved by making the output look as random as possible, even though it is deterministic. This means that changing just one bit of the input should "completely change" the output. That means that typically half of the output bits are changed.

Example

```
echo "The quick brown fox jumps over the lazy dog" | sha1sum
```

`be417768b5c3c5c1d9bcb2e7c119196dd76b5570`

```
echo "The quick brown fox jumps over the lazy dof" | sha1sum
```

`71451f4d87fe0e866ec8ebc57f5379b23fa2921f`

In binary:

10111110010000010111011101101000101101011100001111000101110...
01110001010001010001111101001101100001111111110000011101000...

Of the 60 bits shown, 27 have changed.

One-way vs collision-resistant

One-way: given y, infeasible to find x such that $h(x) = y$.
Collision-resistant: Infeasible to find distinct x and x' such that $h(x) = h(x')$.

Which one of these properties do you think is easier to break?

One-way vs collision-resistant

One-way: given y, infeasible to find x such that $h(x) = y$.
Collision-resistant: Infeasible to find distinct x and x' such that $h(x) = h(x')$.

Which one of these properties do you think is easier to break?

Answer: collision resistance

“Birthday paradox”

Suppose there are 23 people in the room.

- The chance that someone has **my** birthday is approximately 0.06.
- The chance that **some two** of them have the same birthday is much higher: it is approximately 0.5.

Suppose there are 50 people in the room.
Then these probabilities become 0.13 and 0.96.

One-way vs collision-resistant

One-way: given y, infeasible to find x such that $h(x) = y$.
Collision-resistant: Infeasible to find distinct x and x' such that $h(x) = h(x')$.

The work needed to break collision-resistance is much less than the work needed to break one-way. If it takes 2^n operations to break one-way, then it takes only approximately $2^{n/2}$ operations to break collision resistance.

The "birthday paradox" is one way to understand this distinction.

Moral: a hash function that outputs n bits has at most $n/2$ bits of security.

Example hash functions

MD4 128-bits hash length. Proposed 1990. Collisions found 1995.

MD5 128-bits hash length. Proposed 1992. Collisions found 2004 after years of effort. Nowadays, collisions can be found in seconds on an ordinary PC.

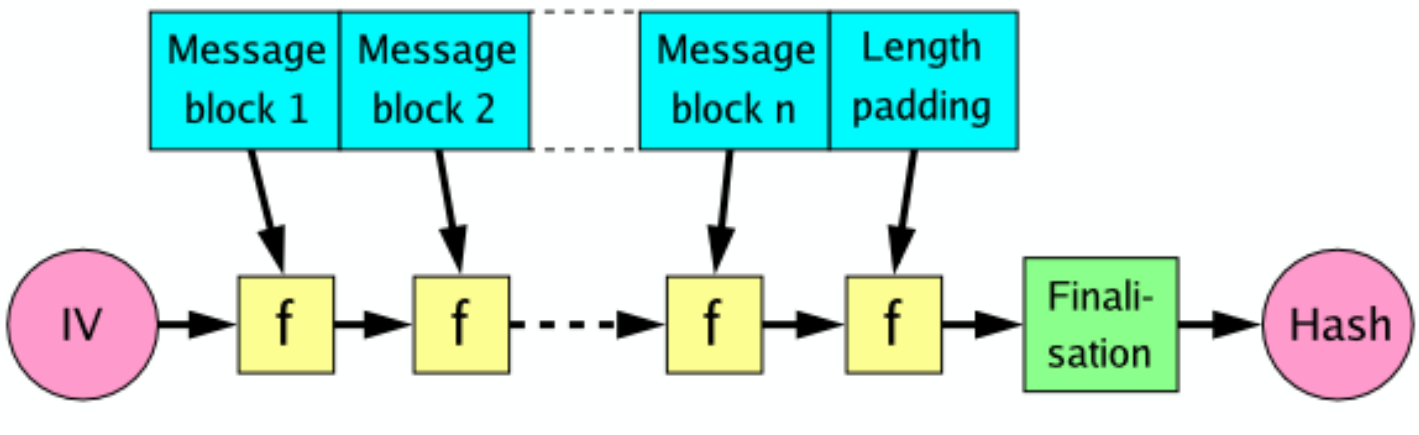
SHA-1 160-bits hash length. Proposed 1995. ~~No collisions have been found. However, it has been shown that they could be found with around 2^{65} operations – vastly less than the theoretical 2^{80}.~~ Collisions found in February 2017. No longer recommended to be used (but still in widespread use).

SHA-2 224, 256, 384, or 512 bits hash length. Proposed 2001. No collisions found for any of the bit lengths. Still considered secure, though there are "erosions".

SHA-3 variable digest size. Proposed 2015.

The Merkle-Damgård Construction

Merkle-Damgård Construction produces a cryptographic hash function from a *compression function* shown as f below.
Idea: Apply compression function repeatedly



Source: Wikipedia

MD4, MD5, SHA-1 and SHA-2 all use the Merkle-Damgard construction. Only SHA-3 does something completely different.

The algorithm for MD4

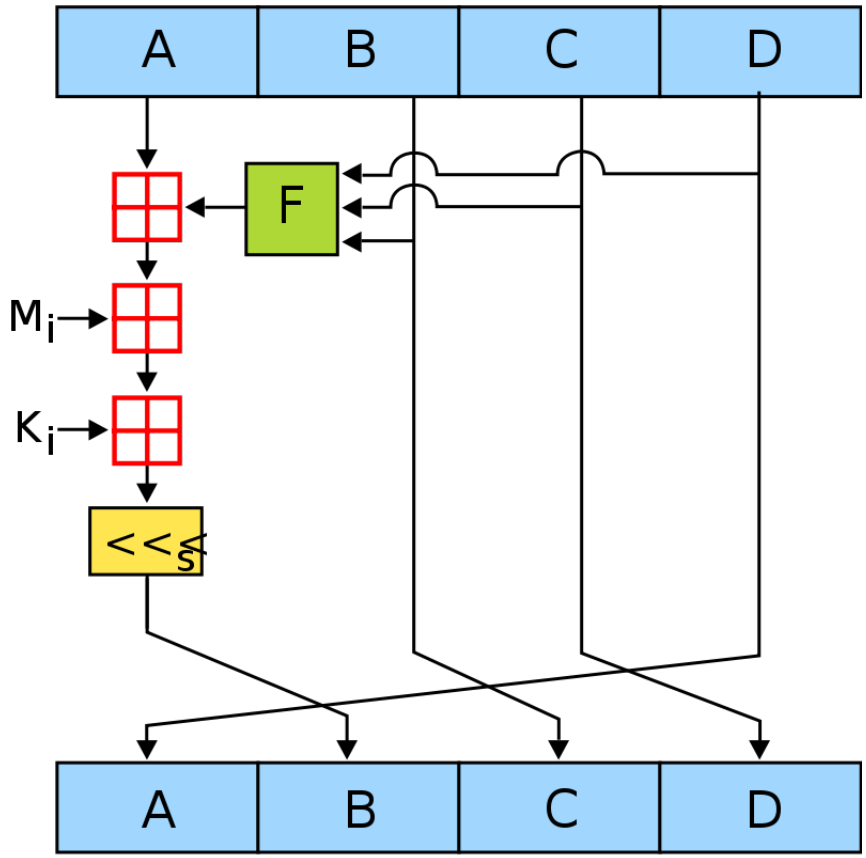
- First, the message is padded to a length that is 64 bits less than a multiple of 512 bits. The padding is 1 followed by a string of 0s.
- The padding is completed by adding a 64 bit representation of the length of the original message. So now the padded message is a multiple of 512 bits.
- A, B, C, D are initialised to some fixed constants (they're just part of the definition of MD4).
- The message is split into 512 bit blocks. Each block is processed in turn:
  - The compression function is applied to A,B,C,D and the current block, resulting in a new value of A,B,C,D.
- The hash value is the final value of A,B,C,D.

The compression function for MD4

- Input: 512 bit block of the message, and current value of A,B,C,D.
- Output: new values of A,B,C,D.
- The 512 bit message is split into 16 chunks $M_0, \ldots, M_{15}$ of 32 bits each
- Three rounds, each of 16 steps, are used to transform A,B,C,D into new values of A,B,C,D. Each of the 16 steps consumes one of the chunks of the message. The rounds make use of three special functions, one for each round:

$$\begin{aligned} F(X,Y,Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\ G(X,Y,Z) &= (X \wedge Z) \vee (Y \wedge Z) \vee (X \wedge Z) \\ H(X,Y,Z) &= X \oplus Y \oplus Z \end{aligned}$$

One of the 16 steps from one of the rounds of MD4. In the picture, M_i is the 32-bit chunk of the message the step consumes. K_i is a 32-bit constant



Source: Wikipedia

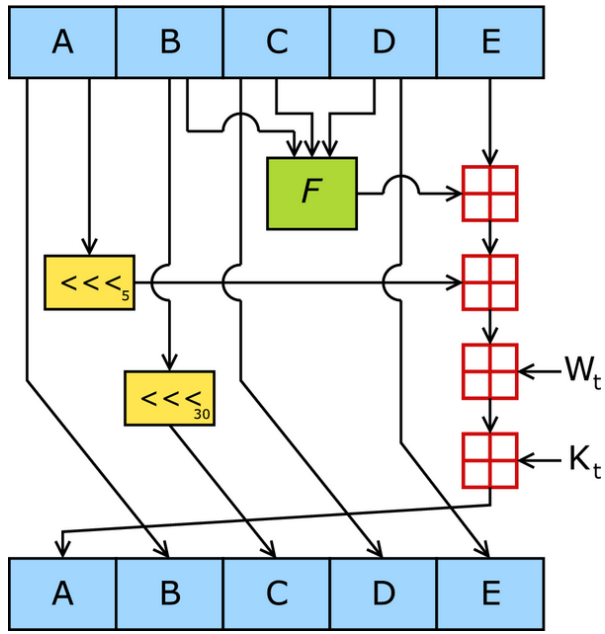
MD5

Same parameters, same initialisation vector
Adds fourth round with a different non-linear function

$$I(X, Y, Z) = Y \oplus (X \vee \neg Z)$$

SHA-1

SHA-1 extends hash size to 160 bits
Extension of MD4 (same non-linear functions used)
Message blocks used differently
Recent usage: TLS, SSL, SSH, BitTorrent



Source: Wikipedia

SHA-2

Successor of SHA-1
Introducing more bitwise operations
increasing block sizes
increasing hash length...

but essentially the same ideas.

Length extension attack

SHA-1, SHA-256 and SHA-512 are vulnerable to length extension attacks:

Given $h(x)$, one can construct $h(x||x')$ for certain x', without knowing x. This is achieved by defining
$x' = padding(x)||length(x)||....$

Although this isn't a devastating attack, it prevents a certain way of using hash functions to produce MACs (see later).

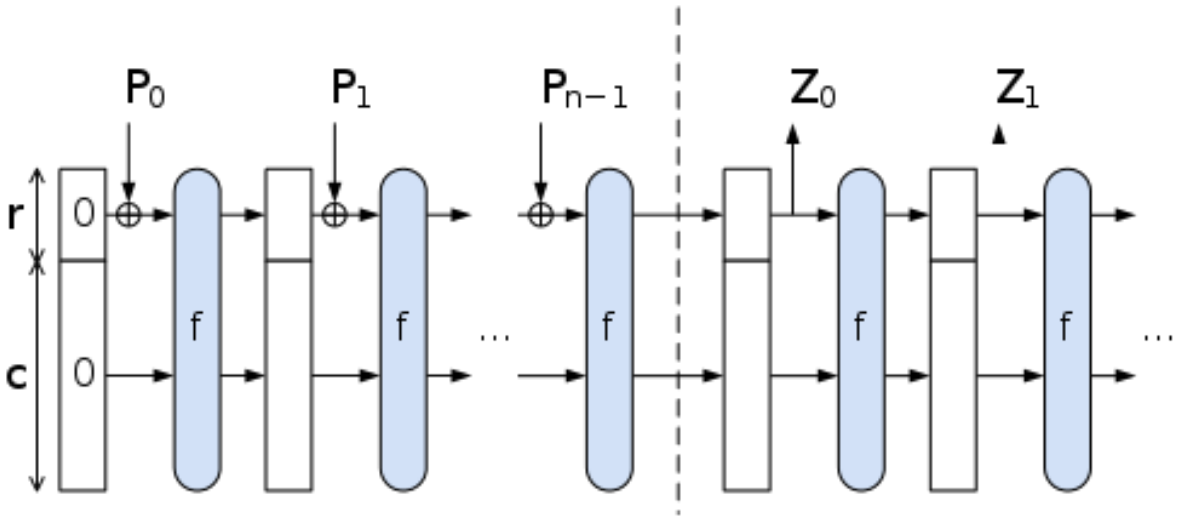
SHA-3

NIST held a competition for new hash functions, which ended in 2015. Keccak was the winner, and one of its family is now known as SHA-3.

In contrast with previous hash functions, it uses the *sponge construction* rather than the *Merkle-Damgard construction*.

SHA-3 can output hash values of an arbitrary length. It also has a parameter c ("capacity") that can be varied. For capacity c, SHA-3 gives $c/2$ bits of security.

SHA-3



Source: Wikipedia

SHA-3 uses a 1600-bit *block transformation function* f. Thus, $r + c = 1600$.

In SHA-3, the security parameter (c) and the output length (the length of the Z's added together) can be varied independently.

SHA-3

1. pad the input using the pad function, yielding a padded bit string P with a length divisible by r.
2. break P into n consecutive r-bit pieces P_0, ..., P_{n1}.
3. initialize the state S zeros.
4. absorb the input into the state: for each block P_i:
   - extend P_i at the end by a string of c zero bits
   - XOR that with S
   - apply the block permutation f to the result, yielding new S
5. initialize Z to be the empty string
6. while the length of Z is less than d:
   - append the first r bits of S to Z
   - if Z is still less than d bits, apply f to S, yielding a new S
7. truncate Z to d bits.

SHA-3 block permutation

Uses XOR, AND and NOT operations, and is designed for easy implementation in both software and hardware.
It maps 1600 bits to 1600 bits. The 1600 bits are structured as a 5x5 array of 64-bit words.
The basic block permutation function consists of 24 rounds of five steps:

- θ (theta). Compute the parity of each of the 320 5-bit columns, and exclusive-or that into two nearby columns in a regular pattern.
- ρ (rho). Bitwise rotate each of the 25 words by a different triangular number 0, 1, 3, 6, 10, 15, ...
- π (pi). Permute the 25 words in a fixed pattern.
- χ (chi). Bitwise combine along rows. This is the only non-linear operation in SHA-3.
- ι (iota). Exclusive-or a round constant into one word of the state. This breaks the symmetry that is preserved by the other steps.

Playing paper-scisors-stone asynchronously

How could Alice and Bob play paper-scisors-stone by WhatsApp?