Hash Functions

Ali El Kaafarani

[1]Mathematical Institute
[2] PQShield Ltd.

Outline

1. The Random Oracle Model
2. Hash Functions: Constructions

Outline

The Random Oracle Model

- Sometimes it is NOT enough for a hash function to be collision resistant/preimage resistant to be able to write a security proof of some cryptosystems that use hash functions.

The Random Oracle Model

- Sometimes it is NOT enough for a hash function to be collision resistant/preimage resistant to be able to write a security proof of some cryptosystems that use hash functions.
- Instead of using cryptosystems that have no proofs at all. They "idealized" the cryptographic hash functions!

The Random Oracle Model

- Sometimes it is NOT enough for a hash function to be collision resistant/preimage resistant to be able to write a security proof of some cryptosystems that use hash functions.
- Instead of using cryptosystems that have no proofs at all. They "idealized" the cryptographic hash functions!
- Let us consider a hash function that is *truly random*.

The Random Oracle Model

- Sometimes it is NOT enough for a hash function to be collision resistant/preimage resistant to be able to write a security proof of some cryptosystems that use hash functions.
- Instead of using cryptosystems that have no proofs at all. They "idealized" the cryptographic hash functions!
- Let us consider a hash function that is *truly random*.
- Additionally, assume that this random function is public, and can answer hash queries to different parties, like a black box!

The Random Oracle Model

- Sometimes it is NOT enough for a hash function to be collision resistant/preimage resistant to be able to write a security proof of some cryptosystems that use hash functions.
- Instead of using cryptosystems that have no proofs at all. They "idealized" the cryptographic hash functions!
- Let us consider a hash function that is *truly random*.
- Additionally, assume that this random function is public, and can answer hash queries to different parties, like a black box!
- If you don't idealise your hash functions in your proofs, then your cryptosystem is said to be secure in the *standard model*, otherwise, it is *only* secure in the random oracle model.

The Random Oracle Model

- Sometimes it is NOT enough for a hash function to be collision resistant/preimage resistant to be able to write a security proof of some cryptosystems that use hash functions.
- Instead of using cryptosystems that have no proofs at all. They “idealized” the cryptographic hash functions!
- Let us consider a hash function that is *truly random*.
- Additionally, assume that this random function is public, and can answer hash queries to different parties, like a black box!
- If you don’t idealise your hash functions in your proofs, then your cryptosystem is said to be secure in the *standard model*, otherwise, it is *only* secure in the random oracle model.
- In the real world, you replace your ideal hash function by an *appropriate* hash function.

The Random Oracle Model

- What do we mean by appropriate hash functions?

The Random Oracle Model

- What do we mean by appropriate hash functions?
- No clear definition!

The Random Oracle Model

- What do we mean by appropriate hash functions?
- No clear definition!
- Concrete hash functions are deterministic and fixed, they *cannot* behave like random functions!

The Random Oracle Model

- What do we mean by appropriate hash functions?
- No clear definition!
- Concrete hash functions are deterministic and fixed, they *cannot* behave like random functions!
- What does a proof in the random oracle buy us?

The Random Oracle Model

- What do we mean by appropriate hash functions?
- No clear definition!
- Concrete hash functions are deterministic and fixed, they *cannot* behave like random functions!
- What does a proof in the random oracle buy us?
- Perhaps, the scheme doesn't have "inherent design flaws"!

The Random Oracle Model

- What do we mean by appropriate hash functions?
- No clear definition!
- Concrete hash functions are deterministic and fixed, they *cannot* behave like random functions!
- What does a proof in the random oracle buy us?
- Perhaps, the scheme doesn't have "inherent design flaws"!
- Can we instantiate a random oracle using a trusted party? There are some suggestions.

The Random Oracle Model

- What do we mean by appropriate hash functions?
- No clear definition!
- Concrete hash functions are deterministic and fixed, they *cannot* behave like random functions!
- What does a proof in the random oracle buy us?
- Perhaps, the scheme doesn't have "inherent design flaws"!
- Can we instantiate a random oracle using a trusted party? There are some suggestions.
- Why is it widely used?

The Random Oracle Model

- What do we mean by appropriate hash functions?
- No clear definition!
- Concrete hash functions are deterministic and fixed, they *cannot* behave like random functions!
- What does a proof in the random oracle buy us?
- Perhaps, the scheme doesn't have "inherent design flaws"!
- Can we instantiate a random oracle using a trusted party? There are some suggestions.
- Why is it widely used?
- So far, there have been no successful *real-world* attacks on *real-world* schemes that are proven secure in the ROM. Additionally, schemes that are proven secure in the ROM are usually efficient.

The Random Oracle Model: definitions and proofs

- In ROM security proofs, the probability is taken over the random choice of H, whereas in the real world, you instantiate H by a deterministic function. Here the adversary doesn't need to query H, he can execute H himself, and even look and use its code in course of his attack!

The Random Oracle Model: definitions and proofs

- In ROM security proofs, the probability is taken over the random choice of H, whereas in the real world, you instantiate H by a deterministic function. Here the adversary doesn't need to query H, he can execute H himself, and even look and use its code in course of his attack!
- Another strong property of the random-oracle model is that if x has not been queried yet to H, then the value $H(x)$ is still considered uniform.

The Random Oracle Model: definitions and proofs

- In ROM security proofs, the probability is taken over the random choice of H, whereas in the real world, you instantiate H by a deterministic function. Here the adversary doesn't need to query H, he can execute H himself, and even look and use its code in course of his attack!
- Another strong property of the random-oracle model is that if x has not been queried yet to H, then the value $H(x)$ is still considered uniform.
- **Extractability:** When $\mathcal{A}$ queries x to H, the challenger learns x.

The Random Oracle Model: definitions and proofs

- In ROM security proofs, the probability is taken over the random choice of H, whereas in the real world, you instantiate H by a deterministic function. Here the adversary doesn't need to query H, he can execute H himself, and even look and use its code in course of his attack!
- Another strong property of the random-oracle model is that if x has not been queried yet to H, then the value $H(x)$ is still considered uniform.
- **Extractability:** When $\mathcal{A}$ queries x to H, the challenger learns x.
- **Programmability:** The challenger sets the (uniformly distributed) values of $H(x_i)$ to answer the adversary's queries!

Hash Functions: Additional Applications

- Fingerprinting: The digest $H(x)$ of a file x (which could be a virus) acts as a fingerprint/identifier of the file
- Deduplication: Particularly important in cloud storage, you send a hash of the file to want to store (e.g. DropBox), they check if the file already exists, in that case they don't need to store it again, a pointer to it would be enough.

Hash Functions: Additional Applications

- Merkle Trees: Suppose you have n files $x_1, \cdots, x_n$, assuming that n is a power 2. Instead of hashing them all, i.e. $H(x_1, \cdots, x_n)$, Ralph Merkle proposes a solution that works as follows:
  - Compute $h_{1,2} \leftarrow H(x_1, x_2), \cdots, h_{n-1,n} \leftarrow H(x_{n-1}, x_n)$.
  - Compute $h_{1,2,3,4} \leftarrow H(h_{1,2}, h_{3,4}), \cdots, h_{n-3,n-2,n-1,n} \leftarrow H(h_{n-3,n-2}, h_{n-1,n})$
  - Iterate, finally compute $h_{1,\cdots,n}$.
- Merkle Tree can be thought of as an alternative to Merkle Damgård transform to extend the domain of collision-resistant hash functions.
- Its drawback: it is not collision-resistant if n is not fixed!

Hash Functions: Additional Applications

Password Hashing:

- A hash of the password is usually stored instead of the password itself.
- What if the password is chosen from a small space?
- Is it enough to have a preimage resistance hash function H?
- ONLY if you are sampling your password *uniformly* from a large space, i.e. $\{0,1\}^n$ with suitable n.
- In practice: if your password is a random combination of 8 alphanumeric characters, say the space is $S = 62^8 \approx 2^{47.6}$.
- There is an attack (that does some preprocessing) which *only* uses time and space $N^{2/3} \approx 2^{32}$.
- There are mechanisms that can be used to mitigate this threat (adding a long random *salt*, etc.).

Commitment Schemes

- A commitment scheme allows a party to commit to a value v by producing a commitment on it.
- The commitment keeps that value hidden, i.e. it reveals nothing about v. This property is called *hiding*.
- The party cannot change it later on, i.e. it cannot open to two different values v_1, v_2. This property is called *binding*.
- Think of it as a sealed envelope!
- It is a very important cryptographic tool.
- It can be built using hash functions!

Commitments Schemes

Definition

A commitment scheme consists of two algorithms KeyGen *and* Commit *as follows*

- KeyGen(n) : *it outputs public parameters* p
- Commit$(\mathsf{p}, m \in \{0,1\}^n, r \in \{0,1\}^n)$: *it takes the public parameters, a message* m *and a random value* r, *it outputs* $\mathsf{com}_{(m)}$

The sender can at anytime reveal the message m *to the receiver by sending* (m, r). *The receiver can easily verify the correctness of the sender's claim by testing* $\mathsf{Commit}(\mathsf{p}, m, r) \stackrel{?}{=} \mathsf{com}_{(m)}$

Informally speaking, a commitment scheme is secure if it is both binding and hiding.

Commitments Schemes

- Suppose that we have a hash function that is modelled as a random oracle, we can define a commitment scheme where Commit $\leftarrow H(m||r)$
- Binding: follows from the fact that the hash function is collision-resistant.
- Hiding: follows from the fact that r is chosen uniformly from $\{0, 1\}^n$.
- There are other commitment schemes that don't assume the existence of a random oracle, i.e. they are proven secure in the standard model.

Outline

Hash Functions From Block Ciphers

- We construct hash functions in two steps.
- First, we construct a compression function h which is a fixed-length hash function.
- To allow for arbitrary-length inputs, we apply some techniques, e.g. Merkle-Damgård tranform, to extend h.
- We can use a *special* block cipher to build a collision-resistant compression function.
- Davies-Meyer method is the most common one.
- Given a block cipher with n-bit key and ℓ-bit block, we can build the compression function h as follows:

$$h : \{0,1\}^{n+\ell} \to \{0,1\}^{\ell}$$

$$h(k,x) \leftarrow F_k(x) \oplus x$$

Hash Functions From Block Ciphers

- Assuming that the F is a strong pseudo-random permutation is NOT enough to prove collision resistance of h.
- We need to rely on something similar to the random oracle model's idea.
- We have to model F as an ideal cipher.
- This means having a public oracle for computing a random keyed permutation $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$ and its inverse F^{-1}.
- Similar to the random oracle model, to compute $F(k,x)$ or $F^{-1}(k,x)$, you can only do that by querying the oracle.

MD5

- Designed in 1991. It has 128-bit output length.
- Totally broken, collisions can be found in less than a minute on a PC!

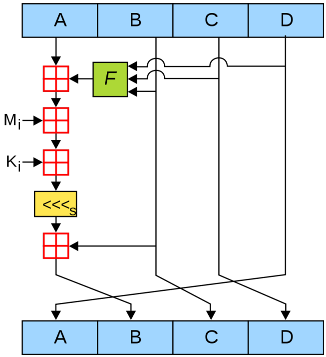


Figure: One MD5 operation (From wikipedia)

MD5

- MD5 consists of 64 operations.
- They are grouped in four rounds, each of 16 operations.
- We have 4 non-linear functions, F, G, H, I;
- One function is used in each round.
- M_i denotes a 32-bit block of the message input.
- K_i denotes a 32-bit constant, different for each operation.
- $\lll_s$ denotes a left bit rotation by s places; s varies for each operation.
- Addition is done modulo 2^{32} (You basically ignore the bit number 33).

MD5

It uses 4 functions that each takes as input three 32-bit words and generate as output one 32-bit word:

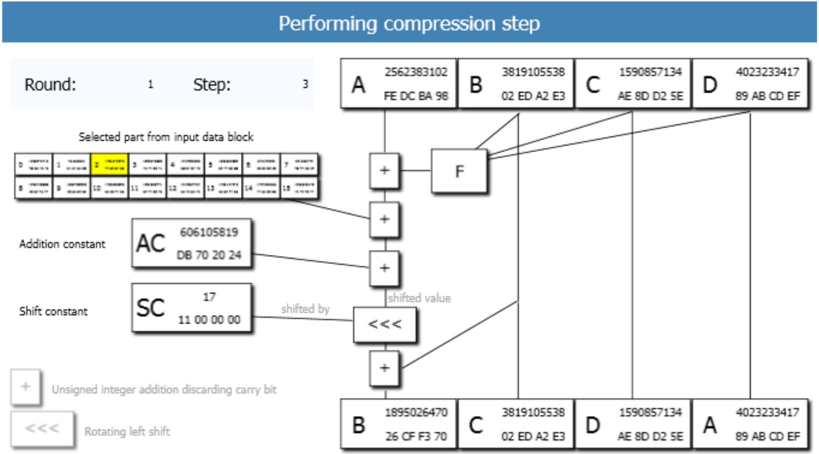
$$\mathbf{F}(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$$

$$\mathbf{G}(B, C, D) = (B \wedge D) \vee (C \wedge \neg D)$$

$$\mathbf{H}(B, C, D) = B \oplus C \oplus D$$

$$\mathbf{I}(B, C, D) = C \oplus (B \vee \neg D)$$

MD5



From Cryptool software.

Secure Hash Algorithms: SHA-1 and SHA-2

- A family of cryptographic hash functions standardized by NIST.
- First, they all use Davies-Meyer construction to build a compression function from a block cipher.
- The block cipher were specifically designed for this purpose.
- The block cipher SHACAL-1 with 160-bit block length for SHA1.
- The block cipher SHACAL-2 with 256-bit block legnth for SHA2.
- The key length is 512-bit in both of them.
- Second, they extend using Merkle-Damgård to handle arbitrary input-length.

SHA-1

- SHA-1 was introduced in 1995.
- It has 160-bit output length and it consists of 80 rounds.
- In theory, collisions can be found significantly better that the birthday attack, i.e. much less 2^{80} hash functions evaluations.
- ~~In practice, no collisions of this type. But highly recommended to~~ move to SHA-2 (or perhaps to SHA-3). SHAttered- Move now to SHA-2!
- Very recent attack (see references at the last slide).
- Example that shows the steps of SHA-1: `http://www.metamorphosite.com/one-way-hash-encryption-sha1-data-software`

SHA-1

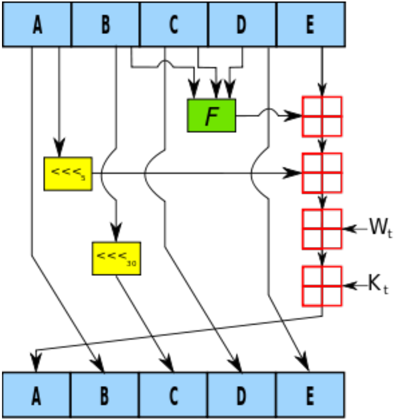
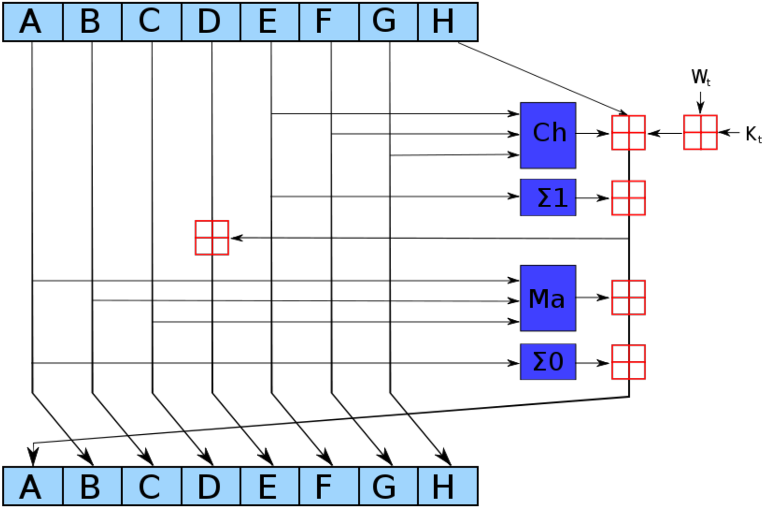


Figure: From wikipedia

SHA-2: SHA-256

- Similar to MD5 and SHA-1.
- First, given a message M s.t. $|M| = \ell$. Append it with 1, then $448 - (\ell + 1)$ zeros, and finally with the number ℓ written in binary.
- Now the padded message is a multiple of 512 bits.
- Parse it into N blocks of size 512 bits, i.e. $M^{(1)}, \cdots, M^{(N)}$.
- Fix the initial hash values $H_1^0, \cdots, H_8^{(0)}$ with the fractional parts of the square roots of the first eight primes.
- Compute $H^{(i)} = H^{(i-1)} + C(M^{(i)}, H^{(i-1)})$ where C is the compression function and addition is word-wise mod 2^{32}.
- Output $H^{(N)}$ as the hash of the message M.
- For detailed description see: `http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf`

SHA-2



SHA-256

The logical functions are as follows:

- $Ch(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$
- $Ma(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$
- $\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$
- $\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$

The constant words, $K_0, \cdots, K_{63}$ are the first 32 bits of the fractional parts of the cube roots of the first sixty-four primes.

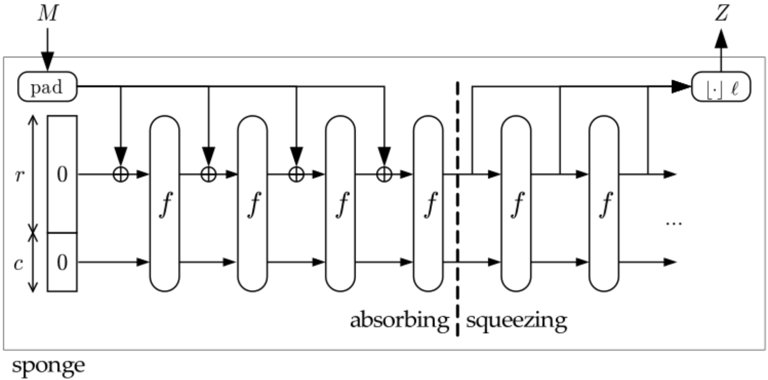
SHA-256

- 24-round-SHA-256 is broken.
- Variants of SHA-256 without $\sigma_0, \sigma_1, \Sigma_0, \Sigma_1$ have been broken as well.

SHA-3 (Keccak)

- In 2012, Keccak was announced as the winner of the NIST competition (was called SHA-3) to design a new cryptographic hash function.
- All candidates were of 256- and 512-bit output length.
- Its structure is different from SHA-1 and SHA-2.
- it uses an *unkeyed* permutation with 1600-bit block length!
- For instance, Davies-Meyer construction uses a keyed permutation
- it doesn’t use Merkle-Damgård to extend the compression function to deal with arbitrary-length input.
- *Sponge construction* is the new approach that it uses instead of Merkle-Damgård.

Keccak- Sponge Function



Complete description:

`http://sponge.noekeon.org/CSF-0.1.pdf`

Further Reading (1)

- Mihir Bellare and Phillip Rogaway.
  Random oracles are practical: A paradigm for designing efficient protocols.
  In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.
- Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche.
  Keccak sponge function family main document.
  Submission to NIST (Round 2), 3:30, 2009.

Further Reading (2)

- Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya.
  Merkle-damgård revisited: How to construct a hash function.
  In *Advances in Cryptology–CRYPTO 2005*, pages 430–448. Springer, 2005.
- Pierre Karpman, Thomas Peyrin, and Marc Stevens.
  Practical free-start collision attacks on 76-step sha-1.
  In *Advances in Cryptology–CRYPTO 2015*, pages 623–642. Springer, 2015.
- Neal Koblitz and Alfred J Menezes.
  The random oracle model: a twenty-year retrospective.
  Designs, Codes and Cryptography, pages 1–24, 2015.

Further Reading (3)

- Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone.
  Handbook of applied cryptography.
  CRC press, 1996.
- Marc Stevens.
  New collision attacks on sha-1 based on optimal joint local-collision analysis.
  In *Advances in Cryptology–EUROCRYPT 2013*, pages 245–261. Springer, 2013.