CRYPTOGRAPHIC HASH FUNCTIONS

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Contents

1. Definitions

We have encountered *cryptographic primitives*, such as pseudorandom functions (PRF), pseudorandom permutations (PRP) and pseudorandom generators (PRG). Each is a P-time or Probabilistic P-time algorithm with additional complexity (said to be "security") requirements. They have been shown to be equivalent.

We now look at Hash functions, and other cryptographic primitive. It is very flexible and has many uses as the dual to encryption, such as authentication.

Definition 1.1. A *Hash Function* if a pair of PPT algorithms, $\langle G, H \rangle$ such that for security parameter n,

(1) $G(1^n)$ returns a pair, $k \in K_n$ some a key space with size dependent on n, and $l(n)$ an integer, for some function l,
(2) the hash function computes,

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

and satisfies certain security properties expressed in the security parameter n.

Date: October 20, 2025.

Definition 1.2. A *Compression Function* is a Hash function where G also outputs an integer $l'(n)$ with $l'(n) > l(n)$ and the hash function is defined as,

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

and H satisfies certain security properties expressed in the security parameter n.

Definition 1.3. A hash function family $\langle\, G, H\,\rangle$ is *collision resistant* if for any adversary, PPT algorithm $\mathcal{A}$,

$$Pr\{\,\mathcal{A}^{\langle\, G,H\,\rangle}(n) = (x, x') \text{ s.t. } x \neq x' \text{ and } H_k(x) = H_k(x')\,\} < negli(n).$$

Definition 1.4. A hash function family H is *preimage resistant* if for any PPT algorithm $\mathcal{A}$ (the adversary),

$$Pr\{\,\mathcal{A}^{\langle\, G,H\,\rangle}(n, y) = x \text{ s.t. } y \in_U \{\,0,1\,\}^{l(n)} \text{ and } H_k(x) = y\,\} < negli(n).$$

Nota Bene: For the attack against a hash, the adversary will have *oracle access* to the functions G and H. The adversary will call G on time, with 1^n as input, and receive the key and the length parameters. It then calls H any number of times with this key and according to the length parameters.

There is a weaker security notion where the adversary only gets oracle access to H_k. That is, the key is chosen in secret and kept secret from the adversary. This models a *keyed hash function.* In practice hash functions are neither keyed nor un-keyed in that they are a single fixed thing. So they are like our definition of hash functions except that $G(n)$ is deterministic.

Theorem 1.1. If a hash family is collision resistant, it is preimage resistant .

Theorem 1.2. There exists a preimage resistant hash function family which is not collision resistant.

It is simplest to focus on collision resistant hash functions.

A bit of philosphy: A hash function can be thought of as the dual of a PRG. For a PRG, from a short random bit string, the seed, is derived a long random bit string. A hash takes a long bit string and compresses it into a short random bit string.

- A PRG's security requirement is the indistinguishability of the resulting string from a truly random stream of coin flips.
- A hash function's security requirement is collision resistance (or the weaker preimage resistance).

2. Merkle–Damgård construction

The Merkel–Damgård construction (Figure 1) iterates a compression function to create a hash function. In addition to the construction, a padding function should be defined to fully implement inputs of arbitrary length. In practice, the length is arbitrary up to byte length. Messages measured in bit length are rare in the context of software, although hardware encryption "on the wire" might require messages in increments of one bit.

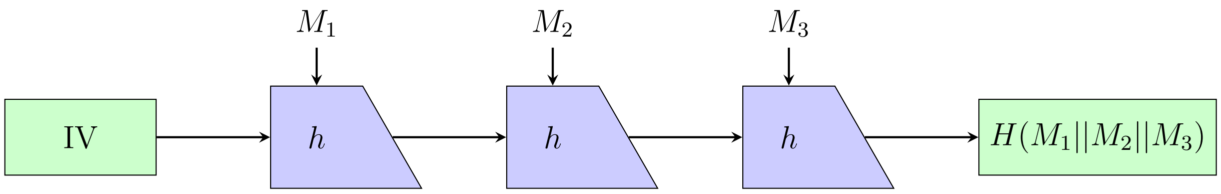

Figure 1. Merkle–Damgård for compression function $h \in H$.

With proper padding, a Merkle-Damgård hash is collision resistant if the hash family H is collision resistant.

3. Hash and MAC, and HMAC

One purpose of a hash function is to create a MAC on a message. For this an element of secrecy is introduced.

A MAC then Hash scheme will MAC the message then encrypt then encrypt the resulting hash.

Attempts are made to combine the hash with the encryption in the creation of *keyed hash functions*. It is possible to create keyed hash functions that fail to give existentially unforgeable MACs. One example is to key the hash by replacing the IV in a Merkle-Damgård with a secret key. A *length extension attack* simply asks for a MAC on the prefix of a message and then continues the iteration since the hash itself is known.

Those attacks can be disqualified if that the domain of hashable values is *prefix free*. That is, for two eligible inputs, x and x', it is neither true that there exits a string y such that $x = x'||y$ or $x' = x||y$. On this domain, the given construction is secure.

A solution for creating a MAChash is the *Hash-based Message Authentication Code* (HMAC). HMAC was introduced by Bellare, Canetti, and Krawczyk in 1996 as a method to use a cryptographic hash function with a secret key to create a message authentication code. It was formally standardized by the IETF as RFC 2104 in 1997.

The formula is,

$$\text{HMAC}_K(M) = H\Big((K' \oplus \text{opad}) \parallel H\big((K' \oplus \text{ipad}) \parallel M\big)\Big)$$

where

(1) K' is a secret key, to make the overall hash keyed,
(2) and opad and ipad are constants. RFC 2014 defined these constants as,

$$\text{ipad} = \underbrace{\text{0x36 0x36} \ldots \text{0x36}}_{B \text{ bytes}}$$
$$\text{opad} = \underbrace{\text{0x5C 0x5C} \ldots \text{0x5C}}_{B \text{ bytes}}$$

4. Hashing and Encryption

Hashes can be used for a large number of things. The Proof-of-Work algorithm of Bitcoin was a very successful principle used to build digital money. It depends on pre-image resistance. It testifies publicly that a certain agent, the miner, has fairly flipped a heavily biased coin and has won the flip. For many miners, collusion is minimal. In essence, a collusion of miners will have an advantage only in proportion to their computing power.

It is also used in Bitcoin to build the *blockchain*. Bitcoin works on a public-ledger model. Participants don't trust, they verify. All participants can check the blockchain for consistency based on a final hash of all the contents of the blockchain. This requires the hash to be collision resistant.

4.1. **Davies–Meyer: From Encryption to Hashing.** The Hash primitive should be compared to PRP's. For instance, given a PRP can one create a hash. One possibility is a $2n$ to n compression by taking the input, using one half as the key to an encryption, and the other half as the input. However for this function collisions are easy to find. Pick a random Y and two keys R_1 AND R_2. Given the decryption unit compute,

$$L_1 = D_{R_1}(Y),\ L_2 = D_{R_2}(Y)$$

Then the inputs $L_1||R_1$ and $L_2||R_2$ collide.

The *Davis-Meyer* construction gets right but suggesting,

$$H(R||L) = E_R(L)\ \oplus\ L$$

5. The Random Oracle Model

The other cryptographic primitives have ideal counterparts. A PRG has a truly random sequence. A PRF has a truly random function. A PRG has a truly random permutation. The ideal Hash is the *Random Oracle Model* (ROM). It is the truly random assignment of a n bit string to a unspecified length bit string.

A way to build a ROM is a program with an initially empty list of input-output pairs. For each input in the list, return the corresponding output. For an input not in the list, flip n coins and append to the list the pair: the given input, with the n bit output from the coins.

Then proceed to reason about cryptographic constructions using a ROM and then substituting a keyed hash function with a randomly chosen key. However, this does not work. It has been proved that some constructions are secure in the Random Oracle Model, but never secure for hash functions that are algorithms.

6. On Hashing and Pseudorandomness

Definition 6.1. A pseudorandom function is a function drawn randomly from a multi-set of of functions

$$f_s \in F_n,\ F_n : \{\,0,1\,\}^{l(n)} \to \{\,0,1\,\}^{l(n)}$$

where $s \in \{\,0,1\,\}^*$, $n = |s|$, $|F_n| = 2^n$, and $l : \mathbb{N} \to \mathbb{N}$ is a function.
The functions f_s must be effectively computable — there is a P-time algorithm F such that $F(s,x) = f_s(x)$. The ensemble F_n must be computationally indistinguishable from a truly random function drawn from the the ensemble of all functions $f \in \mathcal{F}_n : \{\,0,1\,\}^{l(n)} \to \{\,0,1\,\}^{l(n)}$.

Definition 6.2. A collision resistant hash function is a function drawn randomly from a multi-set of of functions

$$h_s \in H_n,\ H_n : \{\,0,1\,\}^* \to \{\,0,1\,\}^{l(n)}$$

where $s \in \{\,0,1\,\}^*$, $n = |s|$, $|H_n| = 2^n$, and $l : \mathbb{N} \to \mathbb{N}$ is a function.
The functions h_s must be effectively computable — there is a P-time algorithm H such that $H(s,x) = h_s(x)$. Over the ensemble H_n it must be be computationally infeasible to compute two values x, x' such that $x \neq x$ and $h_s(x) = h_s(x')$.

Note several differences.

(1) For pseudorandom functions, the s is considered secret. Else it would be easy to tell f_s from a truly random s. For hash functions, the s is public. Else the device loses its usefulness.
(2) For pseudorandom functions, the security property is that the function look random. For hash functions, the security property is collision resistance (or e.g. pre-image resistance).
(3) A hash function is indexed by output length. A pseudorandom function is indexed by output and input length. In the definition here, for simplicity, these lengths are the same. Other definitions allow those lengths to be different.

Lemma 6.1. There are secure hash functions which are not pseudorandom functions.

This would require restricting the hash function of fixed lengths. Since it is collision resistant without this restriction, it is collision resistant with this restriction. However, the hash function could have values that have only even parity. Adjusting one bit of the output will accomplish that and the function is still collition resistant. This function is distinguished from a random function. □

However, if a hash function is not collision resistant, is is not pseudorandom, since a random function is collision resistant.

6.1. **CBC-MAC.**

The CBC-MAC achieved popularity as an integrity and authenticity mechanism required to have secure communication. In effect, our IND-CPA schemes become IND-CCA by adding a MAC (when done properly).

The CBC-MAC, which for prefix-free messages is a proven good MAC (by existentially unforgeability) despite the fact that the scheme as collisions (and is therefore not a pseudorandom function). However the attack model is crafted with a restriction that removes the collisions and it is possible to carry out this restriction on practice.

Here we shall carry out the attack leading to a collision. Let

$$C_1 = E_K(M_1) \quad \text{and} \quad C_2 = E_K(M_2)$$

the a collision exists between messages $M_1||0$ and $M_2||(C_2 \oplus C_1)$,

$$\begin{aligned} C_1' &= E_K(0 \oplus E_K(M_1)) = E_K(C_1) \\ C_2' &= E_K(C_2 \oplus C_1 \oplus E_K(M_2)) = E_K(C_1) \end{aligned}$$

This doesn't rule out CBC-MAC as a MAC because of the details of the existentially unforgeability game. It could be won if the adversary could ask for a MAC on $M_1||0$, and then declare victory with a forged MAC on $M_2||(C_2 \oplus C_1)$. But by the rules, it cannot ask for this MAC, as it has asked for a MAC on M_1, which is a prefix of $M_1||0$.

A technique like proper padding to block length, and prepending a representation of the number of blocks in the message can enforce that the acceptable inputs are those from a prefix-free family. Placing this information at the end of the message does not work. Here is how a forgery would work in that case.

Theorem 6.1. It is possible to forge a CBC-MAC when the block length is placed as the last block.

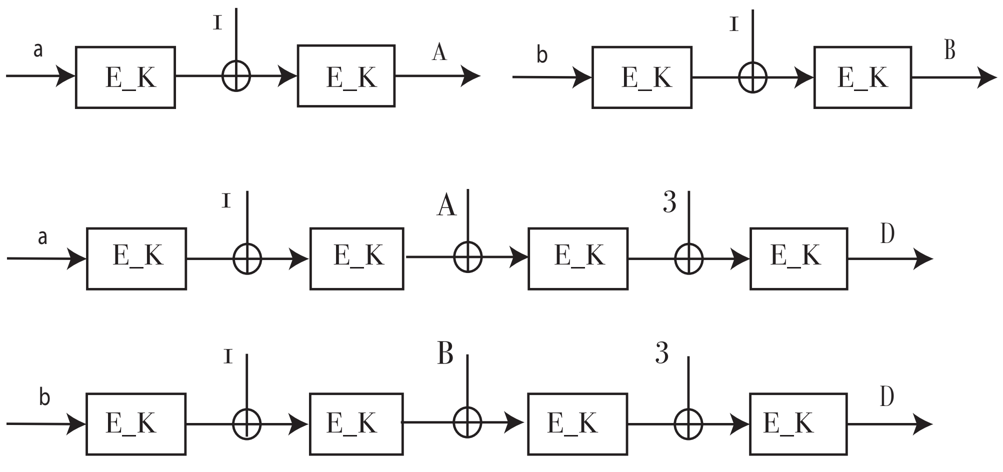

FIGURE 2. Forgery Length Extension.

Proof: The adversary queries for the tags A and B one the block messages a and b,

$$A = E_k(1 \oplus E_k(a)), \quad B = E_k(1 \oplus E_k(b))$$

The adversary queries for the tag D on the three block message $a||1||A$,

$$D = E_k(3 \oplus E_k(A \oplus E_k(1 \oplus E_k(a)))) = E_k(3 \oplus E_k(A \oplus A)) = E_k(3 \oplus E_k(0)).$$

The successful forgery is that the tag on $b||1||B$ is also D,

$$E_k(3 \oplus E_k(B \oplus E_k(1 \oplus E_k(b)))) = E_k(3 \oplus E_k(B \oplus B)) = E_k(3 \oplus E_k(0)) = D.$$

See Figure 2. $\square$

6.2. **Keyed and unkeyed Hash functions.**

Definition 6.3. A *function ensemble* is a pair of functions $l,'' : \mathbb{N} \to \mathbb{N}$ and a function,

$$H : \{0,1\}^* \longrightarrow \left(\{0,1\}^{l'(n)} \to \{0,1\}^{l(n)}\right)$$

where for $H(s)$, $n = |s|$, and $l'(n) \geq l(n)$ for all n.

PRF, Hash functions are keyed hash functions are all function ensembles, each with a distinct security property. Note that these hash functions are called *compression functions*, because the input length is exactly constrained. For the more versatile notion of a hash function, additional constructions are required, and these constructions must show to preserve the security property.

Definition 6.4. A *pseudorandom function* is a function ensemble with the security property, for any PPT adversary $\mathcal{A}$ with oracle access to the function f_s chosen from the ensemble F or a truly random function f with the domain and range of f_s,

$$\big|\, Pr(\mathcal{A}^{f_s})(n) - Pr(\mathcal{A}^{f})(n) \,\big| < negl(n).$$

This property is called *adversarial indistinguishability.*

Definition 6.5. A *hash function* is a function ensemble with the security property, for any PPT adversary $\mathcal{A}$ with oracle access to the function h_s chosen from the ensemble H,

$$\big(\, Pr(\mathcal{A}^{h_s})(n, s) \to (x_1, x_2) \text{ s.t. } h_s(x_1) = h_s(x_2) \,\big) < negl(n).$$

This property is called *collision resistance.*

Definition 6.6. A *keyed hash function* is a function ensemble with the security property, for any PPT adversary $\mathcal{A}$ with oracle access to the function s_k chosen from the ensemble S,

$$\big(\, Pr(\mathcal{A}^{s_k})(n) \to (m, t) \text{ s.t. } s_k(m) = t \,\big) < negl(n).$$

provided that $\mathcal{A}$ never queried s_k on m. This property is called *existential unforgeability on a chosen message attack.* (EUF-CMA).