Hash Functions and Applications to Modern Cryptography

Soham Dam

August 17, 2024

Abstract

Hash functions are a type of one-way function used in many branches of modern cryptography. They are used in many applications, such as digital signatures and identity verification. One way functions are essential to cryptography in order to prevent an attacker from decrypting a message from the hashed ciphertext.

Contents

1 Introduction

Section 2 of this paper will introduce one-way functions and their uses in cryptography. Section 3 will dive further into the rationale behind one-way functions, and section 4 will introduce hash functions. Section 5 will discuss what it means for a cryptosystem to be secure. This will set up section 6, where different hashing algorithms will be covered. The final section will demonstrate how hashing algorithms are applied to different cryptosystems.

1.1 Mathematical Bounding Conventions

Before we dive into one-way functions, we give mathematical definitions of negligible and super-poly functions:

Definition 1.1. A function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is negligible if for all positive real numbers c, there exists positive integer n_0 such that for all $n \geq n_0$, we have $|f(n)| < 1/n^c$. A function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is super-poly if $1/f$ is negligible.

2 One-way Functions

A one-way function is a function that is very easy to compute, but hard to reverse. As we shall see, various types of hash functions satisfy this one-way property. Instead, we look at other functions that satisfy the one-way property.

2.1 Trapdoor Functions

A trapdoor functions works as follows: we know that it is hard to reverse a one-way function, but given a certain key piece of information (the trapdoor), we can easily reverse the computation. The formal definition from [BS23] is as follows:

Definition 2.1 (Trapdoor Function Scheme)**.** Let X and Y be finite sets. The trapdoor function scheme T, defined over (X, Y), is a triple of algorithms (G, F, I), where

- G is a probabilistic key generation algorithm that is invoked as $(pk, sk) \xleftarrow{R} G()$, where pk is the public key and sk is the secret key.

- F is a deterministic algorithm that is invoked as $y \leftarrow F(pk, x)$, where $x \in X$. The output y is an element of Y.
- I is a deterministic algorithm that is invoked as $x \leftarrow I(sk, y)$, where $y \in Y$. The output x is an element of X.

With this definition, the secret key acts as the trapdoor of the scheme. Now consider the following attack game:

- We are given a trapdoor scheme T as defined in Definition 2.1 and an adversary A.
- The challenger computes

$$(pk, sk) \xleftarrow{R} G(), \qquad x \xleftarrow{R} X, \qquad y \leftarrow F(pk, x)$$

  and sends (pk, y) to the adversary.
- The adversary outputs $\hat{x} \in X$.

Define $\text{OWadv}[A, T]$, the adversary's advantage in inverting T, as the probability that $\hat{x} = x$. A trapdoor function scheme is one-way if for all efficient adversaries A, the quantity $\text{OWadv}[A, T]$ is negligible.

2.2 Key Exchange

Alice and Bob can exchange a key using the trapdoor function scheme as follows:

1. Alice computes $(pk, sk) \xleftarrow{R} G()$ and sends pk to Bob.
2. Upon receiving pk from Alice, Bob computes $x \xleftarrow{R} X$, $y \leftarrow F(pk, x)$ and sends y to Alice.
3. Upon receiving y from Bob, Alice computes $x \leftarrow I(sk, y)$.

The element x is Alice and Bob's shared key. Note the similarity to Diffie-Hellman Key Exchange.

2.3 RSA

The RSA cryptosystem also makes use of a trapdoor function scheme. The difficult step of breaking an RSA key is factoring the modulus n. This number usually consists of two very large prime factors. However, clever factorization algorithms and primality tests have been developed to factor n.

3 Why One-way Functions?

In order to prevent an attacker from decrypting messages, we want to give em a problem that is computationally hard. In other words, we want an encryption function that allows us to easily encrypt messages but hard to decrypt without the key. In other words, we must recreate the trapdoor scheme.

3.1 Chosen Plaintext Attacks

An eavesdropper can decrypt ciphertexts in general if they are able to find ways to decrypt certain ciphertexts. Ey can then use the plaintext they obtain to find the decryption key.

One method against chosen plaintext attacks is to use one-time padding. In other words, after Alice and Bob use a key, they switch to another one. This will eliminate the chances that their messages can be decrypted since the key keeps changing.

3.2 Time

The idea behine a one-way function is that it only gets used temporarily. A secret is only kept for a certain amount of time, so by the time an attacker finds a way to reverse a one-way function, there would be no point of learning the secret.

4 Keyed Hash Functions

In this section, we explore keyed hash functions. Unkeyed hash function are another main type of hash function; those will be covered in section 5.

4.1 Basic Definitions and Properties

First, we define a keyed hash function:

Definition 4.1. A keyed hash function H is a deterministic algorithm that takes a key k and a message m as its two inputs; its output $t := H(k, m)$ is called a digest. We also have K as the space of keys, M as the space of messages, and T as the space of digests.

Typically, digests have a fixed size, independent of the message length. This means that gigabyte-long messages can be hashed into 256-bit digests.

We say that two messages m_0, $m_1 \in M$ form a collision for H under key $k \in K$ if $H(k, m_0) = H(k, m_1)$, but $m_0 \neq m_1$. A hash function with the property that it is hard to find a collision is called a universal hash function, or UHF. Consider the following attack game:

- Attack Game 4.1:
- We are given a keyed hash function H defined over (K, M, T) and an adversary A.
- The challenger computes $k \xleftarrow{R} K$ and keeps k to himself
- The adversary outputs two messages $m_0, m_1 \in M$.

The adversary wins the game whenever $H(m_0, k) = H(m_1, k)$, and A's advantage with respect to H, denoted UHFadv$[A, H]$, is the probability that A wins the game.

4.2 Constructing UHFs

We give several methods for how to construct a UHF.

4.2.1 Polynomials

We start with a UHF construction using polynomials modulo a prime. Let ℓ be a length parameter and p be a prime. We define a hash function H_{poly} that hashes a message $m \in (\mathbb{Z}/p\mathbb{Z})^{\leq \ell}$ to a single element $t \in \mathbb{Z}/p\mathbb{Z}$. The key space is $K := \mathbb{Z}/p\mathbb{Z}$.

Let $m = (a_1, a_2, \ldots, a_v)$ for some $0 \leq v \leq \ell$. Let $k \in \mathbb{Z}/p\mathbb{Z}$ be a key. Then we define $H_{\text{poly}}(k, m)$ to be

$$H_{\text{poly}}(k, (a_1, a_2, \ldots, a_v)) := \sum_{i=0}^{v} k^{v-i} a_i,$$

where $a_0 = 1$. The function outputs elements in $\mathbb{Z}/p\mathbb{Z}$.

With this definition, it is not too hard to compute without knowing the length of the message ahead of time. When the message ends, we obtain the final hash. With inputs m and k, Horner's method for polynomial evaluation works as follows:

1. Set $t \leftarrow 1$.
2. For $i \leftarrow 1$ to v:
3. $\quad t \leftarrow t \cdot k + a_i \in \mathbb{Z}/p\mathbb{Z}$
4. Output t

UHFs can be very brittle - an adversary who learns the values of the function at a few points can recover the secret key. Consequently, we will hide values of the UHF from the adversary, either by encrypting them or using other means.

4.2.2 A parallel UHF from a small PRF

First, we formally define a pseudo-random function, or PRF:

Definition 4.2. A pseudo-random function is a deterministic algorithm F that has two inputs: a key k and input data block x. The output $y := F(k, x)$ is the output data block. The inputs and output have their respective spaces K, X, and Y. Denote by $\text{Funs}[X, Y]$ to be the set of all functions $f : X \to Y$.

Now that we have defined a pseudo-random function, it is time to construct our hash function. The XOR-hash $F^{\oplus}$ is defined over $(K, X^{\leq \ell}, Y)$, where Y is the set of all n-bit binary strings. The algorithm works as follows:

1. $t \leftarrow 0^n$
2. For $i = 1$ to v do:
3. $\quad t \leftarrow t \oplus F(k, (a_i, i))$
4. Output t

In other words, we evaluate $F(k, (a_i, i))$ for all i, and then we sum the resulting strings using XOR.

5 Unkeyed Hash Functions and Algorithms

In the previous section, we explored keyed hash functions. Here, we explore some unkeyed hash functions and explore their properties.

5.1 Basic Definitions

As in section 4, we say that we have a collision when $H(m_0) = H(m_1)$, but $m_0 \neq m_1$. We define collision resistance in a similar way. Collision-resistant processes are discussed in the security section (section 6).

5.2 Algorithms

Presented below are various collision-resistant hashing algorithms that do not require a key.

5.2.1 The Merkle-Damgård Paradigm

Many practical constructions follow the Merkle-Damgård paradigm. The idea is to start from a collision-resistant hash function that hashes short messages and build from it a collision-resistant hash function that hashes much longer messages.

Let $h : X \times Y \to X$ be a hash function. Assume that Y is of the form $\{0,1\}^\ell$ for some ℓ. While it is not necessary, typically X is of the form $\{0,1\}^n$ for some n. The Merkle-Damgård function derived from h, denoted H_{MD}, is a hash function defined over $(\{0,1\}^L, X)$ that works as follows (the pad PB is added to M in order to make the length a multiple of ℓ):

- input: $M \in \{0,1\}^L$
- output: a tag in X
- $\hat{M} \leftarrow M \parallel PB$
- partition $\hat{M}$ into consecutive ℓ-bit blocks so that $\hat{M} = m_1 \parallel m_2 \parallel \cdots \parallel m_s$, where the m_i are elements of $\{0,1\}^\ell$
- $t_0 \leftarrow IV \in X$
- For $i = 1$ to s do:
- $\quad t_i \leftarrow h(t_{i-1}, m_i)$
- Output t_s
- (Note that $\parallel$ denotes concatenation.)

The function h is called a compression function. The constant IV is the initial value and is fixed to some pre-specified value. One could take $IV = 0^n$, but usually IV is set to some complicated string. In addition, the padding block PB must contain an encoding of the message length. A standard format for PB is

$$PB := 1000\ldots000 \parallel \langle s \rangle,$$

where $\langle s \rangle$ is a fixed-length bit string that encodes, in binary, the number of ℓ-bit blocks in M.

5.2.2 Building Compression Functions

We explore several ways to build the compression function h. These functions fall into two categories:

- Compression functions built from a block cipher. The most commonly used method is Davies-Meyer.
- Compression functions using number theoretic primitives. These are elegant constructions with clean proofs of security. However, they are generally far less efficient than the first method.

We start with a compression function built with simple modular arithmetic. Let p be a large prime such that $q := \dfrac{p-1}{2}$ is also prime. Let x and y be suitable chosen integers in the range $[1, q]$. Consider the following compression function that takes two integers in $[1, q]$ as inputs and outputs an integer in $[1, q]$:

$$H(a, b) := \text{abs}(x^a y^b \mod p),$$

where $\text{abs}(z) = z$ when $z \leq q$ and $\text{abs}(z) = p - z$ when $z > q$.

The Davies-Meyer compression function is built off of a block cipher $\mathcal{E} = (E, D)$ over (K, X), where $X = \{0,1\}^n$. The function $h_{\text{DM}} : (X \times K, X)$ is defined as

$$h_{\text{DM}}(x, y) := E(y, x) \oplus x.$$

When plugging this function into the Merkle-Damgård paradigm, the inputs are a chaining variable $x := t_{i-1} \in X$ and a message block $y := m_i \in K$. The output is the next chaining variable t_i. Note that the message block is used as the block cipher key, but the adversary has full control over the message.

Many variants of Davies-Meyer construction exists. One can use the following functions:

$$\begin{aligned} h_1(x, y) &:= E(x, y) \oplus y \\ h_2(x, y) &:= E(x, y) \oplus y \oplus x \\ h_3(x, y) &:= E(x \oplus y, y) \oplus y \end{aligned}$$

These can be shown to be collision-resistant.

6 Security

We prove special properties of the functions in section 4. Next, we prove collision-resistance of the hashing algorithms of section 5. Many of the proofs will follow the guide of [BS23]. Then, we explore attacks on some of these algorithms.

6.1 Keyed Hash Functions

We start with some definitions:

Definition 6.1. Let H be a keyed hash function over (K, M, T)

- We say that H is an ϵ-bounded universal hash function, or ϵ-UHF, if $\text{UHFadv}[A, H] \leq \epsilon$ for all adversaries A.
- We say that H is a statistical UHF if it is an ϵ-UHF for some negligible ϵ.
- We say that H is a computation UHF if $\text{UHFadv}[A, H]$ is negligible for all efficient adversaries A.

Statistical UHFs are secure against all adversaries: no adversary can win attack game 4.1 against a statistical UHF with non-negligible advantage. Note that every statistical UHF is also a computational UHF, but the converse is not true.

Theorem 6.2. *H_{poly} defined over $(\mathbb{Z}/p\mathbb{Z}, (\mathbb{Z}/p\mathbb{Z})^\ell, \mathbb{Z}/p\mathbb{Z})$ is an (ℓ/p)-UHF. If p is super-poly, that means that ℓ/p is negligible, so H_{poly} is a statistical UHF.*

Proof. Let $m_0 = (a_1, a_2, \ldots, a_u)$ and $m_1 = (b_1, b_2, \ldots, b_v)$ be two distinct messages in $(\mathbb{Z}/p\mathbb{Z})^\ell$. We show that the probability that $H(k, m_0) = H(k, m_1)$ is less than or equal to ℓ/p, where k is chosen uniformly at random in $\mathbb{Z}/p\mathbb{Z}$. Define the two polynomials:

$$f(X) := \sum_{i=0}^{u} X^{u-i} a_i$$

$$g(X) := \sum_{j=0}^{v} X^{v-j} b_j$$

in $\mathbb{Z}/p\mathbb{Z}[X]$, where $a_0 = b_0 = 1$. Then, by definition of H_{poly}, we need to show that the probability that $f(k) = g(k)$ is at most ℓ/p. Since m_0 and m_1 are distinct messages, we know that $f(X) - g(X)$ is a nonzero polynomial. Furthermore, its degree is at most ℓ; therefore it has at most ℓ roots in $\mathbb{Z}/p\mathbb{Z}$. It follows that there are at most ℓ values of $k \in \mathbb{Z}/p\mathbb{Z}$ for which $f(k) = g(k)$, so for a random k, there is at most a ℓ/p chance such that $f(k) = g(k)$. □

Theorem 6.3. *Let F be a secure PRF and assume $|Y|$ is super-poly. Then $F^{\oplus}$ is a computational UHF.*

Proof. The proof consists of two attack games. We summarize Game 0 here, and we leave Game 1 for [BS23].

Game 0: The challenger in this game computes $k \xleftarrow{R} K, f \leftarrow F(k, \cdot)$. The adversary A outputs two distinct messages U, V in $X^{\leq \ell}$. Let $u := |U|$ and $v := |V|$. We define W_0 to be the event that the condition

$$\bigoplus_{i=0}^{u-1} f(U[i], i) = \bigoplus_{j=0}^{v-1} f(v[j], j)$$

holds in Game 0. Clearly, the probability of W_0 occuring is $\mathrm{UHFadv}[A, F^{\oplus}]$. □

6.2 Collision Resistance

We now formally define collision resistance with the following attack game:

- For a given hash function H defined over (M, T) and adversary A, the adversary takes no input and outputs two messages m_0 and m_1 in M.
- We say that A wins the game if the pair of messages yields a collision. The probability that A wins the game is denoted as $\mathrm{CRadv}[A, H]$, and it is called the advantage with respect to H.
- We say that H is collision resistant if the quantity $\mathrm{CRadv}[A, H]$ is negligible.

We first prove collision resistance of the Davies-Meyer hash function, then we move to collision resistance of the Merkle-Damgård function.

Theorem 6.4 (Davies-Meyer)**.** *Let h_{DM} be the Davies-Meyer hash function derived from a block cipher $\mathcal{E} = (E, D)$ defined over (K, X), where $|X|$ is large. Then h_{DM} is collision resistant when $\mathcal{E}$ is an ideal block cipher.*

Proof. Let A be a collision finder for h_{DM} that makes at most a total of q ideal cipher queries. We shall assume that A is "reasonable," meaning that before A outputs its collision attempt $(x, y), (x', y')$, it makes corresponding ideal cipher queries: for (x, y), either a Π-query on (y, x), or a Π^{-1}-query on $(y, \cdot)$ that yields x, and similarly for (x', y'). If A is not reasonable, it can make at most two more queries in order to be reasonable. So from now on, we assume that A is reasonable and makes at most q' queries.

For $i = 1, 2, \ldots, q'$, the ith ideal cipher query defines a triple (k_i, a_i, b_i): for a Π-query (k_i, a_i), we set $b_i := \Pi_{k_i} a_i$. For a Π^{-1}-query (k_i, b_i), we set $a_i := \Pi_{k_i}^{-1} b_i$. Assume further that A makes no extraneous queries, so triples do not repeat.

If the adversary outputs a collision, then by our reasonableness assumption, for some distinct pair of indices $i, j = 1, 2, \ldots, q'$, we have $a_i \oplus b_i = a_j \oplus b_j$. Call this event Z. We thus have

$$\mathrm{CRadv}[A, h_{\mathrm{DM}}] \leq \Pr[Z].$$

Our goal is to show that

$$\Pr[Z] \leq \frac{q'(q'-1)}{2^n},$$

where $|X| = 2^n$

Consider fixed indices $i < j$. Conditioned on any fixed values of the adversary's coins and the first $j-1$ triples, one of a_j and b_j is completely fixed, while the other is uniformly distributed over a set of at least $|X| - j + 1$ elements. Thus, we have

$$\Pr[a_i \oplus b_i = a_j \oplus b_j] \leq \frac{1}{2^n - j + 1}.$$

So by the union bound, we have

$$\Pr[Z] \leq \sum_{j=1}^{q'} \sum_{i=1}^{j-1} \Pr[a_i \oplus b_i = a_j \oplus b_j] \leq \sum_{j=1}^{q'} \frac{j-1}{2^n - j + 1} \leq \sum_{j=1}^{q'} \frac{j-1}{2^n - q'} = \frac{q'(q'-1)}{2(2^n - q')}.$$

It is not too hard to check that the final fraction is at most $\frac{q'(q'-1)}{2^n}$, finishing the proof. □

Now that we have shown collision resistance for h_{DM}, we now prove collision resistance for the Merkle-Damgård hash function.

Theorem 6.5. *Let L be a poly-bounded length parameter and let h be a collision resistant hash function defined over $(X \times Y, X)$. Then the Merkle-Damgård hash function H_{MD} derived from h, over $(\{0,1\}^{\leq L}, X)$, is collision resistant.*

Proof. We digest the proof here; the remaining details can be found in [BS23].

First, we consider a collision finder B for finding h-collisions. To accomplish this, B first runs A to find a collision for messages M and M'. It then scans the two messages by scanning each block, starting from the last block.

Let the m_i and t_i be the u blocks and $u+1$ chaining variables for M, and let the m'_i and t'_i be the v blocks and $v+1$ chaining variables for M'. We know that $h(t_{u-1}, m_u) = h(t'_{v-1}, m'_v)$. If one pair of inputs is not equal, then we have a collision, and B terminates.

Otherwise we have $t_{u-1} = t'_{v-1}$ and $m_u = m'_v$. There are padding blocks in m_u and m'_v, so we have $u = v$.

We now consider the blocks of M and M' in reverse order. If there is a collision, B terminates, or else we move back another block. Eventually, we reach the first block. There is a collision at the first block, or else we have $M = M'$, contradicting the earlier assumption that M and M' produce a collision for H_{MD}. So B breaks the collision resistance of h. □

6.3 Attacks on Hash Functions

Even though the hash functions we studied are collision resistant, there do exist attacks on such hash functions. We explore one of these attacks here.

6.3.1 Joux's Attack

We briefly describe an attack specific to Merkle-Damgård hash functions. Let H_1 and H_2 be Merkle-Damgård hash functions that outputs tags in $X := \{0,1\}^n$, and set $H_{12}(M) = H_1(M) \parallel H_2(M) \in \{0,1\}^{2n}$. We would expect that finding a collision for H_{12} should take at least $O(2^n)$. This would be the case if H_1 and H_2 were independent, random functions.

We say that an s-collision for H is a set of messages $M_1, M_2, \ldots, M_s \in M$ such that $H(M_i)$ is equal for all i. Joux showed how to find an s-collision in $O((\log_2 s)|X|^{1/2})$. Thus, using Joux's method, we can find a $2^{n/2}$ collision in $O(n2^{n/2})$ time. Then by the birthday paradox, it is very likely that some M_i, M_j is also a collision for H_2. Thus, we have found a collision for H_{12} in $O(n2^{n/2})$ time.

To find an s-collision, let H be a Merkle-Damgård function over (M, X) built from compression function h. We find an s-collision $M_1, \ldots, M_s$, where each of these messages has $b := \log_2 s$ blocks. For simplicity, let s be a power of 2, so $\log_2 s$ is an integer. Let t_0 be the initial value used in the Merkle-Damgård construction.

Next, we use the birthday attack b times on h. We spend $2^{n/2}$ time to find two distinct blocks m_0, m_0' such that $h(t_0, m_0) = h(t_0, m_0')$. Let t_1 equal this common value, and we repeat for m_1 and m_1'. We continue recursively until we have our b pairs of blocks.

Now note that in the message $m_0 m_1 \cdots m_{b-1}$, we can swap an m_i with m_i' without changing the chaining variables. Therefore, we have found a 2^b collision in $b2^{n/2}$ time.

7 Applications

We discuss some applications and case studies involving hash functions.

7.1 File Integrity

Collision resistance is frequently used for file integrity. Consider a set of n critical files that change infrequently, such as a set of executables on disk. We want a method to verify that these files have not been modified by some malicious code or malware. To do so we need a small amount of read-only memory, namely memory that the malware can read, but cannot modify. We place a hash on each of the n critical files in the read-only memory so that this storage area only contains n short hashes. Now we can check integrity of file F by hashing F and comparing the resulting hash to the one stored in read-only memory. If a mismatch is found, then F is declared corrupt, which happens when the hash function H is collision resistant.

7.2 SHA256

The SHA256 hashing algorithm is a Merkle-Damgård function with $\ell = 512$ and $n = 256$. The initial value is set to

$$\text{IV} := 6A09E667BB67AE853C6EF372A54FF53A510E527F9B05688C1F83D9AB5BE0CD19,$$

when written in hex.

We can truncate the result of SHA256; this is what SHA224 does. It uses a different initial value, and it takes the leftmost 224 bits of the SHA256 output.

Next we describe the Davies-Meyer compression function. It is built from a block cipher denoted E_{SHA256}. Instead of using the XOR operator, the compression function uses addition modulo 2^{32}. Set $x_0, x_1, \ldots, x_7 \in \{0,1\}^{32}$, and $y_0, y_1, \ldots, y_7 \in \{0,1\}^{32}$. Set $x := x_0 \parallel x_1 \parallel \cdots \parallel x_7$ and $y := y_0 \parallel y_1 \parallel \cdots \parallel y_7$. We define $x \uplus y$ as follows:

$$x \uplus y := (x_0 + y_0) \parallel (x_1 + y_1) \parallel \cdots \parallel (x_7 + y_7),$$

where all additions are done modulo 2^{32}. Then the SHA256 compression function is described as

$$h(t, m) := E_{SHA256}(m, t) \uplus t \in \{0,1\}^{256}.$$

Our ideal cipher analysis in Theorem 6.4 applies to this function.

7.3 Identity Verification

Hash functions are also applicable to digital signatures and other forms of identity verification. In order for Bob to verify that a message came from Alice, he can use a hash function on the message and signature he receives. If the hashed signature has been tampered in any way, he will instantly know that by looking at the hash

A similar method can be used in cryptographic voting. Eligible voters can only cast at most 1 ballot, so the voting systems compute the hashed voter IDs for verification. Since a hash function is deterministic, if a hash appears a second time in the queue, the vote will be denied.

Acknowledgements

The author would like to thank Euler Circle director Simon Rubinstein-Salzedo and teaching assistant Ophir Horovitz for extensive help with the project.

References

[BS23] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography.* 2023.