One-Way Encryption and Message Authentication

Johannes Mittmann

May 28, 2005

Abstract

In modern society the protection of the authenticity of information has become as important as the protection of its confidentiality. This means that there is a need for data origin authentication as well as for verification of data integrity. Hash functions are versatile cryptographic building blocks that are used in this context, but also in conjunction with digital signature schemes and many other applications such as password protection or pseudo-random numbers generation.

A hash function is an algorithm that takes inputs of arbitrary length and returns a short string of bits, the message digest. However, for cryptographic hash functions to be secure, additional properties are required. For instance, it should be hard to find two distinct messages that hash to the same value. Hash functions that depend on a secret key are called message authentication codes (MACs).

This paper gives definitions of the basic terms of cryptographic hash functions, following the description in [Sti02]. First, we discuss generic attacks that can be applied to arbitrary hash functions and give a comparison of security criteria. Second, we describe design principles of iterated hash functions in general, and the Secure Hash Algorithm (SHA-1) in particular. Finally we introduce message authentication codes and show their construction from other cryptographic primitives.

Contents

1 Introduction

An illustrative example of a hash function is a function

$$h : \{0,1\}^* \to \{0,1\}^m$$

that maps a bitstring x of arbitrary length to a string of fixed length, the *message digest*. A *cryptographic hash function* can provide assurance of data integrity. This can be done by computing $y = h(x)$ and storing y in a secure place. If we want to check later wether the data has been altered, we can recompute the hash value and compare it with our stored one. If our data has changed, we hope that our message digest has changed, too.

A generalized definition for a whole set of hash functions is the following:

Definition 1. A *hash family* is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, where

1. $\mathcal{X}$ is a set of possible *messages*,
2. $\mathcal{Y}$ is a finite set of possible *messages digests* or *authentication tags*,
3. $\mathcal{K}$, the *keyspace*, is a finite set of possible *keys*,
4. for each $K \in \mathcal{K}$, there is a *hash function* $h_K \in \mathcal{H}$, $h_K : \mathcal{X} \to \mathcal{Y}$.

The set $\mathcal{X}$ can be finite or infinite. If it is finite, a hash function is called a *compression function*.

A pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is said to be *valid* under the key K if $h_K(x) = y$.

Let $N = |\mathcal{X}|$ and $M = |\mathcal{Y}|$. Then $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ is also termed an (N, M)-hash family. By $\mathcal{Y}^{\mathcal{X}}$ we denote the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$.

2 Security of Hash Functions

Let $h : \mathcal{X} \to \mathcal{Y}$ be an unkeyed hash function. It is considered secure, if the following three problems are hard to solve:

Problem 2. Preimage
Instance: A hash function $h : \mathcal{X} \to \mathcal{Y}$ and an element $y \in \mathcal{Y}$.
Find: $x \in \mathcal{X}$ such that $h(x) = y$.

A hash function for which Preimage cannot be solved efficiently is said to be *one-way* or *preimage resistant*.

Problem 3. Second Preimage
Instance: A hash function $h : \mathcal{X} \to \mathcal{Y}$ and an element $x \in \mathcal{X}$.
Find: $x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

A hash function for which Second Preimage cannot be solved efficiently is said to be *second preimage resistant*.

Problem 4. Collision
Instance: A hash function $h : \mathcal{X} \to \mathcal{Y}$.
Find: $x, x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

A hash function for which Collision cannot be solved efficiently is said to be *collision resistant*.

In the next two sections we investigate the difficulty of solving these problems, as well as the relative difficulty of the three problems (see [Sti04]).

2.1 Generic Attacks in the Random Oracle Model

Below we discuss generic attacks that can be applied to any hash function.

The *random oracle model*, as introduced by Bellare and Rogaway in [BR93], provides a theoretical model of an "ideal" hash function. In this model, a hash function $h : \mathcal{X} \to \mathcal{Y}$ is selected uniformly and independently from $\mathcal{Y}^{\mathcal{X}}$ at random. Further we are only permitted *oracle* access to the function. These assumptions are very strong, in fact, a random oracle cannot be implemented in practice. However, as a consequence we should have the following:

Theorem 5 (*independence* property)**.** *Let $h \in \mathcal{Y}^{\mathcal{X}}$ be chosen at random, and let $x_1, \ldots, x_\ell \in \mathcal{X}$. Suppose that the values $y_i = h(x_i)$ have been determined for $1 \geq i \geq \ell$. Then*

$$\Pr\left[h(x) = y \,|\, h(x_1) = y_1, \ldots, h(x_\ell) = y_\ell\right] = \frac{1}{M}$$

for all $x \in \mathcal{X} \setminus \{x_1, \ldots, x_\ell\}$ and all $y \in \mathcal{Y}$. □

The attacks we describe are *Las Vegas algorithms*. These are randomized algorithms that may fail to give an answer, but if they do return an answer, then the answer must be correct. By (ϵ, q)-*algorithm* we denote a Las Vegas algorithm with average-case success probability ϵ, in which at most q oracle queries are made.

The first algorithm attempts to solve Preimage.

Algorithm 6. FindPreimage(h, y, q)
1: choose $\mathcal{X}_0 \subseteq \mathcal{X}, \quad |\mathcal{X}_0| = q$
2: **for all** $x \in \mathcal{X}_0$ **do**
3: **if** $h(x) = y$ **then return** x
4: **end for**
5: **return** failure

Theorem 7. *For any $\mathcal{X}_0 \subseteq \mathcal{X}$ with $|\mathcal{X}_0| = q$, the average-case success probability of Algorithm 6 is*

$$\epsilon = 1 - \left(1 - \frac{1}{M}\right)^q.$$

Proof. Let $y \in \mathcal{Y}$ and $\mathcal{X}_0 = \{x_1, \ldots, x_q\}$. For $1 \geq i \geq q$, let E_i denote the event "$h(x_i) = y$". From Theorem 5 it follows that the E_i's are independent and $\Pr[E_i] = \frac{1}{M}$. Hence,

$$\Pr\left[E_1 \vee \cdots \vee E_q\right] = 1 - \left(1 - \frac{1}{M}\right)^q. \qquad \square$$

With a similar algorithm we try to solve Second Preimage.

Algorithm 8. FindSecondPreimage(h, x, q)
1: $y \leftarrow h(x)$
2: choose $\mathcal{X}_0 \subseteq \mathcal{X} \setminus \{x\}, \quad |\mathcal{X}_0| = q - 1$
3: **for all** $x_0 \in \mathcal{X}_0$ **do**
4: **if** $h(x_0) = y$ **then return** x_0
5: **end for**
6: **return** failure

The analysis of Algorithm 8 is the same as the previous one, except for an additional application of h.

Theorem 9. *For any $\mathcal{X}_0 \subseteq \mathcal{X} \setminus \{x\}$ with $|\mathcal{X}_0| = q - 1$, the success probability of Algorithm 8 is*

$$\epsilon = 1 - \left(1 - \frac{1}{M}\right)^{q-1}.$$

□

The attacks just described are called *random (second) preimage attacks*. If the number q of oracle queries is small compared to M we can estimate

$$1 - \epsilon = \left(1 - \frac{1}{M}\right)^{q} = \sum_{i=0}^{q} \binom{q}{i} \left(-\frac{1}{M}\right)^{i} \approx 1 - \frac{q}{M},$$

and hence $q \approx \epsilon M$. So these attacks are $(\epsilon, \mathcal{O}(M))$-algorithms.

Finally, we present an algorithm for Collision.

Algorithm 10. FindCollision(h, q)
1: choose $\mathcal{X}_0 \subseteq \mathcal{X}, \quad |\mathcal{X}_0| = q$
2: **for all** $x \in \mathcal{X}_0$ **do** $y_x \leftarrow h(x)$
3: **if** $y_x = y_{x'}$ for some $x' \neq x$ **then return** (x, x')
4: **else return** failure

The test in line 3 could be implemented by sorting the $y'_x s$, which can be done in $\mathcal{O}(q \log q)$.

The analysis of algorithm 10 is analogous to the *birthday paradox*.

Theorem 11. *For any $\mathcal{X}_0 \subseteq \mathcal{X}$ with $|\mathcal{X}_0| = q$, the success probability of Algorithm 10 is*

$$\epsilon = 1 - \prod_{i=1}^{q-1} \left(1 - \frac{i}{M}\right).$$

Proof. Let $\mathcal{X}_0 = \{x_1, \ldots, x_q\}$. For $1 \leq i \leq q$, let E_i denote the event "$h(x_i) \notin \{h(x_1), \ldots, h(x_{i-1})\}$". From Theorem 5 it follows by induction that $\Pr[E_1] = 1$ and

$$\Pr\left[E_i \mid E_1 \wedge \cdots \wedge E_{i-1}\right] = \frac{M - i + 1}{M}$$

for $2 \leq i \leq q$. Hence,

$$\Pr\left[E_1 \wedge \cdots \wedge E_q\right] = \prod_{i=1}^{q-1} \left(1 - \frac{i}{M}\right). \qquad \square$$

The attack carried out by this algorithms is called a *birthday attack*. Using the estimate

$$\begin{aligned} 1 - \epsilon = \prod_{i=1}^{q-1} \left(1 - \frac{i}{M}\right) &\approx \prod_{i=1}^{q-1} \exp\left(-\frac{i}{M}\right) = \exp\left(-\sum_{i=1}^{q-1} \frac{i}{M}\right) \\ &= \exp\left(-\frac{q(q-1)}{2M}\right) \approx \exp\left(-\frac{q^2}{2M}\right) \end{aligned}$$

we can approximate the number of oracle queries by

$$q \approx \sqrt{2M \log \frac{1}{1-\epsilon}}.$$

So a birthday attack is an $(\epsilon, \mathcal{O}(\sqrt{M}))$-algorithm.

In the setting of the standard birthday paradox ($\epsilon = 0.5$ and $M = 365$) we obtain $q \approx 1.17\sqrt{M} \approx 22.3$. Hence, the probability that two persons in a group of 23 people share a birthday is larger than $1/2$.

Since a birthday attack can be carried out on any hash function, it imposes a lower bound on the output size of a secure hash function. At the moment the minimum acceptable size of a message digest is 128 bits, but 160-bit message digests or larger are usually recommended (the birthday attack will require more than 2^{80} hashes in this case).

The algorithms presented in this section were rather trivial. However, in [Sti04] it is shown that they are optimal in the random oracle model.

2.2 Comparison of Security Criteria

In the last section we have seen that solving Collision is easier than solving (Second) Preimage. We now investigate wether there exist reductions among the three problems.

It is easy to see that we can reduce Collision to Second Preimage.

Algorithm 12. CollisionTo2ndPreimage(h)
1: choose $x \in \mathcal{X}$ uniformly at random
2: **if** Oracle2ndPreimage$(h, x) = x'$ **and** $x' \neq x$ **and** $h(x') = h(x)$ **then return** (x, x')
3: **else return** failure

If Oracle2ndPreimage is an (ϵ, q)-algorithm that solves Second Preimage for a hash function h, then it is clear that CollisionTo2ndPreimage is an $(\epsilon, q+2)$-algorithm that solves Collision for h. As a consequence we obtain that the property of collision resistance implies the property of second preimage resistance.

The more interesting question is wether Collision can be reduced to Preimage as well. Unfortunately, there is no positive answer in general. However, if we can solve Preimage with probability 1 and if we make fairly weak assumptions on the relative size of domain and range of the hash function, then we can solve Collision using Algorithm 13. This means that in this special case collision resistance implies preimage resistance.

Algorithm 13. CollisionToPreimage(h)
Require: OraclePreimage is $(1, q)$-algorithm
1: choose $x \in \mathcal{X}$ uniformly at random
2: $y \leftarrow h(x)$
3: **if** OraclePreimage$(h, y) = x'$ **and** $x' \neq x$ **then return** (x, x')
4: **else return** failure

Theorem 14. *Let $h : \mathcal{X} \to \mathcal{Y}$ be a compression function, where $|\mathcal{X}| \geq 2\,|\mathcal{Y}|$. Suppose* OraclePreimage *is a $(1, q)$-algorithm that solves Preimage for h. Then* CollisionToPreimage *is a $(1/2, q+1)$-algorithm for Collision, for the fixed compression function h.*

Proof. The relation defined by

$$x \sim x' \iff h(x) = h(x')$$

is an equivalence relation on $\mathcal{X}$. Let $\mathcal{C} := \mathcal{X}/\sim = \{[x] : x \in \mathcal{X}\}$ be the set of equivalence classes. Each equivalence class $[x]$ consists of the inverse image of an element in $\mathcal{Y}$, so we have $|\mathcal{C}| = |\mathcal{Y}|$.

Let $x \in \mathcal{X}$ be the randomly chosen element in CollisionToPreimage. The probability that OraclePreimage finds a different preimage that yields a collision is $(|[x]| - 1)/|[x]|$. Thus the average-case success probability of Algorithm 13 is

$$\begin{aligned}
\Pr[\text{success}] &= \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \frac{|[x]| - 1}{|[x]|} = \frac{1}{|\mathcal{X}|} \sum_{C \in \mathcal{C}} \sum_{x \in C} \frac{|C| - 1}{|C|} \\
&= \frac{1}{|\mathcal{X}|} \sum_{C \in \mathcal{C}} (|C| - 1) = \frac{|\mathcal{X}| - |\mathcal{Y}|}{|\mathcal{X}|} \\
&\geq \frac{|\mathcal{X}| - |\mathcal{X}|/2}{|\mathcal{X}|} = \frac{1}{2}.
\end{aligned}$$

□

3 Iterated Hash Functions

A hash function must be able to process arbitrary-length input. Since it is not easy to find such a function rule, *iterated hash functions* are used in practice. They break the input up into a series of equal-size blocks and operate on them in sequence using a compression function.

So let Compress : $\{0,1\}^{m+t} \to \{0,1\}^m$ be a compression function (where $t \geq 1$). The construction of the iterated hash function h consists of three steps:

preprocessing: Given an input string x, where $|x| \geq m + t + 1$, construct a string y such that $|y| = 0 \pmod{t}$. Denote

$$y = y_1 \,\|\, y_2 \,\|\, \cdots \,\|\, y_r,$$

where $|y_i| = t$ for $1 \leq i \leq r$.

A commonly used preprocessing step is to construct y by appending additional bits to x using a *padding function*:

$$y = x \,\|\, \text{Pad}(x).$$

The preprocessing step must ensure that the mapping $x \mapsto y(x)$ is an injection, otherwise it may be possible to find collisions for h easily. In particular this means that $|y| = rt \geq |x|$.

processing: Let IV be a public initial value ($|\text{IV}| = m$). Then compute the following:

$$\begin{aligned}
z_0 &\leftarrow \text{IV} \\
z_1 &\leftarrow \text{Compress}(z_0 \,\|\, y_1) \\
z_2 &\leftarrow \text{Compress}(z_1 \,\|\, y_2) \\
&\vdots \\
z_r &\leftarrow \text{Compress}(z_{r-1} \,\|\, y_r).
\end{aligned}$$

optional output transformation: Let $g : \{0,1\}^m \to \{0,1\}^\ell$ be a function. Define $h(x) = g(z_r)$. What we obtain is a hash function

$$h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \to \{0,1\}^\ell.$$

3.1 The Merkle-Damgård Construction

There are two elements in the generic construction described above, which have an important influence on the security of the resulting hash function: the choice of the padding rule and the initial value IV. Merkle [Mer90] and Damgård [Dam90] independently showed that an iterated hash function preserves the desired collision-resistance of the underlying compression function, if the padding bits contain the binary representation of the input length $|x|$ and if the IV is fixed. This padding scheme is called *MD-strengthening*.

Algorithm 15 shows the details of this construction for a block length $t \geq 2$.

Algorithm 15. MERKLE-DAMGÅRD(x)
Require: COMPRESS : $\{0,1\}^{m+t} \to \{0,1\}^m$, $\quad t \geq 2$
1: $n \leftarrow |x|$
2: $k \leftarrow \lceil n/(t-1) \rceil$
3: $d \leftarrow n - k(t-1)$
4: **for** $i \leftarrow 1$ **to** $k-1$ **do** $y_i \leftarrow x_i$
5: $y_k \leftarrow x_k \,\|\, 0^d$
6: $y_{k+1} \leftarrow$ the binary representation of d
7: $z_1 \leftarrow 0^{m+1} \,\|\, y_1$
8: $g_1 \leftarrow$ COMPRESS(z_1)
9: **for** $i \leftarrow 1$ **to** k **do**
10: $\quad z_{i+1} \leftarrow g_i \,\|\, 1 \,\|\, y_{i+1}$
11: $\quad g_{i+1} \leftarrow$ COMPRESS(z_{i+1})
12: **end for**
13: **return** g_{k+1}

In the case $t = 1$ the input x has to be encoded first. This is done using the function f defined by

$$f(0) = 0$$
$$f(1) = 01.$$

Algorithm 16. MERKLE-DAMGÅRD2(x)
Require: COMPRESS : $\{0,1\}^{m+1} \to \{0,1\}^m$
1: $n \leftarrow |x|$
2: $y \leftarrow 11 \,\|\, f(x_1) \,\|\, f(x_2) \,\|\, \cdots \,\|\, f(x_n)$
3: denote $y = y_1 \,\|\, y_2 \,\|\, y_2 \,\|\, \cdots \,\|\, y_k$, where $y_i \in \{0,1\},\ 1 \leq i \leq k$
4: $g_1 \leftarrow$ COMPRESS($0^m \,\|\, y_1$)
5: **for** $i \leftarrow 1$ **to** $k-1$ **do** $g_{i+1} \leftarrow$ COMPRESS($g_i \,\|\, y_{i+1}$)
6: **return** g_k

As a result of these algorithms we obtain the following theorem:

Theorem 17 (Merkle-Damgård)**.** *Suppose* Compress : $\{0,1\}^{m+t} \to \{0,1\}^m$ *is a collision resistant compression function, where* $t \geq 1$*. Then there exists a collision resistant hash function*

$$h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \to \{0,1\}^m.$$

The number of times Compress *is computed in the evaluation of* h *is at most*

$$\begin{array}{ll} 1 + \left\lceil \frac{n}{t-1} \right\rceil & \text{if } t \geq 2, \\ 2n + 2 & \text{if } t = 1, \end{array}$$

where $|x| = n$*.*

Proof. The proof works by contraposition. Suppose that we can find $x \neq x'$ such that $h(x) = h(x')$. We will show how we can find a collision for Compress in polynomial time for the case that $t \geq 2$ and $|x| \neq |x'| \pmod{t-1}$.

Denote

$$\begin{aligned} y(x) &= y_1 \,\|\, y_2 \,\|\, \cdots \,\|\, y_{k+1} \quad \text{and} \\ y(x') &= y'_1 \,\|\, y'_2 \,\|\, \cdots \,\|\, y'_{\ell+1}. \end{aligned}$$

In the case $|x| \neq |x'| \pmod{t-1}$ we have $y_{k+1} \neq y'_{\ell+1}$. Hence,

$$\begin{aligned} \textsc{Compress}(g_k \,\|\, 1 \,\|\, y_{k+1}) &= g_{k+1} = h(x) = h(x') = g'_{\ell+1} \\ &= \textsc{Compress}(g'_\ell \,\|\, 1 \,\|\, y'_{\ell+1}), \end{aligned}$$

which is a collision for Compress, because $y_{k+1} \neq y'_{\ell+1}$.

The remaining cases are proved similarly. □

3.2 The Secure Hash Algorithm (SHA-1)

In this section we give a description of the *secure hash algorithm SHA-1*. This hash function was designed by NIST and NSA, modeled closely after the MD4 message digest algorithm by Rivest, and adopted as FIPS 180-1 standard.

SHA-1 produces an 160-bit message digest and is based on 32-bit word operations:

AB	bitwise AND of A and B
$A \vee B$	bitwise OR of A and B
$A \oplus B$	bitwise XOR of A and B
$\neg A$	bitwise complement of A
$A + B$	integer addition modulo 2^{32}
$A \ll s$	circular left shift of A by s positions ($0 \leq s \leq 31$).

The padding scheme of SHA-1 uses MD-strengthening (see Figure 1) in order to make the length of the input a multiple of 512 bits:

Algorithm 18. SHA-1-Pad(x)
Require: $|x| \leq 2^{64} - 1$
Ensure: $|y| = 0 \pmod{512}$
1: $d \leftarrow (447 - |x|) \mod 512$

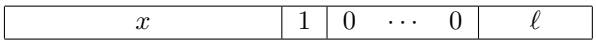

Figure 1: MD-strengthening

2: $\ell \leftarrow$ the binary representation of $|x|$, where $|\ell| = 64$
3: **return** $y \leftarrow x \,\|\, 1 \,\|\, 0^d \,\|\, \ell$

The main algorithm divides the padded input string into 512-bit blocks. The compression function maps $160 + 512$ bits to 160 bits and processes on these blocks. It consists of 80 rounds and uses the round dependent functions

$$f_t(B,C,D) = \begin{cases} BC \vee (\neg B)D, & \text{if } \ 0 \le t \le 19, \\ B \oplus C \oplus D, & \text{if } 20 \le t \le 39, \\ BC \vee BD \vee CD, & \text{if } 40 \le t \le 59, \\ B \oplus C \oplus D, & \text{if } 60 \le t \le 79, \end{cases}$$

and round dependent word constants

$$K_t = \begin{cases} \texttt{5A827999}, & \text{if } \ 0 \le t \le 19, \\ \texttt{6ED9EBA1}, & \text{if } 20 \le t \le 39, \\ \texttt{8F1BBCDC}, & \text{if } 40 \le t \le 59, \\ \texttt{CA62C1D6}, & \text{if } 60 \le t \le 79. \end{cases}$$

The details of SHA-1 are given below:

Cryptosystem 19. SHA-1(x)
1: $y \leftarrow$ SHA-1-PAD(x)
2: denote $y = M_1 \,\|\, M_2 \,\|\, \cdots \,\|\, M_n$, where each M_i is a 512-bit block
3: $H_0 \leftarrow \texttt{67452301}$
4: $H_1 \leftarrow \texttt{EFCDAB89}$
5: $H_2 \leftarrow \texttt{98BADCFE}$
6: $H_3 \leftarrow \texttt{10325476}$
7: $H_4 \leftarrow \texttt{C3D2E1F0}$
8: **for** $i \leftarrow 1$ **to** n **do**
9: denote $M_i = W_0 \,\|\, W_1 \,\|\, \cdots \,\|\, W_{15}$, where each W_i is a word
10: **for** $t \leftarrow 16$ **to** 79 **do** $W_t \leftarrow (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1$
11: $A \leftarrow H_0$
12: $B \leftarrow H_1$
13: $C \leftarrow H_2$
14: $D \leftarrow H_3$
15: $E \leftarrow H_4$
16: **for** $t \leftarrow 0$ **to** 79 **do**
17: $temp \leftarrow (A \lll 5) + f_t(B,C,D) + E + W_t + K_t$
18: $E \leftarrow D$
19: $D \leftarrow C$
20: $C \leftarrow B \lll 30$
21: $B \leftarrow A$
22: $A \leftarrow temp$
23: **end for**

24: $H_0 \leftarrow H_0 + A$
25: $H_1 \leftarrow H_1 + B$
26: $H_2 \leftarrow H_2 + C$
27: $H_3 \leftarrow H_3 + D$
28: $H_4 \leftarrow H_4 + E$
29: **end for**
30: **return** $H_0 \,\|\, H_1 \,\|\, H_2 \,\|\, H_3 \,\|\, H_4$

The Chinese researchers Wang, Yin and Yu showed that collision for SHA-1 can be found with less than 2^{69} hash operations. Since a birthday attack would require about 2^{80} hash operations, SHA-1 is broken (from the academic cryptographer's point of view).

4 Message Authentication Codes (MACs)

Related to hash functions are *message authentication codes* (MACs), which are keyed hash functions h_K satisfying certain security properties. They are used to provide data origin authentication. In this scheme, Alice and Bob share a secret key K that determines the hash function h_K. A message x can then together with its MAC $h_K(x)$ be transmitted over an insecure channel. When Bob receives the pair (x, y), he can verify if $y = h_K(x)$. Someone who does not know the secret key K should be unable to create a valid MAC.

So for a MAC algorithm to be secure, the following problem should be hard to solve:

Problem 20. (Existential) Forgery
Instance: Valid pairs $(x_1, y_1), \ldots, (x_q, y_q)$ under unknown key K.
Find: A valid pair (x, y) such that $x \notin \{x_1, \ldots, x_q\}$.

By (ϵ, q)-forger we denote a forgery with worse-case success probability ϵ.

Below we give two examples of MACs constructed from other cryptographic primitives: HMAC and CBC-MAC.

4.1 Nested MACs, HMAC and CBC-MAC

A *nested MAC* is a composition of two hash families. Let $(\mathcal{X}, \mathcal{Y}, \mathcal{L}, \mathcal{H})$ and $(\mathcal{Y}, \mathcal{Z}, \mathcal{K}, \mathcal{G})$ be hash families where $|\mathcal{X}| > |\mathcal{Y}| \geq |\mathcal{Z}|$. The *composition* of these hash families is the hash family $(\mathcal{X}, \mathcal{Z}, \mathcal{M}, \mathcal{G} \circ \mathcal{H})$ in which $\mathcal{M} = \mathcal{K} \times \mathcal{L}$ and

$$\mathcal{G} \circ \mathcal{H} = \left\{ (g \circ h)_{(K,L)} : \; g_K \in \mathcal{G}, h_L \in \mathcal{H} \right\},$$

where $(g \circ h)_{(K,L)}(x) = g_K(h_L(x))$ for all $x \in \mathcal{X}$.

The following theorem shows that a nested MAC is secure, assuming that the two hash families from which it is constructed satisfy appropriate security requirements:

Theorem 21. *Let $(\mathcal{X}, \mathcal{Z}, \mathcal{M}, \mathcal{G} \circ \mathcal{H})$ be a nested MAC. Suppose there does not exist an $(\epsilon_1, q+1)$-collision attack for a randomly chosen function $h_L \in \mathcal{H}$, where L is secret, and there does not exist an (ϵ_2, q)-forger for a randomly chosen function $g_K \in \mathcal{G}$, where K is secret. Further suppose there exists an (ϵ, q)-forger for the nested MAC $(g \circ h)_{(K,L)} \in \mathcal{G} \circ \mathcal{H}$. Then*

$$\epsilon \leq \epsilon_1 + \epsilon_2.$$

HMAC is an example of a nested MAC algorithm that constructs a MAC from an (unkeyed) hash function (see [BCK96]). We describe a version that is based on SHA-1 and uses a 512-bit key K. Define the hexadecimal constants

$$\begin{aligned} ipad &= \texttt{3636}\cdots\texttt{36}, \\ opad &= \texttt{5C5C}\cdots\texttt{5C}. \end{aligned} \qquad \text{(512-bit)}$$

Then the 160-bit MAC is defined as follows:

Cryptosystem 22. HMAC(x, K)

$$\text{HMAC}_K(x) = \text{SHA-1}((K \oplus opad) \,\|\, \text{SHA-1}((K \oplus ipad) \,\|\, x))$$

Note that the outer computation of SHA-1 requires only one application of the compression function. If we assume, that SHA-1 used in this way is secure as a MAC , and if we furthermore assume that the inner application of SHA-1 is collision resistant, then Theorem 21 tells us that HMAC is secure as a MAC.

Another popular way to construct a MAC is to use a block cipher in CBC mode with a fixed initialization vector. Let $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be an endomorphic cryptosystem, where $\mathcal{P} = \mathcal{C} = \{0,1\}^t$. Then CBC-MAC is defined as follows:

Cryptosystem 23. CBC-MAC(x, K)

denote $x = x_1 \,\|\, \cdots \,\|\, x_n, \quad |x_i| = t$
$y_0 \leftarrow 00\cdots0$
for $i \leftarrow 1$ **to** n **do** $y_i \leftarrow e_K(y_{i-1} \oplus x_i)$
return y_n

CBC-MAC is secure under certain assumptions about the randomness of the underlying encryption scheme.

4.2 Unconditionally Secure MACs

In this section we will discuss MACs, that are unconditionally secure. For this we will assume that a key is used to produce only one authentication tag. So an adversary is able to make at most one query before he outputs a possible forgery. Stated another way, we will construct MACs for which we can prove the non-existence of an $(\epsilon, 0)$-forger (*impersonation*) and an $(\epsilon, 1)$-forger (*substitution*), for appropriate values of ϵ, even if the adversary possesses infinite computing power.

For $q = 0, 1$, we define the *deception probability* Pd_q to be the maximum value of ϵ such that an (ϵ, q)-forger exists. Let K_0 be the key chosen by Alice and Bob.

For $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ we define payoff(x, y) to be the probability that (x, y) is a valid pair. This means

$$\text{payoff}(x,y) = \Pr\left[y = h_{K_0}(x)\right] = \frac{|\{K \in \mathcal{K} :\ h_K(x) = y\}|}{|\mathcal{K}|},$$

and hence

$$Pd_0 = \max\left\{\text{payoff}(x,y) :\ x \in \mathcal{X},\, y \in \mathcal{Y}\right\}.$$

Now let $(x, y) \in \mathcal{X} \times \mathcal{Y}$ be a valid pair. For $x' \in \mathcal{X}$ with $x \neq x'$ and $y' \in \mathcal{Y}$ we define $\text{payoff}(x', y'; x, y)$ to be the probability that (x', y') is a valid pair, given that (x, y) is a valid pair. Then we can compute

$$\begin{aligned}\text{payoff}(x', y'; x, y) &= \Pr\left[y' = h_{K_0}(x') \,|\, y = h_{K_0}(x)\right] \\ &= \frac{\Pr\left[y' = h_{K_0}(x') \wedge y = h_{K_0}(x)\right]}{\Pr\left[y = h_{K_0}(x)\right]} \\ &= \frac{|\{K \in \mathcal{K} :\ h_K(x') = y', h_K(x) = y\}|}{|\{K \in \mathcal{K} :\ h_K(x) = y\}|}.\end{aligned}$$

If we denote by

$$\mathcal{V} = \{(x, y) :\ |\{K \in \mathcal{K} :\ h_K(x) = y\}| \geq 1\}$$

the set of pairs that are valid under at least one key, then we obtain

$$Pd_1 = \max\left\{\text{payoff}(x', y'; x, y) :\ x \in \mathcal{X},\ y, y' \in \mathcal{Y},\ (x, y) \in \mathcal{V},\ x \neq x'\right\}.$$

We now define a class of hash families that immediately yield authentication codes in which Pd_0 and Pd_1 can easily be computed.

Definition 24. Let $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ be an (N, M)-hash family. This hash family is *strongly universal*, if

$$|\{K \in \mathcal{K} :\ h_K(x) = y,\ h_K(x') = y'\}| = \frac{|\mathcal{K}|}{M^2}$$

for all $x, x' \in \mathcal{X}$ such that $x \neq x'$, and for all $y, y' \in \mathcal{Y}$.

Lemma 25. *Let $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ be a strongly universal (N, M)-hash family. Then*

$$|\{K \in \mathcal{K} :\ h_K(x) = y\}| = \frac{|\mathcal{K}|}{M} \qquad \forall x \in \mathcal{X} \quad \forall y \in \mathcal{Y}.$$

Proof. Let $x, x' \in \mathcal{X}$ such that $x \neq x'$, and let $y \in \mathcal{Y}$. Then

$$\begin{aligned}|\{K \in \mathcal{K} :\ h_K(x) = y\}| &= \sum_{y' \in \mathcal{Y}} |\{K \in \mathcal{K} :\ h_K(x) = y,\ h_K(x') = y'\}| \\ &= \sum_{y' \in \mathcal{Y}} \frac{|\mathcal{K}|}{M^2} = \frac{|\mathcal{K}|}{M}.\end{aligned}$$

□

Theorem 26. *Let $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ be a strongly universal (N, M)-hash family. Then $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ is an authentication code with*

$$Pd_0 = Pd_1 = \frac{1}{M}.$$

Proof. Let $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. Then

$$\text{payoff}(x, y) = \frac{|\{K \in \mathcal{K} :\ h_K(x) = y\}|}{|\mathcal{K}|} = \frac{|\mathcal{K}|/M}{|\mathcal{K}|} = \frac{1}{M}.$$

Now let $x, x' \in \mathcal{X}$ such that $x \neq x'$, and let $y, y' \in \mathcal{Y}$, where $(x, y) \in \mathcal{V}$. Then

$$\begin{aligned}\text{payoff}(x', y'; x, y) &= \frac{|\{K \in \mathcal{K} :\ h_K(x') = y', h_K(x) = y\}|}{|\{K \in \mathcal{K} :\ h_K(x) = y\}|} \\ &= \frac{|\mathcal{K}|/M^2}{|\mathcal{K}|/M} = \frac{1}{M}.\end{aligned}$$

Therefore $Pd_0 = Pd_1 = \frac{1}{M}$. □

In [Sti02] it is shown that the converse of this theorem is also true and that these deception probabilities are optimal.

Here is an simple example of a strongly universal hash family:

Example. Let p be prime. For $a, b \in \mathbb{Z}_p$ define $f_{(a,b)} : \mathbb{Z}_p \to \mathbb{Z}_p$ by

$$f_{(a,b)}(x) = ax + b \pmod{p}.$$

In order to prove that $(\mathbb{Z}_p, \mathbb{Z}_p, \mathbb{Z}_p \times \mathbb{Z}_p, \{f_{(a,b)} : a, b \in \mathbb{Z}_p\})$ is a strongly universal (p, p)-hash family, we have to show that for $x, x', y, y' \in \mathbb{Z}_p$, where $x \neq x'$, there is a unique key $(a, b) \in \mathbb{Z}_p \times \mathbb{Z}_p$ such that $ax + b = y \pmod{p}$ and $ax' + b = y' \pmod{p}$. But this follows from the fact that (a, b) is the solution of a linear system of equations over the field $\mathbb{Z}_p$ with determinant

$$\det \begin{pmatrix} x & 1 \\ x' & 1 \end{pmatrix} = x - x' \neq 0.$$

References

[BCK96] M. Bellare, R. Canetti and R. Krawczyk, "Keying hash functions for message authentication", in *Lecture Notes in Computer Science*, **1109**, Springer-Verlag, 1996.

[BR93] M. Bellare and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols", in *ACM Conference on Computer and Communications Security*, ACM Press, 1993.

[Dam90] I. B. Damgård, "A design principle for hash functions.", in *Lecture Notes in Computer Science*, **435**, Springer-Verlag, 1990.

[Mer90] R. C. Merkle, "One-way hash functions and DES.", in *Lecture Notes in Computer Science*, **435**, Springer-Verlag, 1990.

[Sti02] D. R. Stinson, *Cryptography: Theory and Practice*, 2nd ed., Chapman & Hall/CRC, 2002.

[Sti04] D. R. Stinson, "Some observations on the theory of cryptographic hash functions", preprint to appear in *Designs, Codes and Cryptography*, 2004.