Introduction to Hash Functions

Sugata Gangopadhyay

Indian Institute of Technology Roorkee

Hash functions

- A cryptographic hash function provides assurance of data integrity.
- A hash function is used to construct a short “fingerprint” of some data.
- Even if the data is stored in an insecure place, its integrity can be checked from time to time by recomputing the fingerprint and verifying that the fingerprint has not changed.

Hash functions: fingerprinting data, message digest

- Let h be a hash function and x be some data.
- Usually, x is a binary string of arbitrary length.
- The corresponding fingerprint $y = h(x)$ is said to be a *message digest*.
- A message digest would typically be a fairly short binary string; 160 bits is a common choice.

Hash functions: fingerprinting data, message digest

- Suppose that y is stored in a secure place, but x is not.
- Suppose x is changed to x', say.
- The fact that x has been altered can be detected by computing $y' = h(x')$ and verifying that $y' \neq y$.

Keyed hash functions: message authentication code, MAC

- Suppose that Alice and Bob share a secret key K which determines a hash function, say h_K.
- For a message x, the corresponding authentication tag $y = h_K(x)$, can be computed by Alice and Bob.
- The pair (x, y) can be transmitted over an insecure channel from Alice to Bob.
- When Bob receives the pair (x, y), he can verify if $y = h_K(x)$.
- If this condition is satisfied, he is confident that neither x nor y was altered by an adversary, provided that the hash family is "secure".
- In particular, Bob is assured that the message x originates from Alice.

Hash functions

- A hash family is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, where following conditions are satisfied:

  1. $\mathcal{X}$ is a set of possible *messages*
  2. $\mathcal{Y}$ is a finite set of possible *message 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* $h_K : \mathcal{X} \to \mathcal{Y}$.

Notation and terminology

- In the definition of a hash function, $\mathcal{X}$ could be a finite or infinite set; $\mathcal{Y}$ is always a finite set.
- If $\mathcal{X}$ is a finite set, a hash function is sometimes called a *compression function*, and we always assume that $|\mathcal{X}| \geq |\mathcal{Y}|$ |.
- It is customary to assume that $|\mathcal{X}| \geq 2|\mathcal{Y}|$.
- A pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is said to be a *valid pair* under the key K if $h_K(x) = y$.
- One aim of the study of hash functions is to develop methods that resist creation of valid pairs by adversaries.

Notation and terminology

- Let $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$ denote the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$.
- $|\mathcal{X}| = N, |\mathcal{Y}| = M$.
- The number of all possible hash functions from $\mathcal{X}$ to $\mathcal{Y}$ is $\left|\mathcal{F}^{\mathcal{X},\mathcal{Y}}\right| = M^N$.
- Any hash family $\mathcal{F} \subseteq \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is termed as an (N, M)-*hash family*.
- An unkeyed function is a function $h : \mathcal{X} \rightarrow \mathcal{Y}$ such that $|\mathcal{K}| = 1$

Security of Hash Functions

- Suppose that $h : \mathcal{X} \to \mathcal{Y}$ is an unkeyed hash function. Let $x \in \mathcal{X}$, and define $y = h(x)$.

- If a hash function is to be considered to be secure, it should be the case that the following three problems are difficult to solve.
  1. 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 $h(x) = y$.
  2. 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 $h(x) = h(x')$.
  3. Collision:
     - Instance: A hash function $h : \mathcal{X} \to \mathcal{Y}$.
     - Find: $x, x' \in \mathcal{X}$ such that $x \neq x'$ and $h(x) = h(x')$.

Random Oracle Model

- Random oracle model is an idealized model for a hash function which attempts to capture the concept of an "ideal" hash function.
- If a hash function h is well designed, it should be the case that the only efficient way to determine the value $h(x)$ for a given x is actually evaluate the function h at the value x.
- This should remain true even if many other values $h(x_1)$, $h(x_2)$, ... have already been calculated.

Random Oracle Model

- The random oracle model, which was introduced by Bellare and Rogaway, provides a mathematical model of an "ideal" hash function.
- In this model, a hash function $h : \mathcal{X} \rightarrow \mathcal{Y}$ is chosen randomly from $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$, and we are only permitted *oracle* access to the function h.
- This means that we are not given a formula or an algorithm to compute values of the function h, and the only way to compute a value $h(x)$ is to query the oracle.

The random oracle model

Theorem

Suppose that $h \in \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is chosen randomly, and let $\mathcal{X}_0 \subseteq \mathcal{X}$. Suppose that the values $h(x)$ have been determined (by querying an oracle h) if and only if $x \in \mathcal{X}_0$. The $\Pr[h(x) = y] = \frac{1}{M}$ for all $x \in \mathcal{X} \setminus \mathcal{X}_0$ and all $y \in \mathcal{Y}$.

Example where the random oracle model does not apply

$h : \mathbb{Z}_n \times \mathbb{Z}_n \rightarrow \mathbb{Z}_n, h(x, y) = ax + by \mod n$ $a, b \in \mathbb{Z}_n$ and $n \geq 2$ is a positive integer.
Suppose $h(x_1, y_1) = z_1$ and $h(x_2, y_2) = z_2$. Let $r, s \in \mathbb{Z}_n$. Then

$$\begin{aligned} & h(rx_1 + sx_2 \mod n, ry_1 + sy_2 \mod n) \\ &= a(rx_1 + sx_2) + b(ry_1 + sy_2) \mod n \\ &= r(ax_1 + by_1) + s(ax_2 + by_2) \mod n \\ &= rh(x_1, y_1) + sh(x_2, y_2). \end{aligned}$$

- Suppose we are told that a hash function in use is a linear function from $\mathbb{Z}_n \times \mathbb{Z}_n$ to $\mathbb{Z}_n$.
- Then given the hash values for any two points, we can determine the hash values at several other points without actually evaluating the hash function.
- This proves that the random oracle model does not hold for such a function.

Randomized algorithms

- A *Las Vegas algorithm* is a randomized algorithm which may fail to give an answer, but if the algorithm does return an answer, then the answer must be correct.
- Suppose $0 \leq \epsilon < 1$ is a real number. A randomized algorithm has worst-case success probability ϵ if the probability that the algorithm returns a correct answer, averaged over all problem instances of a specified size, is at least ϵ.
- (ϵ, Q)-algorithm denotes a Las Vegas algorithm with average-case success probability ϵ, in which the oracle queries made by the algorithms is at most Q.
- The success probability ϵ is the average over all possible random choices of $h \in \mathcal{F}^{\mathcal{X},\mathcal{Y}}$, and all possible random choices of $x \in \mathcal{X}$ or $y \in \mathcal{Y}$, if x and y are specified as part of the problem instance.

Find-Preimage

Input: h, y, Q;
choose any $\mathcal{X}_0 \subseteq \mathcal{X}, |\mathcal{X}_0| = Q$;
for $x \in \mathcal{X}_0$ **do**
    if $h(x) = y$ **then**
        return x
    end
    return *failure*
end

Algorithm 1: Find-Preimage

Find-Preimage

Theorem

For any $\mathcal{X}_0 \subseteq \mathcal{X}$ with $|\mathcal{X}_0| = Q$, the average-case success probability of Algorithm 1 is $\epsilon = 1 - (1 - \frac{1}{M})^Q$.

Proof.

Let $y \in \mathcal{Y}$, and $\mathcal{X}_0 = \{x_1, \ldots, x_Q\}$. Let E_i denote the event "$h(x_i) = y$. " $\Pr[E_i] = 1/M$ and $\Pr[E_i^c] = 1 - 1/M$.

$$\begin{aligned}\Pr[E_1 \vee E_2 \vee \cdots \vee E_Q] &= 1 - \Pr[E_1^c \wedge E_2^c \wedge \cdots \wedge E_Q^c] \\ &= 1 - \left(1 - \frac{1}{M}\right)^Q .\end{aligned}$$

Find-Second-Preimage

Input: h, x, Q;
$y \leftarrow h(x)$;
choose $\mathcal{X}_0 \subseteq \mathcal{X} \setminus \{x\}, |\mathcal{X}_0| = Q - 1$;
for $x' \in \mathcal{X}_0$ **do**
    if $h(x') = y$ **then**
        return x'
    end
    return *failure*
end

Algorithm 2: Find-Second-Preimage

Find-Second-Preimage

Theorem

For any $\mathcal{X}_0 \subseteq \mathcal{X}$ with $|\mathcal{X}_0| = Q - 1$, the average-case success probability of Algorithm 2 is $\epsilon = 1 - (1 - \frac{1}{M})^{Q-1}$.

Proof.

Let $y \in \mathcal{Y}$, and $\mathcal{X}_0 = \{x_1, \ldots, x_{Q-1}\}$. Let E_i denote the event "$h(x_i) = y$. " $\Pr[E_i] = 1/M$ and $\Pr[E_i^c] = 1 - 1/M$.

$$\begin{aligned}\Pr[E_1 \vee E_2 \vee \cdots \vee E_{Q-1}] &= 1 - \Pr[E_1^c \wedge E_2^c \wedge \cdots \wedge E_{Q-1}^c] \\ &= 1 - \left(1 - \frac{1}{M}\right)^{Q-1}.\end{aligned}$$

Find-Collision

Input: h, Q;
choose $\mathcal{X}_0 \subseteq \mathcal{X}, |\mathcal{X}_0| = Q$;
for $x \in \mathcal{X}_0$ **do**
  $y_x \leftarrow h(x)$;
end
if $y_x = y_{x'}$ *for some* $x \neq x'$ **then**
  return (x, x')
end
else
  return *failure*
end

Algorithm 3: Find-Collision

Find-Collision

Theorem

For any $\mathcal{X}_0 \subseteq \mathcal{X}$ with $|\mathcal{X}_0| = Q$, the success probability of Algorithm 3 is

$$\epsilon = 1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\cdots\left(\frac{M-Q+1}{M}\right)$$

Proof.

Let $\mathcal{X}_0 = \{x_1, x_2, \ldots, x_Q\}$. For $1 \leq i \leq Q$, let E_i denote the event $h(x_i) \notin \{h(x_1), h(x_2), \ldots, h(x_{i-1})\}$.

$$\Pr[E_i | E_1 \wedge E_2 \wedge \cdots \wedge E_{i-1}] = \frac{M-i+1}{M}.$$

Therefore

$$\Pr[E_1 \wedge E_2 \wedge \cdots \wedge E_Q] = \prod_{i=1}^{Q-1}\left(1 - \frac{i}{M}\right).$$

Find-Collision

$$\Pr[E_1 \wedge E_2 \wedge \cdots \wedge E_Q] = \prod_{i=1}^{Q-1} \left(1 - \frac{i}{M}\right)$$
$$= \prod_{i=1}^{Q-1} e^{\frac{-i}{M}} = e^{-\sum_{i=1}^{Q-1} \frac{i}{M}} = e^{\frac{-Q(Q-1)}{2M}}.$$

The probability of finding at least one collision is $\epsilon = 1 - e^{\frac{-Q(Q-1)}{2M}}$.
$e^{\frac{-Q(Q-1)}{2M}} \approx 1 - \epsilon; \frac{-Q(Q-1)}{2M} \approx \ln(1-\epsilon); Q^2 - Q \approx 2M \ln \frac{1}{1-\epsilon}$

$$Q \approx \sqrt{2M \ln \frac{1}{1-\epsilon}}$$

If we take $\epsilon = 0.5$, then our estimate is $Q \approx 1.17\sqrt{M}$.

Comparison of security criteria

- COLLISION-TO-SECOND-PREIMAGE Finding a collision is easier than finding the second-preimage. If we have an algorithm that solves the second-primage problem, then it can be used to find collision. This is sail to be a "reduction of the problem COLLISION to the problem SECOND-PREIMAGE.

COLLISION-TO-SECOND-PREIMAGE

Input: external ORACLE-2ND-PREIMAGE
choose $x \in \mathcal{X}$ uniformly at random;
if ORACLE-2ND-PREIMAGE$(h, x) = x'$ **then**
  return (x, x')
end
else
  return *failure*
end

Algorithm 4: Collision-To-Second-Preimage

COLLISION-TO-PREIMAGE

- COLLISION-TO-PREIMAGE Suppose that we have a $(1, Q)$ algorithm to solve preimage. The question is whether it can be used to solve collision. The following randomized algorithm performs that task. The is a "reduction of the problem COLLISION to the problem PREIMAGE.

COLLISION-TO-PREIMAGE

Input: external ORACLE-PREIMAGE
choose $x \in \mathcal{X}$ uniformly at random;
$y \leftarrow h(x)$;
if ORACLE-PREIMAGE$(h, y) = x'$ *and* $x \neq x'$ **then**
  return (x, x')
end
else
  return *failure*
end

Algorithm 5: Collision-To-Preimage

Collision-to-Preimage

- Let $x \sim x'$ if and only if $h(x) = h(x')$.
- $[x] = \{x' \in \mathcal{X} : x' \sim x\}$ is the equivalence class of x.
- Let the set of all equivalence classes be $\mathcal{C}$.
- For $x \in \mathcal{X}$, let $y = h(x)$. The probability that COLLISION-TO-PREIMAGE is successful is $\frac{\left|[x]\right|-1}{\left|[x]\right|}$.
- The average probability of success

$$\Pr[success] = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \frac{\left|[x]\right| - 1}{\left|[x]\right|} = \frac{1}{|\mathcal{X}|} \sum_{C \in \mathcal{C}} \sum_{x \in C} \frac{|C| - 1}{|C|}$$

$$= \frac{1}{|\mathcal{X}|} \sum_{C \in \mathcal{C}} (|C| - 1) = \frac{|\mathcal{X}| - |\mathcal{Y}|}{|\mathcal{X}|} = 1 - \frac{|\mathcal{Y}|}{|\mathcal{X}|} \geq \frac{1}{2}, \text{ if } |\mathcal{X}| \geq 2|\mathcal{Y}|.$$

Iterated Hash Functions

- We denote the length of a bitstring x by $|x|$.
- The concatenation of the bitstrings x and y is written as $x \| y$.
- Let **compress** : $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$.
- We use the compression function **compress** to construct a hash function

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

Iterated Hash Functions

Preprocessing step

Given an input string x, where $|x| \geq m + t + 1$, construct a string y, using a public algorithm, such that $|y| \equiv 0 \pmod{t}$. Denote $y = y_1 \| y_2 \| \cdots \| y_r$, where $|y_i| = t$ for $1 \leq i \leq r$.

Processing step

Let IV be a public initial value that is a bitstring of length m. Then compute the following:

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

Iterated Hash Functions

Output step

Let $g : \{0,1\}^m \to \{0,1\}^\ell$ be a public function. Define $h(x) = g(z_r)$.

Padding

- Padding function is a publicly disclosed function that is applied on x to produce $\mathrm{pad}(x)$.
- Typically $\mathbf{pad}(x)$ involves the length $|x|$ and additional zeros so that the length of $y = x \| \mathbf{pad}(x)$ is divisible by t.

Merkle-Damgård Construction

Suppose **compress** : $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$ is a collision resistant compression function, where $t \geq 1$. We will use **compress** to construct a collision resistant hash function $h : \mathcal{X} \rightarrow \{0,1\}^m$, where

$$\mathcal{X} = \cup_{i=m+t+1}^{\infty} \{0,1\}^i.$$

Case 1: $t \geq 2$

Suppose $x \in \mathcal{X}$, and $|x| = n \geq m + t + 1$. $k = \lceil \frac{n}{t-1} \rceil$ and $d = k(t-1) - n$. We can express x as the concatenation: $x = x_1 \| x_2 \| \cdots \| x_k$, where $|x_1| = |x_2| = \cdots = |x_{k-1}| = t - 1$ and $x_k = t - 1 - d$.
$y(x) = y_1 \| y_2 \| \cdots \| y_{k+1}$. y_k is formed from x_k by padding on the right with d zeroes, so that all the blocks y_i $(1 \leq i \leq k)$ are of length $t - 1$. y_{k+1} should be padded on the left with zeroes so that $|y_{k+1}| = t - 1$.

Merkle-Damgård Construction

external compress;
comment compress : $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$, where $t \geq 2$;
$n \leftarrow |x|$;
$k \leftarrow \lceil n/(t-1) \rceil$;
$d \leftarrow k(t-1) - n$;
for $i \leftarrow 1$ **to** $k-1$ **do**
  $y_i \leftarrow x_i$;
end
$y_k \leftarrow x_k \| 0^d$;
$y_{k+1} \leftarrow$ the binary representation of d;
$z_1 \leftarrow 0^{m+1} \| y_1$;
$g_1 \leftarrow \mathbf{compress}(z_1)$;
for $i \leftarrow 1$ **to** k **do**
  $z_{i+1} \leftarrow g_i \| 1 \| y_{i+1}$;
  $g_{i+1} \leftarrow \mathbf{compress}(z_{i+1})$;
end

Collision Resistance

Theorem

Suppose **compress** : $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$ *is a collision resistant compression function, where* $t \geq 2$. *Then the function*

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

as constructed in 6 is a collision resistant hash function.

Merkle-Damgård Construction

Merkle-Damgård(x) for $t = 1$

external compress;
comment compress : $\{0,1\}^{m+1} \rightarrow \{0,1\}^m$;
$n \leftarrow |x|$;
$y \leftarrow 11 \| f(x_1) \| f(x_2) \| \cdots \| f(x_n)$;
denote $y = y_1 \| y_2 \| \cdots \| y_k$, where $y_i \in \{0,1\}$, $1 \leq i \leq k$;
$g_1 \leftarrow \mathbf{compress}(0^m \| y_1)$;
for $i \leftarrow 1$ **to** $k-1$ **do**
  $g_{i+1} \leftarrow \mathbf{compress}(g_i \| y_{i+1})$;
end
return g_k;

Algorithm 7: MERKLE-DAMGÅRD(x)

- $|x| = n \geq m + 2$. $f(0) = 0$, $f(1) = 01$.

Collision Resistance $t = 1$

Theorem

Suppose **compress** : $\{0,1\}^{m+1} \rightarrow \{0,1\}^m$ *is a collision resistant compression function, where* $t \geq 2$. *Then the function*

$$h : \bigcup_{i=m+2}^{\infty} \{0,1\}^i \rightarrow \{0,1\}^m,$$

as constructed in 7 is a collision resistant hash function.

Some Examples of iterated hash functions

Hash functions constructed by using Merkle-Damgård approach:

- *MD4* was proposed by Rivest in 1990.
- *MD5* was proposed by Rivest in 1992.
- *SHA* was proposed as a standard by NIST in 1993, and published as FIPS 180-1. Now *SHA* is referred to as *SHA-0*.

Discovery of collisions:

- Collision in the compression function of *MD4* and *MD5* were discovered in mid-1990s.
- It was shown in 1998 that *SHA-0* has a weakness that would allow collision to be found in approximately 2^{61} steps that is much more efficient than a birthday attack, which requires 2^{80} steps.

Some Examples of iterated hash functions

List of further attacks:

- In CRYPTO-2004:
  - Collision for *SHA-0* was found by Joux.
  - Collision for *MD5* and several other popular hash functions were found by Wang, Lai, and Yu.
- The first collision for *SHA-1* was found by Stevens, Bursztein, Karpman, Albertini, and Markov and announced in 23 February 2017. This attack was approximately 100000 times faster than a brute-force "birthday paradox" search having roughly 2^{80} trials.
- *SHA-2* includes four hash functions known as *SHA-224*, *SHA-256*, *SHA-384*, and *SHA-512*.
- The last three of the above are approved as FIPS standard in 2002.

Operations used in *SHA-1*

$X \wedge Y$	bitwise "and" of X and Y
$X \vee Y$	bitwise "or" of X and Y
$X \oplus Y$	bitwise "x-or" of X and Y
$\neg X$	bitwise complement of X
$X + Y$	integer addition modulo 2^{32}
$\mathbf{ROTL}^{s}(X)$	circular left shift of X by s positions $(0 \leq s \leq 31)$

- These operations are very efficient.
- However, when a suitable sequence of these operations is performed, the output is quite unpredictable.

The Sponge Construction

- *SHA-3* is based on a design called the ***sponge construction***.
- This technique was developed by Bertoni, Daemen, Peeters, and Van Assche.
- Instead of using a compression function, the basic "building block" is a function f that maps bitstrings of a fixed length to bitstrings of the same length.
- Typically f will be a bijection, so every bitstring will have a unique preimage.

The Sponge Construction

- Suppose that f operates on bitstrings of length b. That is $f' : \{0,1\}^b \rightarrow \{0,1\}^b$. The integer b is call the ***width***.
- Write $b = r + c$, where r is the ***bitrate*** and c is the ***capacity***.
- The value of r affects the efficiency of the resulting sponge function, as a message will be processed r bits at a time.
- The value of c affects the resulting security of the sponge function.
- The security level against a certain kind of collision attack is intended to be roughly $2^{c/2}$. This is comparable to the security of a random oracle with a c-bit output.

The Sponge Construction

The sponge function based on f works as follows:

- The input message M is a bitstring of arbitrary length.
- M is padded appropriately so that its length is a multiple of r.
- Then the padded message is split into blocks of length r.

The Sponge Construction

The sponge function based on f works as follows:

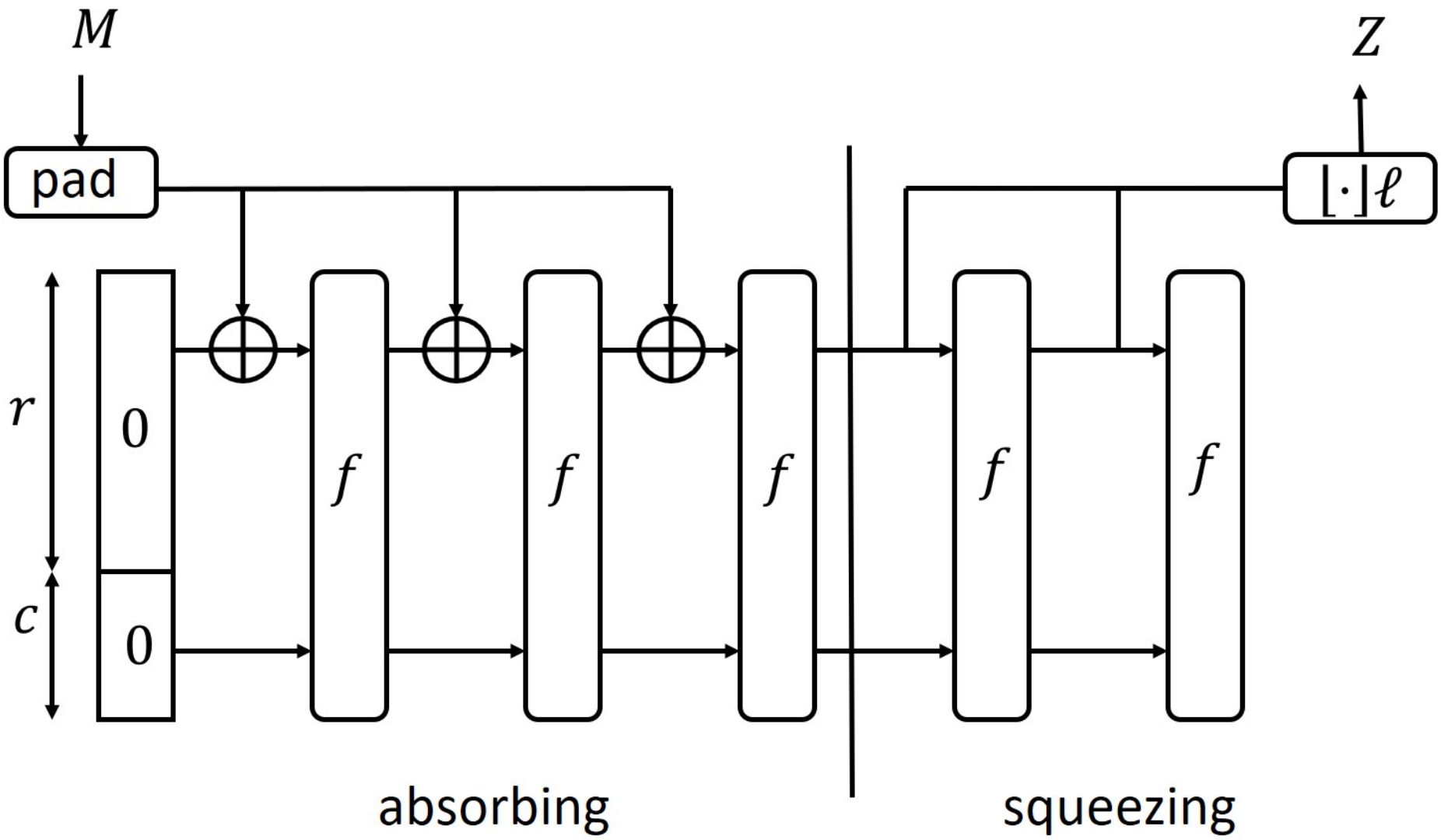
Absorbing phase

- Initially the *state* is a bitstring of length b consisting of zeroes.
- The first block of the padded message is exclusive-ored with the first r bits of the state. Then the function f is applied which updates the state.
- This process is repeated with the remaining blocks of the padded message.

Squeezing phase

- Suppose ℓ output bits are desired.
- Take the first r bits of the current state; this forms an output block.
- If $\ell > r$ we apply f to the current state and take the first r output bits as another output block.
- The process is repeated until we have a total of at least ℓ bits.

Diagram of the sponge construction



The Sponge Construction

- $M = m_1 \| \cdots \| m_k$, where $m_1, \ldots, m_k \in \{0,1\}^r$.
- Let the initial state be $x_0 \| y_0$ where $x_0 = \underbrace{00\ldots0}_{r} \in \{0,1\}^r$ and $y_0 = \underbrace{00\ldots0}_{c} \in \{0,1\}^c$, and let the state after the ith step be $x_i \| y_i$, where $x_i \in \{0,1\}^r$ and $y_i \in \{0,1\}^c$.
- The following equations describe the state transitions.

$$\begin{aligned} x_1 \| y_1 &= f(m_1 \oplus x_0 \| y_0) \\ x_2 \| y_2 &= f(m_2 \oplus x_1 \| y_1) \\ \vdots \quad & \quad \vdots \quad \quad \vdots \\ x_k \| y_k &= f(m_k \oplus x_{k-1} \| y_{k-1}). \end{aligned}$$

Generation of an internal collision

Suppose

$$
\begin{aligned}
x_1\|y_1 &= f(x_0\|y_0)\\
x_2\|y_2 &= f(x_0\|y_1)\\
&\vdots \quad \vdots \quad \vdots\\
x_k\|y_k &= f(x_0\|y_{k-1}).
\end{aligned}
$$

$$
\begin{aligned}
M &= x_0\|x_1\|\cdots\|x_h\\
M' &= x_0\|x_1\|\cdots\|x_k
\end{aligned}
$$

There exists $h < k$, $h \neq k$ such that $f(x_0\|y_k) = f(x_0\|y_h)$.

$$
\begin{aligned}
x_1\|y_1 &= f(x_0 \oplus x_0\|y_0) = f(x_0\|y_0)\\
x_2\|y_2 &= f(x_1 \oplus x_1\|y_1) = f(x_0\|y_1)\\
&\cdot \quad \cdot \quad \cdot\\
x_h\|y_h &= f(x_{h-1} \oplus x_{h-1}\|y_{h-1})\\
&= f(x_0\|y_{h-1})\\
x_{h+1}\|y_{h+1} &= f(x_h \oplus x_h\|y_h) = f(x_0\|y_h)\\
&\cdot \quad \cdot \quad \cdot\\
x_{k+1}\|y_{k+1} &= f(x_k \oplus x_k\|y_k) = f(x_0\|y_k).
\end{aligned}
$$

Since $f(x_0\|y_h) = f(x_0\|y_k)$

$$x_{h+1}\|y_{h+1} = x_{k+1}\|y_{k+1}.$$

Generating collision from the output

- Suppose the squeezing phase produces an ℓ-bit output string.
- We can generate a collision by mounting a birthday attack on the output by evaluating the sponge function approximately $2^{\frac{\ell}{2}}$ times.
- Therefore, we can generate collision by applying the sponge function $\min\{2^{\frac{\ell}{2}}, 2^{\frac{c}{2}}\}$ times.
  - If $\ell \leq c$, then generate collision from ℓ-bit output strings by mounting birthday attack.
  - If $c < \ell$ generate output collision by constructing internal collision using the technique discussed in the previous slide.

SHA-3

- *SHA3-224*, *SHA3-256*, *SHA3-384*, and *SHA3-512*.
- *SHAKE128*, *SHAKE256* are extendable output functions that is abbreviated to *XOF*.

hash function	b	r	c	collision security	preimage security
SHA3-224	1600	1152	448	112	224
SHA3-256	1600	1088	512	128	256
SHA3-384	1600	832	768	192	384
SHA3-512	1600	576	1024	256	512
SHAKE128	1600	1344	256	$\min\{\frac{d}{2}, 128\}$	$\min\{d, 128\}$
SHAKE256	1600	1088	512	$\min\{\frac{d}{2}, 256\}$	$\min\{d, 256\}$

Message Authentication Codes: keyed hash function

Keyed hash function from an unkeyed hash function

- Suppose h is an unkeyed hash function with IV as the initial value that required every input message x to have length that is a multiple of t.
- h utilizes the compression function **compress** : $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$.
- The initial value IV is set to the key K, i.e., $IV = K$.

An iterative keyed has function

$$
\begin{aligned}
z_0 &\leftarrow K \\
z_1 &\leftarrow \mathbf{compress}(z_0 \| y_1) \\
z_2 &\leftarrow \mathbf{compress}(z_1 \| y_2) \\
&\vdots \quad \vdots \\
z_r &\leftarrow \mathbf{compress}(z_{r-1} \| y_r).
\end{aligned}
$$

Unkeyed to keyed hash functions

$IV = K$

$$
\begin{aligned}
z_0 &\leftarrow K \\
z_1 &\leftarrow \mathbf{compress}(z_0 \| y_1) \\
z_2 &\leftarrow \mathbf{compress}(z_1 \| y_2) \\
&\vdots \quad \vdots \\
z_r &\leftarrow \mathbf{compress}(z_{r-1} \| y_r).
\end{aligned}
$$

Length extension attack

- We have x and $h_K(x)$.
- Consider the message $x \| x'$. Then $h_K(x \| x') = \mathbf{compress}(h_k(x) \| x')$.

Length extension attack with padding

Length extension attack with padding

- Suppose $y = x \| \mathbf{pad}(x)$, such that $|y| = rt$.
- Let $|w| = t$. Define: $x' = x \| \mathbf{pad}(x) \| w$.
- $y' = x' \| \mathbf{pad}(x') = x \| \mathbf{pad}(x) \| w \| \mathbf{pad}(x')$. where $|y'| = r't$ for some integer $r' > r$.

Computing $h_K(x')$

Let $z_r = h_K(x)$.

$$
\begin{aligned}
z_{r+1} &\leftarrow \mathbf{compress}(h_K(x) \| y_{r+1}) \\
z_{r+2} &\leftarrow \mathbf{compress}(z_{r+1} \| y_{r+2}) \\
\vdots &\quad \vdots \\
z_{r'} &\leftarrow \mathbf{compress}(z_{r'-1} \| y_{r'})
\end{aligned}
$$

MAC attack models

- The objective of an adversary (Oscar) is to try to produce a message-tag pair (x, y) that is valid under a fixed but unknown key, K.

- Oscar might have access to some valid pairs for the key K:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_Q, y_Q).$$

Two standard attack models are

1. known message attack;
2. chosen message attack.

Forgery

- Suppose Q valid pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_Q, y_Q)$ for an unknown key K is available to Oscar.
- If Oscar can output a message-tag valid pair (x, y) such that $x \notin \{x_1, \ldots, x_Q\}$ with the probability bounded below by ϵ, then Oscar is said to be an (ϵ, Q)-forger for the given MAC.
- The pair (x, y) is said to be a forgery.
- The probability ϵ can be an average-case probability over all possible keys, or the worst-case probability.

Two obvious attacks

- Key guessing attack:
  1. Oscar chooses $K \in \mathcal{K}$ uniformly at random, and outputs the tag $h_K(x)$ for an arbitrary message x.
  2. This attack succeeds with probability $\frac{1}{|\mathcal{K}|}$.
- Tag guessing attack:
  1. Oscar chooses the tag $y \in \mathcal{Y}$ uniformly at random and outputs y has the tag for any arbitrary message x.
  2. This attack succeeds with probability $\frac{1}{|\mathcal{Y}|}$.

Nested MACs

- A nested MAC builds a MAC algorithm from the composition of two (keyed) hash families.
- Compositions of the hash families $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{G})$ and $(\mathcal{Y}, \mathcal{Z}, \mathcal{L}, \mathcal{H})$ is $(\mathcal{X}, \mathcal{Z}, \mathcal{M}, \mathcal{G} \circ \mathcal{H})$ in which $\mathcal{M} = \mathcal{K} \times \mathcal{L}$ and

$$\mathcal{G} \circ \mathcal{H} = \{g \circ h : g \in \mathcal{G}, h \in \mathcal{H}\}$$

  where $(g \circ h)_{(K,L)}(x) = h_L(g_K(x))$ for all $x \in \mathcal{X}$.
- $|\mathcal{Y}| \geq |\mathcal{Z}|$ and $|\mathcal{X}|$ is either finite or infinite.
- If $\mathcal{X}$ is finite, then $|\mathcal{X}| > |\mathcal{Y}|$.

Nested MACs

- The nested MAC is secure if the following two conditions are satisfied:
  1. $(\mathcal{Y}, \mathcal{Z}, \mathcal{L}, \mathcal{H})$ is secure as a MAC, given a fixed (unknown) key.
  2. $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{G})$ is collision resistant, given a fixed (unknown) key.
- We will refer to $(\mathcal{Y}, \mathcal{Z}, \mathcal{L}, \mathcal{H})$ as the "little MAC".
- $(\mathcal{X}, \mathcal{Z}, \mathcal{M}, \mathcal{G} \circ \mathcal{H})$ is the "big MAC" or the "nested MAC."

Adversaries

- We consider the following three adversaries:
  1. a forger for the little MAC (which carries a "little MAC attack"),
  2. a collision-finder for the has family $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{G})$, when the key is secret (this is an "unknown-key collision attack"), and
  3. a forger for the nested MAC (which we term a "big MAC attack").

Types of attacks

- ***Little MAC attack*** : a key L is chosen and kept secret. Oscar is allowed to choose values for y and query a little MAC oracle for the value of $h_L(y)$. Then Oscar attempts to output a pair (y', z) such that $z = h_L(y')$ where y' was not one of his previous queries.
- ***Unknown-key collision attack*** : a key K is chosen and kept secret. Oscar is allowed to choose values for x and query a hash oracle for values of $g_K(x)$. Then Oscar attempts to output a pair x', x'' such that $x' \neq x$ and $g_K(x') = g_K(x'')$.
- ***Big MAC attack*** : a pair of keys (K, L) is chosen and kept secret. Oscar is allowed to choose values for x and query a big MAC oracle for values of $h_L(g_K(x))$. Then Oscar attempts to output a pair (x', z) such that $z = h_L(g_K(x'))$ where x' was not one of its previous queries.

Assumptions

We assume that:

1. there does not exist an $(\epsilon_1, Q+1)$-unknown-key collision attack for a randomly chosen function $g_K \in \mathcal{G}$ where K is secret.

2. there does not exist an (ϵ_2, Q)-little MAC attack for a randomly chosen function $h_L \in \mathcal{H}$, where L is secret.

3. There exists an (ϵ, Q)-big MAC attack for a randomly chosen function $(g \circ h)_{(K,L)} \in \mathcal{G} \circ \mathcal{H}$, where (K, L) is a secret.

The attack

- The big MAC algorithm outputs a valid pair (x, z) after making at most Q queries to a big MAC oracle.
- $x_1, \ldots, x_Q$ are the queries, say, generating valid message-tag pairs $(x_1, z_1), \ldots, (x_Q, z_Q)$, as well the valid message-tag pair (x, z) with probability at least ϵ.
- Make $Q + 1$ queries to a hash oracle g_K to obtain $y_1 = g_K(x_1), \ldots, y_Q = g_K(x_Q)$, and $y = g_K(x)$.
- If $y = y_i$ for some $i \in \{1, \ldots, Q\}$ we have a collision. Else, we have a valid pair for the little MAC, hence a forger for the little MAC.

Probability bounds

- Any unknown-collision attack has probability at most ϵ_1 of succeeding.
- The big MAC attack has success probability at least ϵ.
- Therefore, the probability that (x, z) is a valid pair and $y \notin \{y_1, \ldots, y_Q\}$ is at least $\epsilon - \epsilon_1$.
- The success probability of any little MAC attack is at most ϵ_2.
- So $\epsilon \leq \epsilon_1 + \epsilon_2$.

Theorem

Suppose $(\mathcal{X}, \mathcal{Z}, \mathcal{M}, \mathcal{G} \circ \mathcal{H})$ *is a nested MAC. Suppose that there does not exist an* $(\epsilon_1, Q+1)$*-collision attack for a randomly chosen function* $g_K \in \mathcal{G}$*, when the key* K *is secret. Further, suppose that there does not exist an* (ϵ_2, Q)*-forger for a randomly chosen function* $h_L \in \mathcal{H}$*, where* L *is secret. Finally, suppose there exist an* (ϵ, Q)*-forger for the nested MAC, for a randomly chosen function* $(g \circ h)_{(K,L)} \in \mathcal{G} \circ \mathcal{H}$*. Then* $\epsilon \leq \epsilon_1 + \epsilon_2$*.*

HMAC

- ***HMAC*** is a nested MAC algorithm that was adopted as a FIPS standard in March, 2002.
- $\text{HMAC}_\text{K}(\text{x}) = \text{SHA-1}((\text{K} \oplus \text{opad}) || \text{SHA-1}((\text{K} \oplus \text{ipad}) || \text{x}))$
- $ipad$ and $opad$ are 512-bit constants, defined in hexadecimal notation as $ipad = 3636 \cdots 36$, $opad = 5\text{C}5\text{C} \cdots 5\text{C}$.

CBC-MAC

CBC-MAC(x, K)

denote $x = x_1 \| x_2 \| \cdots \| x_n$;
IV $\leftarrow 00 \cdots 0$;
$y_0 \leftarrow$ IV;
for $i \leftarrow 1$***to***$k - 1$ **do**
  $y_i \leftarrow E_K(y_{i-1} \oplus x_i)$;
end
return y_y;

Algorithm 8: MAC FROM BLOCK CIPHERS

Authenticated encryption

- **MAC-and-encrypt**: Given a message x, compute a tag $z = h_{K_1}(x)$ and a ciphertext $y = e_{K_2}(x)$. The pair (y, z) is transmitted. The receiver would decrypt y, obtaining x, and then verify the correctness of the tag z on x.
- **MAC-then-encrypt** Here the tag $z = h_{K_1}(x)$ would be computed first. Then the plaintext and tag would both be encrypted, yielding $y = e_{K_2}(x \| z)$. The ciphertext y would be transmitted. The receiver will decrypt y, obtaining x and z, and then verify the correctness of the tag z on x.
- **encrypt-then-MAC** Here the first step is to encrypt x, producing a ciphertext $y = e_{K_2}(x)$. Then a tag is created for the ciphertext y, namely, $z = h_{K_1}(y)$. The pair (y, z) is transmitted. The receiver will first verify the correctness of the tag z on y. Then, provided that the tag is valid, the receiver will decrypt y to obtain x.

Authenticated encryption

- Encrypt-then-MAC is preferred over the other methods.
- A security result due to Bellare and Namprempre says that this method of authenticated encryption is secure provided that the two component schemes are secure.
- There exist instantiations of MAC-then-Encrypt and MAC-and-Encrypt then are insecure, even though the component schemes are secure.

Counter with CBC MAC (CCM) mode of operation

- CCM mode computes a tag using CBC-MAC. This is then followed by an encryption in counter mode.
- Let K be the encryption key and let $x = x_1 \| x_2 \| \cdots \| x_n$ be the plaintext.
- We choose a counter ctr, and construct a sequence $T_0, T_1, \ldots, T_n$ defined as $T_i = ctr + i \mod 2^m$ where m is the block length of the cipher.
- The plaintext blocks $x_1, x_2, \ldots, x_n$ are encrypted by computing $y_i = x_i \oplus e_K(T_i)$.
- Compute $temp =$ CBC-MAC(x, K) and $y' = T_0 \oplus temp$.
- The ciphertext is the string $y = y_1 \| y_2 \| \cdots \| y_n \| y'$.

Decryption

- To decrypt and verify y, one would first decrypt $y_1 \| \cdots \| y_n$ using the counter mode decryption with the counter sequence $T_1, T_2, \ldots, T_n$, obtaining the plaintext string x.
- The second step is to compute CBC-MAC(x, K) and see if it is equal to $y' \oplus T_0$.
- The ciphertext is rejected if this condition does not hold.

Galois Counter mode

- A detailed description of GCM is given in NIST Special Publication 800-38D.
- The encryption is done in counter mode using a 128-bit AES key. The initial value of the 128-bit counter is derived from an IV that is typically 96 bits in length.
- The IV is transmitted along with the ciphertext, and it should be changed every time a new encryption is performed.
- The computation of the authentication tag requires performing multiplications by a constant value H in the finite field $\mathbb{F}_{2^{128}}$. The value of H is determined by encrypting Counter 0.

Unconditionally secure MACS

- Assumptions:
  - The adversary has infinite computational power.
  - Any given key is used to produce only one authentication tag.
- For $Q \in \{0, 1\}$ we define deception probability Pd_Q to be the probability that the adversary can create a successful forgery after observing Q valid message-tag pairs.
- The attack when $Q = 0$ is said to be impersonation attack, and the attack when $Q = 1$ is said to be substitution attack.

Unconditionally Secure MACs

- Assumption: the key K is chosen uniformly at random from $\mathcal{K}$.
- In a substitution attack Oscar's success probability ϵ may depend on the the particular message-tag pair (x, y) that he observes.
- We will assume Pd_1 to be the maximum of the relevant values $\epsilon(x, y)$.
- Thus when we say that $Pd_1 \leq \epsilon$, it means that Oscar's success probability is at most ϵ regardless of the message-tag pair he observes prior to making his substitution.

Unconditionally Secure MACs

Example

Suppose $\mathcal{X} = \mathcal{Y} = \mathbb{Z}_3$, and $\mathcal{K} = \mathbb{Z}_3 \times \mathbb{Z}_3$. For each $K = (a, b) \in \mathcal{K}$ and each $x \in \mathcal{X}$, define $h_{(a,b)}(x) = ax + b \mod 3$, and then define $\mathcal{H} = \{h_{(a,b)} : (a, b) \in \mathbb{Z}_3 \times \mathbb{Z}_3\}$. Each of the 9 keys are used with probability $\frac{1}{9}$.

key	0	1	2
$(0,0)$	0	0	0
$(0,1)$	1	1	1
$(0,2)$	2	2	2
$(1,0)$	0	1	2
$(1,1)$	1	2	0
$(1,2)$	2	0	1
$(2,0)$	0	2	1
$(2,1)$	1	0	2
$(2,2)$	2	1	0

Table 1: An authentication matrix

Unconditionally Secure MACs

Deception probabilities

- Any message-tag pair (x, y) will be a valid pair with probability $\frac{1}{3}$.
- So $Pd_0 = \frac{1}{3}$.
- If Oscar sees the message-tag pair $(0, 0)$ he knows that $K_0 = \{(0, 0), (1, 0), (2, 0)\}$.
- $(1, 1)$ is a forgery if $K_0 = (1, 0)$. This happens with probability $\frac{1}{3}$.
- Repeating this for all possible message-tag pairs gives us the same probability. So $Pd_1 = \frac{1}{3}$.

key	0	1	2
$(0, 0)$	0	0	0
$(0, 1)$	1	1	1
$(0, 2)$	2	2	2
$(1, 0)$	0	1	2
$(1, 1)$	1	2	0
$(1, 2)$	2	0	1
$(2, 0)$	0	2	1
$(2, 1)$	1	0	2
$(2, 2)$	2	1	0

Table 2: An authentication matrix

Payoff: deception probability of impersonation

- Let K_0 denote the key chosen by Alice and Bob. For $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, define $\mathbf{payoff}(x, y)$ to be the probability that the message-tag pair (x, y) is valid.

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

- $Pd_0 = \max\{\mathbf{payoff}(x, y) : x \in \mathcal{X}, y \in \mathcal{Y}\}$.

Payoff: deception probability of substitution

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

- $\mathcal{V} = \{(x, y) : \left|\{K \in \mathcal{K} : y = h_K(x)\}\right| \geq 1\}$.
- $Pd_1 = \max_{(x,y)\in\mathcal{V}}\{\max_{(x',y'),x'\neq x}\{\textbf{payoff}(x', y'; x, y)\}\}$.

Strongly Universal Hash Families

Definition

Suppose $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ is an (N, M) hash family. This hash family is strongly universal provided that the following condition is satisfied for every $x, x' \in \mathcal{X}$ such that $x \neq x'$, and for every $y, y' \in \mathcal{Y}$:

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

Strongly Universal Hash Families

Suppose that $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ is a strongly universal (N, M)-hash family. Then

$$\left|\{K \in \mathcal{K} : h_K(x) = y\}\right| = \frac{|\mathcal{K}|}{M},$$

for every $x \in \mathcal{X}$ and for every $y \in \mathcal{Y}$.

Suppose $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ is a strongly universal (N, M)-hash family. Then $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ is an authentication code with $Pd_0 = Pd_1 = \frac{1}{M}$.

Optimality of Deception Probabilities

- Suppose $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ is and (N, M)-hash family. Suppose we fix a message $x \in \mathcal{X}$. Then we can computer as follows:

$$\sum_{y \in \mathcal{Y}} \mathbf{payoff}(x, y) = \sum_{y \in \mathcal{Y}} \frac{\left|\{K \in \mathcal{K} : h_K(x) = y\}\right|}{|\mathcal{K}|}$$
$$= \frac{|\mathcal{K}|}{|\mathcal{K}|} = 1.$$

- Hence, for every $x \in \mathcal{X}$, there exists an authenticating tag y (depending on x), such that

$$\mathbf{payoff}(x, y) \geq \frac{1}{M}.$$

Optimality of deception probabilities

Theorem

Suppose $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ *is an* (N, M)*-hash family. Then* $Pd_0 \geq \frac{1}{M}$*. Further* $Pd_0 = \frac{1}{M}$ *if and only if*

$$\left|\{K \in \mathcal{K} : h_K(x) = y\}\right| = \frac{|\mathcal{K}|}{M}$$

for every $x \in \mathcal{X}$ *and* $y \in \mathcal{Y}$*.*

Theorem

Suppose $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ *is an* (N, M)*-hash family. Then* $Pd_1 \geq \frac{1}{M}$*.*

Optimality of deception probabilities

Theorem

Suppose $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ *is an* (N, M)*-hash family. Then* $Pd_1 = \frac{1}{M}$*. Further* $Pd_0 = \frac{1}{M}$ *if and only if the hash family is strongly universal.*

Theorem

Suppose $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ *is an* (N, M)*-hash family such that* $Pd_1 = \frac{1}{M}$*. Then* $Pd_0 = \frac{1}{M}$.

The End