Cryptographic Hash Functions

Electrical and Computer Engineering, University of Arizona,
Loukas Lazos

1 Cryptographic Hash Functions

A hash is a short fingerprint of the data. We need the fingerprint to be unique and also shorter than the message/data length. An example is as shown in Figure 1. The hash \( h(m) \) (the fingerprint) will allow us to check if message \( m \) was modified. An example follows.

Let \( h(m) = m^2 \mod 2 \). Let \( m = 5 \) i.e. 101 binary string, then \( h(m) = 5^2 = 25 \equiv 1 \mod 2 \). Now if we change \( m = 5 \) to \( m = 6 \) i.e. binary string 101 to 110 then \( h(m) = 6^2 = 36 \equiv 0 \mod 2 \). So if the message \( m \) has changed from even to odd or odd to even then we can detect it. However if \( m = 5 \) is modified to \( m = 7 \), then \( h(m) = 7^2 = 49 \equiv 1 \mod 2 \). This change cannot be detected! Hence we cannot use \( h(m) = m^2 \mod 2 \) as a good fingerprint of data. We will not always know if it was changed.

1.1 Desired Properties of a Hash Function

Let \( X \) be the set of possible messages (called domain). Let \( Y \) be the set of hash values (called range). Let \( h : X \rightarrow Y \) be the hash function.

1. Given only the hash value, one should not find a different message that generates same hash. This is called Preimage problem. Formally,
Given: \( h : X \rightarrow Y \) and \( y \in Y \).
Problem: Find \( x \in X \) such that \( h(x) = y \).
A hash function for which the preimage problem cannot be efficiently solved, is called one-way or Preimage-resistant.

2. Given a message \( x \), and a hash function \( h \), one should not find another message \( x' \) that yields the same hash. This problem is known as the 2nd Preimage problem. Formally,
Given: \( h : X \rightarrow Y \) and \( x \in X \).
Problem: Find \( x' \in X \), \( x' \neq x \) such that \( h(x) = h(x') \).
A hash function for which the second preimage problem cannot be efficiently solved is called second Preimage-resistant.

3. Given only the hash function \( h \), find two messages that yield the same hash value. This is called the Collision problem. Formally,
Given: \( h : X \rightarrow Y \).
Problem: Find \( x, x' \in X \) such that \( x \neq x' \) and \( h(x') = h(x) \).
A hash function for which the collision problem cannot be efficiently solved, is called collision-resistant.

Now consider the previous example, \( h(x) = x^2 \mod 2 \). Say, we are given \( h(x) = 1 \mod 2 \). Then it is not difficult to choose \( x \). If we let \( x = 2k + 1 \) then \( k \pm 1, \pm 2, \pm 3, \ldots \) (all the odd values of \( x \)) will
satisfy $x^2 \equiv 1 \mod 2$. Therefore \textbf{Preimage} is easy! which implies that $h(x) = x^2 \mod 2$ is a \textit{weak} hash function.

Clearly, $x = 2k + 1$, $x' = 2l + 1$, where $k = \pm 1, \pm 2, \ldots$ and $l = \pm 1, \pm 2, \ldots$ and for $l \neq k$ we get many solutions. As an example, $x = 1, x' = 3$ and so on. So even 2\textsuperscript{nd} \textbf{Preimage} is easy! We can similarly observe that obtaining \textbf{Collision} is also trivial in this hash.

1.2 \textbf{The Random Oracle Model}

An idealized model intended to capture the “ideal” hash function operation. In the random oracle model, an adversary is given a black box and the specific implementation details of the hash function are not disclosed. The only way to compute the output of the black box is to ask the oracle for the specific output given a specific input. If the oracle is ideal, then all the responses look random, regardless of how many queries we have done so far. Formally the following theorem shall hold.

\textbf{Theorem 1.} Suppose that a hash function $h$ has been chosen at random from a family of hashes $F^{X \times Y}$ and let $X_0 \subseteq X$. Suppose that the values $h(x)$ have been determined by querying the oracle if and only if $x \in X_0$. Then $\mathbf{Pr}[h(x) = y] = \frac{1}{M}$ for all $x \in X / X_0$ and all $y \in Y$.

1.3 Analyzing the Security of Hashes using the Random Oracle Model

\textbf{Definition 1. Las Vegas algorithm:} A randomized algorithm\textsuperscript{1} which may give the answer correct or failure.

An $(\epsilon, Q)$-algorithm is a Las Vegas algorithm with average success probability $\epsilon$ when the maximum number of queries are $Q$. The success probability is analyzed over all possible hash functions from the hash family and all possible query sets.

\textbf{Solving the Preimage problem}

\textbf{Algorithm 4.1: FIND-PREIMAGE}(h, y, Q)

\begin{itemize}
\item choose any $X_0 \subseteq X$, $|X_0| = Q$
\item for each $x \in X_0$
\item do: if $h(x) = y$ return $x$
\item else return failure.
\end{itemize}

\textbf{Theorem 2.} For any $X_0 \subseteq X$ with $|X_0| = Q$ the average success probability of algorithm 4.1 is

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

\textsuperscript{1} An algorithm that makes random choices during execution.
Proof. Fill the proof here
Algorithm 4.2: FIRST-SECOND-PREIMAGE\((h, x, Q)\)

\[ y \leftarrow h(x) \]
choose \(X_0 \subseteq X \setminus \{x\}, |X_0| = Q - 1\)
for each \(x_0 \in X_0\)
do: if \(h(x_0) = y\) return \(x_0\)
else return failure.

Theorem 3. For any \(X_0 \subseteq X \setminus \{x\}\) with \(|X_0| = Q - 1\) the average success probability of algorithm 4.2 is

\[
\epsilon = 1 - \left(1 - \frac{1}{M}\right)^{Q-1} \approx \frac{Q}{M},
\]

(2)

Proof. Similar to the proof of algorithm 4.1.

Algorithm 4.3: FIRST-COLLISION \((h, Q)\)

choose \(X_0 \subseteq X, |X_0| = Q\)
for each \(x_0 \in X_0\)
do: \(y_x \leftarrow h(x)\)
if \(y_x = y_{x'}\) for some \(x' \neq x\)
then return \((x, x')\)
else return failure.

The algorithm 4.3 is analyzed using a probabilistic argument known as the Birthday Paradox, or birthday problem presented in the section below.

1.4 Birthday Problem

Consider our class of size \(Q = 8\). The number of days in a year is \(M = 365\). Let us now find the probability that at least two of us have the same birthday.

Pick the first one. The probability of 2\textsuperscript{nd} one having different birthday is \(\frac{364}{365}\). The probability that the 3\textsuperscript{rd} person has a different birthday is \(\frac{363}{365}\).

Hence, the probability of 3 having different birthdays is \((\frac{363}{365})(\frac{362}{365})\). All eight of us having different birthdays is then:

\[
\prod_{i=1}^{7}(1 - \frac{i}{365}) = \prod_{i=1}^{7}(1 - \frac{i}{365}) \leq \prod_{i=1}^{8-1} e^{-\frac{i-1}{365}} = e^{-\frac{\sum_{i=1}^{8-1} i}{365}} = e^{-\frac{q(q-1)}{2M}}.
\]

(3)

In the above equation, we have used the approximation, \(e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \ldots \approx 1 - x\), where \(x\) is small. Now, the probability that at least two having the same birthday, say \(\epsilon\), is:

\[
\epsilon = 1 - e^{-\frac{q(q-1)}{2M}},
\]

(4)

i.e. \(e^{-\frac{q(q-1)}{2M}} = 1 - \epsilon\),

(5)

i.e. \(q(q-1) \approx 2M \ln(\frac{1}{1 - \epsilon})\).
In general from the above equation by assuming \((q^2 - q) \approx q^2\), we get,

\[ q \approx \sqrt{2M \ln \left( \frac{1}{1-\epsilon} \right)} \tag{7} \]

When \(\epsilon = 0.5\), \(q \approx 1.177\sqrt{M}\).

When \(M = 365\), \(q \approx 22.4\) or 23 as integer.

Hence, we need only 23 people to have at least 50% chance to get identical birthdays.

For the general case, if we take a 40 bit hash i.e. \(M = 2^{40}\) then \(1.177\sqrt{M} \approx \sqrt{M} = 2^{20} \approx 10^6\).

This implies that \(10^6\) random choices are sufficient to find a collision. A collision implies that the hash function is weak. Hence the usual length is 128 bits and the suggested length is \(160\) bits. Then a collision may only occur when \(2^{80} \approx 10^{24}\) random choices! are considered.

2 Security criteria for Hashes

Let there be an efficient algorithm to solve the second preimage problem for a particular hash function \(h\). Then the algorithm, given inputs \((x, h)\) returns a value \(x' \neq x\) such that \(h(x) = h(x')\).

Hence, using the algorithm for solving the second preimage problem one can find a collision for the particular hash \(h\). Hence, we can say that the property of collision resistance implies the property of second preimage resistance.

Algorithm 4.4: COLLISION-TO-SECOND PREIMAGE\((h)\)

choose any \(x \in \mathcal{X}\) uniformly at random
if ORACLE-SECOND-PREIMAGE\((h, x) = x'\) then return \((x, x')\) else return failure.

Let there be an efficient algorithm to solve the Preimage problem with probability 1. That is there is an \((1, Q)\) Las Vegas algorithm that solves the preimage problem. Assume that this algorithm solves the preimage problem for some hash function \(h\) with \(|\mathcal{X}| \geq 2|\mathcal{Y}|\). Then we can solve the collision problem with the following algorithm.

Algorithm 4.5: COLLISION-TO-PREIMAGE\((h)\)

choose any \(x \in \mathcal{X}\) uniformly at random
if ORACLE-PREIMAGE\((h, x) = x'\) and \((x \neq x')\) then return \((x, x')\) else return failure.

For the algorithm 4.5 we can prove the following theorem:

**Theorem 4.** The collision-to-preimage algorithm is a \(\left(\frac{1}{2}, Q + 1\right)\)-algorithm for collision, for a fixed hash function \(h\).

**Proof.** In the book.

Under certain conditions, by solving the preimage problem one can solve the collision problem with a high probability. Hence, collision resistance implies preimage resistance (under the pre-specified conditions).
3 Iterated Hash Functions

Functions we have seen so far are compress functions, reducing a fixed size input to a fixed length output. However, we would like to generate hashes for messages of arbitrary length. Using iterated hashes, we can extend the domain of the hash function to infinity. Iterative functions are the most commonly used hashes in practical applications. The computation of an iterative hash consists of three steps.

1. **Pre-processing step:** Given an input $x$ of length greater than some number $m + t + 1$ construct a string $y$ using a public algorithm such that the length of $y$ is a multiple of $t$.

   \[ y = y_1 || y_2 || ... || y_r, \quad |y_i| = t, \forall i. \quad (8) \]

2. **Processing step:** Use a publicly known initialization vector (IV) of length $m$ as the first input and compute the following:

   \[
   \begin{align*}
   z_0 & \leftarrow IV \\
   z_1 & \leftarrow \text{compress}(z_0 || y_1) \\
   z_2 & \leftarrow \text{compress}(z_1 || y_2) \\
   & \quad \vdots \\
   z_r & \leftarrow \text{compress}(z_{r-1} || y_r). 
   \end{align*}
   \]
3. **Output Transformation** (Optional): Let \( g : \{0,1\}^m \to \{0,1\}^t \) be a public function. Define \( h(x) = g(z_r) \).

4 **Message Authentication Codes**

Message Authentication Codes (MAC) are keyed hash functions, that is, hash functions that take as an input not only the message but also a key.

**Question:** Why is the key needed for a hash to work as a MAC? Can this MAC be used as a source authentication as well?

To generate a keyed hash, the key has to be somehow incorporated in the hash generation. One way to incorporate the key is by making it part of the message to be hashed. However this needs to be incorporated in the “right” manner to avoid certain attacks.

**Goal of the adversary:** Produce a valid \((x, y)\) pair, when the key \(K\) is unknown.

**Example of MAC generation:** Assume that the the key is the initial vector (IV) and is kept secret. Assume that the hash is generated with no pre-processing step and no output transformation. Since there is no pre-processing step, this hash takes as input any message of length multiple of \(t\) and compresses it to a message of length \(t\).

Let an adversary pick a message \(x\) and pass it through \(h\) to obtain \(h(x)\). Let \(x'\) be another message of length \(t\) and consider the concatenation \(x||x'\). The MAC for the concatenated message is

\[
h_K(x||x') = \text{compress}(h_K(x)||x').
\]  

Since both \(h_K(x), x'\) are known, the adversary can compute the MAC for \((x||x')\), without knowing the key \(K\).

Assume now that the pre-processing step is included, and involves padding of the message by a number of bits. Suppose that the message is padded so it becomes a multiple of \(t\). After pre-processing the message is \(y = x||\text{pad}(x)\), with \(|y| = rt\) for some positive integer \(r\). Let \(w\) be a bitstring of length \(t\) and let \(x' = x||\text{pad}(x)||w\).

Then in the pre-processing step we compute

\[
y' = x'||\text{pad}(x')||w||\text{pad}(x'),
\]  

with \(|y'| = r't\). After \(r\) rounds of compression we have that \(z_r = h_K(x)\). The adversary can

\[
z_{r+1} \leftarrow \text{compress}(h_K(x)||y_{r+1})
\]
\[
z_{r+2} \leftarrow \text{compress}(z_{r+1}||y_{r+2})
\]
\[
\vdots
\]
\[
z_r \leftarrow \text{compress}(z_{r-1}||y_r).
\]
The adversary can actually compute \( h_K(x') \) without knowing the key \( K \) by just using \( h_K(x), y \).

### 4.1 Formal Definition of the Adversary

The objective of the adversary is to generate a pair \((x, y)\) such that \( y = h_K(x) \) given that the key is unknown. The adversary has at its disposal up to \( Q \) valid MACs generated from chosen plaintexts. The pair \((x, y)\) if found, is said to be a forgery. If the probability of forgery is at least \( \epsilon \), the adversary is said to be an \((\epsilon, Q)\)-forger for the particular MAC. The probability \( \epsilon \) is considered to be either the average case or the worst case probability, considering all possible keys.

### 4.2 CBC-MAC

An example of a keyed hash function is the Cipher-Block-Chaining MAC called CBC-MAC, based on the CBC mode of operation.

Let the message be \( x = x_0 || x_1 || ... || x_{n-1} \). Let an initial vector \( IV = 0_t = 000...0 = y_0 \). i.e. IV contains \( t \) zeros. The algorithm is as follows:

\[
\text{for } (i = 1; i \leq n; i++) \{ \quad /* \text{loop} */ \\
y_i = e_K(y_{i-1} \oplus x_i); \quad \}
\]

\[
\text{RETURN}(y_n);
\]

The main idea:

1. Take message and break into blocks of size \( t \).
2. Set Initial Vector, \( IV = y_0 = 000...0 \) (\( t \) zeroes).
3. Choose a key \( K \) and encryption method \( e_K \). Example, \( e_K = x + K \mod 26 \).
4. Compute loop till all blocks are done. Final encryption is the Hash MAC.

### 4.3 CBC-MAC Example

Let \( t = 4 \) i.e. block size=4. \( IV = y_0 = 0000 \). Let the key \( K = 1101 \). Let \( e_K(x) = x \oplus K \).

Let \( x = 1001 \ 1101 \ 1011 \ 0110 \ 1111 \).

Alice and Bob know \( K \). Eve does not know \( K \).

(1) Alice splits message into blocks of size \( t = 4 \).
(2) Hence Alice will send Bob: \( x || h_K(x) = 1001 \ 1101 \ 1011 \ 0110 \ 1111 \ 1011 \).

Questions: (a) How does Bob know the message length? (b) Where should Alice put that information? Ans: Alice needs to agree with Bob about the maximum size of message or block size. Knowing max length/block length size will allow Bob to compute MAC and compare.
Problem with the example? Note $y_0 = 0000$.

$y_1 = K \oplus y_0 \oplus x_0 = e_K(IV \oplus x_0)$

$y_2 = e_K(y_1 \oplus x_1) = K \oplus (y_1 \oplus x_1)$

$= y_1 \oplus x_0 \oplus x_1 \leftarrow$ No effect of key $K$. Note that there are now two blocks of $x$.

$y_3 = e_K(y_2 \oplus x_2)$

$= K \oplus (y_0 \oplus x_0 \oplus x_1 \oplus x_2)$

Similarly,

$y_4 = y_0 \oplus x_0 \oplus x_1 \oplus x_2 \oplus x_3 \leftarrow$ No effect of key. 4 message blocks are present.

Claim: If $|x_1|_{even}$, then we use shift cipher, then we are in trouble!

As an example: Take first two blocks sent by Alice, $x_0 = 1001, x_1 = 1101, y_2 = y_0 \oplus x_0 \oplus x_1 = 0000 \oplus 1001 \oplus 1101 = 0100$. Hence Eve can obtain the CBC-MAC as 1011. Note that if Eve choose $x_0 = 0110, x_1 = 0010$, then $y_1 = 0100$, which implies collision. Bob will not know that the message has been modified by Eve!!

CBC-MACs are assumed to be secure if the underlying encryption scheme satisfies certain properties, regarding the randomness of the cipher. Example is the DES encryption that indeed exhibits these properties, but they have not been formally proven.

### 4.4 Unconditionally Secure MACs

To construct unconditionally secure MACs, we assume that the keys are used only once (like a one time pad), hence the adversary is limited to at most one query to forge a message. We look into $(\epsilon, 0)$ and $(\epsilon, 1)$-forgers.

**Definition 2. Deception Probability** $P_{d_q}$: The maximum value of $\epsilon$ such that an $(\epsilon, q)$-forger exists, computed over all possible values of the key $K$, when the key is randomly selected from $\mathcal{K}$. 

![Fig. 3. Schematic of CBC MAC](image)
An attack launched by an \((\epsilon, 0)\)-forger is usually termed as an \textit{impersonation}, while an attack carried by an \((\epsilon, 1)\)-forger is termed as \textit{substitution}.

**Example of unconditionally secure MAC**

Let \(\mathcal{X} = \mathcal{Y} = \mathbb{Z}_3\) and \(\mathcal{K} = \mathbb{Z}_3 \times \mathbb{Z}_3\). Define the hashing operation as

\[
h_{(a,b)}(x) = ax + b \mod 3
\]

and define the family of possible hashes as:

\[
\mathcal{H} = \{h_{(a,b)} : (a, b) \in \mathbb{Z}_3 \times \mathbb{Z}_3\}.
\]

**Question:** Is \(h\) invertible?

We can actually build the authentication matrix for the family of hashes \(\mathcal{H}\) that gives the output of all possible combinations of \(x, K\).

**Table 1. Authentication matrix**

<table>
<thead>
<tr>
<th>Key/x</th>
<th>(x = 0)</th>
<th>(x = 1)</th>
<th>(x = 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(0,1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(0,2)</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>(1,0)</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(1,1)</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>(1,2)</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(2,0)</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>(2,1)</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(2,2)</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Fig. 4.** Schematic of CBC MAC
Consider first the impersonation attack. Since the adversary does not have any plaintext MAC pair, all it can do is select a plaintext and try to guess the right MAC tag. However each $y$ appears exactly 3 times on each column. Hence, given that the key is chosen at random, the adversary has a probability of success equal to $P_{d_0} = \frac{1}{3}$.

Consider now the substitution attack. Assume that the adversary picks $x = 0$ and observes that $y = 0$. Given the authentication matrix, now the adversary known that $K_0 \in \{(0, 0), (1, 0), (2, 0)\}$

Now if the adversary selects the pair $(1, 1)$ that corresponds to $K_0 = (1, 0)$, it has a probability of success of $\frac{1}{3}$. If we compute the probability of success for all possible selections of $x$, we can compute that $P_{d_1} = \frac{1}{3}$.

We can generalize the computation of the deception probabilities for an arbitrary MAC given an authentication matrix by the definition of the payoff $(x, y)$ which is the probability that a pair $(x, y)$ is valid, given that a particular key $K_0$ is selected.

$$\text{payoff}(x, y) = \frac{\Pr[y = h_{K_0}(x)]}{|K|} = \frac{|\{K \in K : h_K(x) = y\}|}{|K|}.$$ (15)

To maximize the chance of success in a substitution attack, Oscar will pick the pair $(x, y)$ that maximizes the payoff. Hence,

$$P_{d_0} = \max\{\text{payoff}(x, y) : x \in X, y \in Y\}. \quad \text{(16)}$$

Define the “conditional” payoff $(x', y'; x, y)$ to be the probability that $(x', y')$ is a valid pair, given that $(x, y)$ is a valid pair. Let $K_0$ be the key chosen by Alice and Bob.

$$\text{payoff}(x', y'; x, y) = \frac{\Pr[y' = h_{K_0}(x') | y = h_{K_0}(x)]}{\Pr[y = h_{K_0}(x)]} = \frac{|\{K \in K : h_K(x') = y', h_K(x) = y\}|}{|\{K \in K : h_K(x) = y\}|}. \quad \text{(17)}$$

In the above formula, the numerator of the fraction is the number of rows in the authentication metric that have a value of $y$ in the column $x$, and also have a value $y'$ in column $x'$, while the denominator is the number of rows that have a value $y$ in column $x$. To compute the deception probability we use the following formula

$$P_{d_1} = \max_x \{\min_y \{\max_{(x', y')} \{\text{payoff}(x', y'; x, y)\}\}\}, \quad \text{(18)}$$

with $x' \neq x$ and $(x, y)$ being a valid pair.

The adversary first chooses $x$ and obtains $y$. Then he guesses on the pair $(x', y')$ so as to maximize the payoff. The adversary maximizes with respect to $x$, and $(x', y')$ because it has the freedom to choose the appropriate values. On the other hand the value of $y$ is defined by the selection of the key $K$ that is not controlled by the adversary. Given that we want to say that the success probability is at least $P_{d_1}$, minimize over all possible choices of $y$. 

4.5 Strongly Universal Hash Families

**Definition 3.** Let \((X, Y, K, H)\) be an \((N, M)\)-hash family. This hash family is said to be strongly universal if for every \(x, x' \in X\), \(x \neq x'\) and every \(y \neq y' \in Y\):

\[
|\{K \in K : h_K(x) = y, h_K(x') = y'\}| = \frac{|K|}{M^2}.
\]  
(19)

**Lemma 1.** Let \((X, Y, K, H)\) be an \((N, M)\)-hash family. Then

\[
|\{K \in K : h_K(x) = y\}| = \frac{|K|}{M}, \forall x \in X, \forall y \in Y.
\]  
(20)

**Proof.** Let \(x, x' \in X\) and fix \(y \in Y\), with \(x \neq x'\). Then

\[
|\{K \in K : h_K(x) = y\}| = \sum_{y' \in Y} |\{K \in K : h_K(x) = y, h_K(x') = y'\}|
\]

\[
= \sum_{y' \in Y} \frac{|K|}{M^2}
\]

\[
= \frac{|K|}{M}.
\]  
(21)

**Theorem 5.** Let \((X, Y, K, H)\) be an \((N, M)\)-hash family. Then \((X, Y, K, H)\) is an authentication code with \(P_{d_0} = P_{d_1} = \frac{1}{M}\).

**Proof.** Use the above lemma to prove that \(P_{d_0} = \frac{1}{M}\).

Consider now the case for \(P_{d_1}\). Let \(x, x' \in X\) and \(y, y' \in Y\), with \(x \neq x'\), and \((x, y)\) being a valid pair. Then

\[
\text{payoff}(x', y'; x, y) = \frac{|\{K \in K : h_K(x') = y', h_K(x) = y\}|}{|\{K \in K : h_K(x) = y\}|}
\]

\[
= \frac{|K|}{M \times \frac{|K|}{M}}
\]

\[
= \frac{1}{M}.
\]  
(22)

Given the desirable properties of strongly universal hash function the relevant question is how do we construct such functions.

**Theorem 6.** Let \(p\) be prime. For \(a, b \in \mathbb{Z}_p\) define function \(f\) to be \(f_{(a,b)} : \mathbb{Z}_p \to \mathbb{Z}_p\) by the rule

\[
f_{(a,b)}(x) = ax + b \mod p.
\]  
(23)

The hash family \((\mathbb{Z}_p, \mathbb{Z}_p, \mathbb{Z}_p \times \mathbb{Z}_p, \{f_{(a,b)} : a, b \in \mathbb{Z}_p\})\) is a strongly universal \((p, p)\)-hash family.
Theorem 7. Let \( \ell \) be a positive integer and let \( p \) be a prime number. Define

\[
X = \{0, 1\}^\ell \setminus \{0, \ldots, 0\}.
\]

For every \( r \in (\mathbb{Z}_p)\), define \( f_r : X \to \mathbb{Z}_p \) by the rule

\[
f_r(x) = (rx) \mod p,
\]

where \( x \in X \) and

\[
(rx) = \sum_{i=1}^\ell r_i x_i
\]

is the inner product of two vectors.

Then \((X, \mathbb{Z}_p, (\mathbb{Z}_p)\), \{\(f_r : r \in (\mathbb{Z}_p)\}\}\) is a strongly universal \((2^\ell - 1, p)\)-hash family.

5 Applications of Hash Functions

We have already seen the application of keyed hashes for the generation of MACs. Other applications involve the following.

5.1 Source Authentication

Assume that a company where to store a file with all the passwords of its clients. The simplest way of doing so is by listing all the passwords in one file. If the file were to be stolen, all the passwords would be compromised. An alternative way is to store the hashed value of the password instead of the actual password value. When the user wants to log in into the system, it inputs his password and the hash value is checked against the stored value. If the password file were to be stolen, the adversary would not be able to recover the passwords from the file, unless it solved the pre-image or collision problem.

Question: Do you see any problem with this type of authentication?

One way hash chains: A one-way hash chain is generated by the iterative application of a one-way hash function to an initial seed. The chain is used in the opposite way that is generated.

![Fig. 5. Generation and use of a one-way hash chain.](image-url)
The chain size is the constraining factor for the use of one way hash chains. Also there is a tradeoff between computation and storage for one way hash chains. One can reduce the storage and computation requirements to $O(\log n)$. An alternative way is to construct Merkle trees.

**Merkle trees**
A method to commit(sign) to $n$ values using hashes. Can be used to verify the source authenticity. Assume that Alice wants to commit to $n$ values (these can any data such as parts of a file, measurements, etc). The notion of committing here is that one can indisputably prove that these were the values that were initially committed to (like signing a contract).

![Fig. 6. Generation and use of a one-way hash tree.](image)

In order for Alice to commit to values $v_1 \ldots v_n$ it generates a tree as in figure 6. To prove that a value $v_i$ is authentic, it releases to Bob $v_i$, $i$ and all the hash siblings in the path from the leaf $v_i$ to the root of the tree. The root of the tree is the commitment for all values $v_1 \ldots v_n$. Bob, can verify the authenticity of $v_i$ by computing the hash of the root and comparing it with the one already stored.

**Question:** How is the authenticity of any value guaranteed?

**Question:** Why are the initial values originally blinded?