One-way Hash Chain

- Used for many network security applications
- Example: S/Key
- Good for authentication of the hash values

Properties of One-way Hash Chain

- Given $K_i$
  - Anybody can compute $K_j$, where $j<i$
  - It is computationally infeasible to compute $K_j$, where $l > i$, if $K_i$ is unknown
  - Any $K_j$ disclosed later can be authenticated by verifying if $H^i(K_i) = K_j$
  - Disclosing of $K_{i+1}$ or a later value authenticates the owner of the hash chain
Merkle Hash Tree

• A binary tree over data values
  – For authentication purpose
• The root is the commitment of the Merkle tree
  – Known to the verifier.
• Example
  – To authenticate \( k_2 \), send \( (k_2, m_3, m_0, m_47) \)
  – Verify \( m_07 = h(h(m_01 || h(f(k_2) || m_3)) || m_47)) \)

Merkle Hash Tree (Cont’d)

• Hashing at the leaf level is necessary to prevent unnecessary disclosure of data values
• Authentication of the root is necessary to use the tree
  – Typically done through a digital signature or pre-distribution
• Limitation
  – All leaf values must be known ahead of time

Bloom Filters

• It is used to verify that some data is not in the database (mismatch)
  – List of bad credit card numbers
  – Useful when the data consumes a very small portion of search space
• A Bloom filter is a bit string
• \( k \) hash functions that map the data into \( n \) bits in the Bloom filter
A Simple Example

- Use a bloom filter of 16 bits
  - \( H_1(key) = key \mod 16 \)
  - \( H_2(key) = key \mod 14 + 2 \)

- Insert numbers 27, 18, 29 and 28

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- Check for 22:
  - \( H_1(22) = 6, H_2(22) = 10 \) (not in filter)

- Check for 51
  - \( H_1(51) = 3, H_2(51) = 11 \) (false positive)

Probability of False Positive

- Consider an \( m \)-bit Bloom filter with \( k \) hash functions
  - After inserting \( n \) elements, the probability of false positive

\[
\left(1 - \left(1 - \frac{1}{m}\right)^k\right)^n = \left(1 - e^{-kn/m}\right)^n.
\]

Client Puzzles

- The problem being addressed
  - Denial of Service (DoS) attacks

- Three basic constructions
  - Use pre-image of crypto hash functions
  - Use special image of crypto hash functions
  - Use constrained discrete logarithm problem (DLP)
Client Puzzle Based on Pre-image of Crypto Hash Functions

An Example Scenario: TCP SYN Flooding

Client Puzzle: Intuition
Client Puzzle: Intuition

- A puzzle takes an hour to solve
- There are 40 tables in restaurant
- Reserve at most one day in advance

A legitimate patron can easily reserve a table

An attacker has to reserve many tables to have a real impact
→ too many puzzles to solve

The Client Puzzle Protocol

Client

Service request $M$

Server

Buffer

O.K.
Puzzle Basis: Partial Hash Image

Pair \((X', Y)\) is \(k\)-bit-hard puzzle

Puzzle Basis (Cont’d)

- Only way to solve puzzle \((X', Y)\) is brute force method. (hash function is not invertible)
- Expected number of steps (hash) to solve puzzle: \(2^k / 2 = 2^{k-1}\)

Puzzle Construction

Client

Service request \(M\)

Server

Secret \(S\)
Puzzle Construction

Server computes:

- Secret $S$
- Time $T$
- Request $M$

Hash

Pre-image $X$

Hash

Image $Y$

**Sub-puzzle**

- Construct a puzzle consisting of $m$ $k$-bit-hard sub-puzzles.
- Increase the difficulty of guessing attacks.
- Expected number of steps to solve: $m \times 2^{k-1}$.

**Why not use $k+\log m$ bit puzzles?**

- $(k+\log m)$-bit puzzle
  - Expected number of trials $m \times 2^{k-1}$

- But for random guessing attacks, the successful probability
  - One $(k+\log m)$-bit puzzle
    - $2^{k+\log m}$ (e.g., $2^{8k}$)
  - $m$ $k$-bit subpuzzles
    - $(2^k)^m = 2^{km}$ (e.g., $2^{8k}$)
## Puzzle Properties

- Puzzles are stateless
- Puzzles are easy to verify
- Hardness of puzzles can be carefully controlled
- Puzzles use standard cryptographic primitives

## A Possible Way to use Client Puzzle

### Client puzzle protocol (normal situation)

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_i ), “Puzzle?”</td>
<td>( )</td>
</tr>
<tr>
<td>“No puzzle.”</td>
<td>Registers permission of ( M_i )</td>
</tr>
<tr>
<td>( M_i )</td>
<td>( )</td>
</tr>
</tbody>
</table>

\( M_i^j \) : first message of \( i\)-th execution of protocol \( M \)

### Client puzzle protocol (under attack)

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_i ), “Puzzle?”</td>
<td>( )</td>
</tr>
<tr>
<td>“Yes, puzzle.”, ( P, t )</td>
<td>( )</td>
</tr>
<tr>
<td>( )</td>
<td>Verifies that ( t_i \leq t ) ( )</td>
</tr>
<tr>
<td>( )</td>
<td>Computes ( k \leftarrow \text{PRG}(M_i) ) ( )</td>
</tr>
<tr>
<td>( )</td>
<td>Verifies that solution is correct ( )</td>
</tr>
<tr>
<td>( )</td>
<td>Registers permission of ( M_i ) ( )</td>
</tr>
<tr>
<td>( M_i )</td>
<td>( )</td>
</tr>
</tbody>
</table>
Client Puzzle Based on Special Image of Crypto Hash Functions

Puzzle Construction

- $C \rightarrow S$: Hello
- $S \rightarrow C$: $N_S$
- $C \rightarrow S$: $C$, $N_C$, $X$
- $S$: verify $h(C, N_C, N_S, X)$ has $k$ leading zeros

Expected Cost of Finding a Puzzle Solution

- Given puzzle strength $k$, the probability of finding a solution after $x$ trials:
  
  $$P_{x,k} = 1 - (1 - 2^{-k})^x$$

- Expected number of trials to find a solution is $2^k$
Client Puzzle based on Constrained Discrete Logarithm Problem

Context
– Client puzzle outsourcing for DoS resistance

Motivation
– Client puzzle mechanism can become the target of DoS attacks
  • Servers have to validate solutions which require resources
– Puzzles must be solved online
  • User time is more important than CPU time

Properties of the Solution
• The creation of puzzles is outsourced to a secure entity, the bastion
  – Create puzzle with no regard to which server is going to use them
• Verifying puzzle solutions is a table lookup
• Clients can solve puzzles offline ahead of time
• A puzzle solution gives access to a virtual channel for a short time period
C: A group of prime numbers with generator $g$

Pick $r_{c,t} \in \mathbb{Z}_q$

Let $g_{c,t} = g^{r_{c,t}}$, puzzle $x_{c,t} = (g_{c,t}, f'(a))$

Enumerate $i$ values to solve $a_{c,t}$

Solution is $\sigma_{c,t} = Y_{1}^{f'(a)}$

Take the easy way

$\sigma_{c,t} = g_{c,t}^{X_1}$

---

**Puzzle Properties**

- **Unique puzzle solutions**
  - Each puzzle has a unique solution
- **Per-channel puzzle distribution**
  - Puzzles are unique per each (server, channel, time period) triplet
- **Per-channel puzzle solution**
  - If a client has a solution for one channel, he can calculate a solution for another server with the same channel easily
Secret Sharing

- **Objective**
  - Divide data $D$ into $n$ pieces $D_1, \ldots, D_n$ in such a way that
    - Knowledge of any $k$ or more $D_i$ pieces makes $D$ easy to compute,
    - Knowledge of any $k - 1$ or fewer $D_i$ pieces leaves $D$ completely undetermined.
  - Such a scheme is called a $(k, n)$ threshold scheme.

- **Useful when no single entity can be trusted with the secret**
  - Management of cryptographic keys

Shamir’s Secret Sharing

- **Underlying fact**
  - Based on polynomial interpolation.
  - Given $k$ points in the 2-d plane $(x_1, y_1), \ldots, (x_k, y_k)$ with distinct $x_i$’s,
  - there is one and only one polynomial $q(x)$ of degree $k - 1$ such that
    
    $$q(x_i) = y_i \text{ for all } i.$$
Shamir’s Secret Sharing (Cont’d)

- Split the secret D
  - To divide D into pieces D_i …
  - Pick a random \( k-1 \) degree polynomial
    \[ q(x) = a_0 + a_1 x + \ldots + a_{k-1} x^{k-1} \]
    in which \( a_0 = D \).
  - Evaluate \( D_1 = q(1), D_2 = q(2), \ldots, D_n = q(n) \).

- The secret shares represent distinct points on the polynomial.

Shamir’s Secret Sharing (Cont’d)

- Reconstruction
  - Given any subset of \( k \) of these \( D_i \) values (with their identifying indices)
    • Find the coefficients of \( q(x) \) by interpolation,
    • Evaluate \( D = q(0) \).
  - Given just \( k-1 \) of these values,
    • \( D \) could be any value
    • In other words, \( D \) being any value will give one and only one possible polynomial
  - Alternatively, view these as linear equations.
Motivation

- IDA was developed to provide safe and reliable transmission of information in distributed systems.

- Inefficiency of retransmission of lost packets
  - In multicast transmission, different receivers lose different sets of packets.
  - Re-request and retransmission increases delays.

- Forward error correction technique might be desirable in distributed systems.

High-level Operations

- Dispersal($F$, $m$, $n$):
  - Split input $F$ with redundancy into $n$ pieces $F_i (1 \leq i \leq n)$.
  - $|F_i|=|F|/m$, and $m \leq n$

- Recovery($\{F_i \mid 1 \leq j \leq m\}, 1 \leq i \leq n, m, n$):
  - Reconstruct $F$ from any $m$ out of the $n$ pieces ($F_i (1 \leq i \leq n)$)

Dispersal($F$, $m$, $n$) – Example 1

- $|F|=32$ bytes, $m=4$, $n=8$

  - $|F_i|=32/4=8$ bytes ($1 \leq i \leq m$)
Recovery($\{F_i \mid (1 \leq j \leq m), (1 \leq i \leq n)\}, m, n$) – Example 2

• $|F| = 32$ bytes, $m = 4$, $n = 8$, $|F| = 8$ bytes ($1 \leq i \leq 8$

• Assume the following 4 ($= m$) pieces are received.

\[
\begin{align*}
F_1 & \quad F_2 & \quad F_3 & \quad F_4 \\
\downarrow & & & \\
\text{Recovery}(F_1, F_3, F_4, 4, 8) & & & \\
\end{align*}
\]

Dispersal($F, m, n$)

• $F = b_1, b_2, \ldots, b_N$
  
  - $N = |F|$, and $b_i$ represents each byte in $F$ ($0 \leq b_i \leq 255$).
  
  - All computations performed in $\mathbb{GF}(2^8)$.
    - $\mathbb{GF}(2^8)$ is closed under addition and multiplication.
    - Every nonzero element in $\mathbb{GF}(2^8)$ has a multiplicative inverse.

• $F = (b_1, b_2, \ldots, b_m), (b_{m+1}, \ldots, b_{2m}), \ldots, (b_{N-m+1}, \ldots, b_N)$
  
  - $S_i = (b_{i-m+1}, \ldots, b_i)$ ($1 \leq i \leq N/m$)

• The matrix $M_{n \times m}$ is constructed as follows:
  
  $M = [S_1, S_2, \ldots, S_{N/m}]$

Dispersal($F, m, n$)

• The matrix $A_{n \times n}$ is constructed as follows:

\[
A = \begin{bmatrix}
    a_1 \\
    a_2 \\
    \vdots \\
    a_n
\end{bmatrix}
\]

- $a_i = (a_{i1}, \ldots, a_{in})$ ($1 \leq i \leq n$)
  
  - Every subset of $m$ different vectors should be linearly independent.
Dispersal$(F, m, n)$

- The following Vandermonde matrix satisfies the property required for $A$:

\[
\begin{bmatrix}
1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\
1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\
1 & x_3 & x_3^2 & \cdots & x_3^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_n & x_n^2 & \cdots & x_n^{n-1}
\end{bmatrix}
\]

- $m \leq n$, and all $x_i$'s are nonzero elements in $\text{GF}(2^8)$ and pairwise different.
- Any $m$ different rows are linearly independent, so any matrix composed of a set of any $m$ different rows is invertible.

\[A \cdot M = [S_1 \ S_2 \ \cdots \ S_{m \times n}] \begin{bmatrix}
a_1 \\
a_2 \\
\vdots \\
a_m \\
\end{bmatrix} = [F_1 \ F_2 \ \cdots \ F_m]
\]

where $a_i \cdot S_k = a_i b_{k(i-1)+1} + \cdots + a_i b_{km}$

\[|F|=32 \text{ bytes, } m=4, n=8 \]
- $F = b_1, b_2, \ldots, b_{32}$
- Represented as $M_{4 \times 8}$

\[
M = \begin{bmatrix}
S_1 & S_2 & \cdots & S_8
\end{bmatrix}
\begin{bmatrix}
b_1 & b_2 & \cdots & b_{32} \\
b_2 & b_3 & \cdots & b_{31} \\
b_3 & b_4 & \cdots & b_{30} \\
b_4 & b_5 & \cdots & b_{29}
\end{bmatrix}
\]
Dispersal($F, m, n$) – Example 3

- $A_{8\times 4}$

\[ A = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_8 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_8 & x_8^2 & x_8^3 \end{bmatrix} \]

Dispersal($F, m, n$) – Example 3

- $F_i$ (1 ≤ $i$ ≤ 8) are computed as follows:

\[ A \cdot M = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_8 \end{bmatrix} \begin{bmatrix} S_i & S_i & \ldots & S_i \end{bmatrix} = \begin{bmatrix} a_1 \cdot S_i & a_2 \cdot S_i & \ldots & a_8 \cdot S_i \end{bmatrix} \begin{bmatrix} F_i \\ F_i \\ \vdots \\ F_i \end{bmatrix} = \begin{bmatrix} F_i \\ F_i \\ \vdots \\ F_i \end{bmatrix} \]

Recovery({$F_i$} | (1 ≤ $i$ ≤ $m$), (1 ≤ $j$ ≤ $n$)), $m, n$)

- Given $m$ pieces $F_i$ (1 ≤ $f$ ≤ $m$), (1 ≤ $j$ ≤ $n$),

\[ \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_m \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \cdot M = A' \cdot M \]

- $M$ can be recovered from the given $m$ pieces $F_i$ (1 ≤ $f$ ≤ $m$), (1 ≤ $j$ ≤ $n$) because $A'$ is invertible.

\[ \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_m \end{bmatrix} = M \]

\[ \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_m \end{bmatrix} = M \]
Example 4

The original data \( M \) can be recovered by the following computation:

\[
\begin{bmatrix}
F_1 \\
F_2 \\
F_3 \\
F_4 \\
F_5
\end{bmatrix} = \begin{bmatrix}
a_1 \\
a_3 \\
a_4 \\
a_7
\end{bmatrix}^{-1} \begin{bmatrix}
a_1 \\
a_3 \\
a_4 \\
a_7
\end{bmatrix} \cdot M
\]

\( |F| = 32 \text{ bytes, } m = 4, n = 8 \)

In example 3, \( F_i \) (1 \( \leq i \leq 8 \)) pieces of 8 bytes are resulted.

Assume that \( \{ F_1, F_3, F_4, F_7 \} \) are received among them.

\[
\begin{bmatrix}
F_1 \\
F_2 \\
F_3 \\
F_4 \\
F_5
\end{bmatrix} = \begin{bmatrix}
a_1 \cdot S_1 & a_1 \cdot S_2 & \ldots & a_1 \cdot S_8 \\
a_3 \cdot S_1 & a_3 \cdot S_2 & \ldots & a_3 \cdot S_8 \\
a_4 \cdot S_1 & a_4 \cdot S_2 & \ldots & a_4 \cdot S_8 \\
a_7 \cdot S_1 & a_7 \cdot S_2 & \ldots & a_7 \cdot S_8
\end{bmatrix}
\]

\( \cdot M \)
Goals

• Authenticate without revealing credentials
  – Consider two groups $G_1$ and $G_2$
  – Two parties $A \in G_1$ and $B \in G_2$. $A$ and $B$ wants to authenticate each other.
  – If $G_1 \neq G_2$: $A$ and $B$ only know they are not in the same group.
  – If $G_1 = G_2$: $A$ and $B$ can authenticate to each other.
  – A third party learns nothing by observing conversations between $A$ and $B$.

Preliminaries: Pairing-based Cryptography

• Bilinear Maps:
  – Two cyclic groups of large prime order $q$: $G_1$ and $G_2$
  – $\hat{\cdot}: G_1 \times G_1 \rightarrow G_1$ is a bilinear map if
    \[ \forall a, b \in \mathbb{Z}_q; P, Q \in G_1; \hat{\cdot}(aP, bQ) = \hat{\cdot}(P, Q)^{ab} \]
  – $\hat{\cdot}$ should be computable, non-degenerate and satisfies Bilinear Diffie-Hellman assumption, i.e., given $P, aP, bP, cP$, it is hard to compute $\hat{\cdot}(P, P)^a$

Protocol Sketch

• Equipped with bilinear map $\hat{\cdot}$ and one-way hash function $H_1$
• CA has a master key $t$.
• Assume a driver-and-cop scenario.
Protocol Sketch

\[ K_A = h(H("xy6542678d-cop"), T_A), \quad K_B = h(T_A, H("p65748392a-driver")) \]

Driver’s License: 
"p65748392a", \( T_A \)

Traffic cop credential: 
"xy6542678d", \( T_B \)

\( T_A = h_A(H("p65748392a-driver")) \)
\( T_B = h_B(H("xy6542678d-cop")) \)

Protocol Sketch – Attacker Igor

\[ K_A = h(H("xy6542678d-cop"), T_A) \]

Driver’s License: 
"p65748392a", \( T_A \)

Obtains Bob’s pseudonym 
"xy6542678d"

\( T_A = h_A(H("p65748392a-driver")) \)

Secret-Handshake Scheme (SHS)

- SHS.CreateGroup(G): executed by an administrator, generates the group secret GroupSecretG for G.
- SHS.HandShake(A, B): Users A and B authenticate each other. B discovers A ∈ G if and only if A discovers B ∈ G.
- SHS.TraceUser: Administrator tells the user from a transcript T generated during conversation between A and B.
- SHS.RemoveUser: Administrator revokes user U.
Pairing-Based Handshake (PBH)

- **PBH.CreateGroup**: Administrator sets GroupSecret, as a random number $s_i \in \mathbb{Z}_q$.
- **PBH.AddUser**: Administrator generates pseudonyms for users:
  
  $$[\text{id}_1, \ldots, \text{id}_n]$$

  and then generates the corresponding secret points:
  
  $$\{\text{priv}_{i1}, \ldots, \text{priv}_{in}\}$$

  where
  
  $$\text{priv}_{i1} = s_i H_i(\text{id}_{i1})$$

  $H_i$ is a one-way hash function.

Pairing-Based Handshake (PBH)

- **PBH.Handshake**:
  
  $A \xrightarrow{\text{id}_A, n_A} B$
  
  $A \xrightarrow{\text{id}_A, n_A, V_3} B$
  
  $A \xrightarrow{V'_i} B$

  
  $$V'_i = H_i(\hat{\epsilon}(H_i(\text{id}_A), \text{priv}_A)) | \text{id}_A | n_A | 0$$
  
  $$V'_i = H_i(\hat{\epsilon}(\text{priv}_A, H_i(\text{id}_A))) | \text{id}_A | n_A | 1$$
  
  $$S = H_i(\hat{\epsilon}(|\text{priv}_A, H_i(\text{id}_A))) | \text{id}_A | n_A | n_2 | 2$$

  
  $$= H_i(\hat{\epsilon}(H_i(\text{id}_A), \text{priv}_A)) | \text{id}_A | n_A | 0 | n_2 | 2$$

Pairing-Based Handshake (PBH)

- **PBH.TraceUser**: Since the conversations of handshaking include the pseudonyms, administrator can easily figure out the users.
- **PBH.RemoveUser**: Administrator removes user $U$ by broadcasting its pseudonyms to all the other users, so that other users won’t accept pseudonyms of $U$. 
Computational Diffie-Hellman Instead of Bilinear Diffie-Hellman

- **CreateGroup:** Administrator picks \((p, q, g)\), \(p\) and \(q\) are primes, \(g\) is a generator of a subgroup in \(\mathbb{Z}_p^*\) of order \(q\). Also, she picks up a private key \(x\), and computes the public key \(y = g^x \mod p\).

- **AddUser:** For user \(U\), administrator generates \(id_U\), and then generates a pair \((w, t)\) \(\in (\mathbb{Z}_p^*, \mathbb{Z}_q)\) so that \(g^t = wy^{D(w, ID)}\). \(id_U, w, t\) will be given to the user.

- **How to generate the pair \((w, t)\)?**
  - Randomly pick \(r\), compute \(w = g^r \mod p\)
  - \(t = r + xH(w, ID)\)

- **Handshake:** Assume user \(A\) has \((id_A, w_A, t_A)\) and user \(B\) has \((id_B, w_B, t_B)\).

- **Define notation (ElGamal Encryption):**
  - \(\text{Recover}(y, id, w) = PK = wy^{H(y||id) \mod p}\)
  - \(\text{Enc}_{id_m}(m) = [c_1, c_2] = [g^e \mod p, m \oplus H'(PK^e \mod p)]\)
  - \(\text{Dec}_{id_m}([c_1, c_2]) = m \oplus H'(c_1 \mod p)\)
Computational Diffie-Hellman Instead of Bilinear Diffie-Hellman

- **Handshake:**

  \[
  PK_A = \text{Recover}(y, id_A, w_A) \\
  \text{randomly picks } r_A, \text{ch}_A \\
  C_A = \text{Enc}_{y_A}(r_A) \\
  \text{verifies } \text{resp}_A \\
  r_A = \text{Dec}_{y_A}(C_A) \\
  \text{resp}_A = H(r_A, r_B, \text{ch}_B) \\
  \text{verifies } \text{resp}_A
  \]

- **Intuition:**

  - If A and B are in the same group, each of them can decrypt the random number \( (r_A \text{ and } r_B) \).
  - If not, neither of them can get any information about \( r_A \text{ or } r_B \).