

Topic 7. Broadcast and Multicast Key Distribution and Authentication (chapter 8 in the text) (cont.)

- 1 **Basic** Models for Multicast Key Distribution
- 2 **Logic Key Tree** Based Multicast Key Distribution
- 3 **Hash Chain** Based Authentication
- 4 Merkle Trees for **Authentication**

2. Logic Key Tree Based Multicast Key Distribution (cont.)

PROTOCOL 3 (REKEYING FOR A JOIN)

Assume that $n < 2^h$. The **KMD server** performs the following operations.

- 1 **Run the MAKE** protocol with entity u_n and establish an individual key of u_n , denoted as k_n . **Find a leave vertex** with the smallest depth, say h' . Make this vertex as a **parent vertex** of the two leaves to which the entities u_i and u_n are attached.
- 2 Generate a key set for u_n from a PRNG, denoted as

$$K'(u_n) = \{k_n, k'_{s,j_s}, k'_{s+1,j_{s+1}}, \dots, k'_{h-1,j_{h-1}}, r'\}$$

where k'_{s,j_s} is a parent key of u_i and u_n .

- 3 **Encrypt** k'_{s,j_s} using their individual keys of u_i and u_n and send to them.
- 4 **Execute Step 4 in Protocol 2** in a way that u_i were treated as leaving the network.

Performance Evaluation

Property

The parameter set of Protocol Π is given by

- n users, and $h = \lceil \log n \rceil$.
- Each user has $h + 1$ keys.
- $N_{init} = hn$, $N_{leave} = 2h - 2$, $N_{join} = 2h$ where N_{init} , N_{leave} , and N_{join} are the maximum numbers of messages that KMD will send to entities in the initial key distribution phase and the re-keying phase for a leave or a join, respectively.

Comparisons of Protocols Θ and Π (Protocols 1-3)

Topology	Protocol	$R = K(u_i) $	N_{ini}	N_{leave}	N_{join}	Robust to compromising keys
Star	Θ	2	n	$n - 1$	$n + 1$	No
LKT	Π	$\leq \lceil \log n \rceil + 1$	$\leq \lceil \log n \rceil n$	$\leq 2 \lceil \log n \rceil - 2$	$\leq 2 \lceil \log n \rceil$	Yes

3. Hash Chain Based Authentication

Problem

- (a) A server A broadcasts n messages (x_1, x_2, \dots, x_n) at n different time instances. Design a scheme for which each receiver can efficiently authenticate x_i with modest computation cost and memory requirement.
- (b) Given n messages (x_1, x_2, \dots, x_n) generated by one entity or n entities where each entity responds to generate one message, design an algorithm by which a server A or system A can efficiently authenticate a randomly chosen x_i , but the system does not hold a copy of x_i .

Conditions

We assume that entity A 's security association contains a hash function h , an MAC (message authentication code), digital signature scheme Sig , and related crypto schemes.

Hash chains

- Entity A picks an initial value y , computes

$$k_i = h^{n-i}(y), i = 0, 1, \dots, n-1, \quad (1)$$

and the end point k_0 is either signed by A as $Sig_A(k_0)$ or k_0 is committed by A for which any intended verifier knows this commitment and accepts as an authenticated value.

- The vector

$$(Sig_A(k_0), k_0, k_1, \dots, k_{n-1}) \quad (2)$$

is called a **hash chain** with length n .

Property

If hash function h is a one-way function or with collision resistance, then each key, say k_i , in the hash chain, defined by (1), is authenticated by the end point k_0 in the following fashion.

- 1 A verifier checks whether the digital signature of A on k_0 is true. If so, continue the next step. Otherwise, declare a failure.
- 2 Authenticate k_i by checking the validity of $k_{i-1} = h(k_i)$ for $i = 1, \dots, n - 1$.

Hash Chain Based Message Authentication: A Solution to Problem (a)

- Entity A prepares a hash chain $\{k_i\}$ with length $n + 1$:
 $k_i = h^{n+1-i}(y), i = 0, 1, \dots, n$ for authenticating n messages x_1, \dots, x_n .
- The authentication tag of message x_i is $MAC(k_i, x_i)$ where k_i is served as a key for MAC .

- 1 **At time instance** t_0 , the entity sends $(k_0, \text{Sig}_A(k_0))$. A receiver verifies the digital signature of A on k_0 .
- 2 **At time instance** t_1 : $t_0 < t_1$, the entity A sends $(x_1, \text{MAC}(k_1, x_1))$, and at time instance $t_1 + \Delta t$, A releases k_1 . Then a receiver first confirms whether

$$k_0 = h(k_1). \quad (3)$$

If this is true, the receiver verifies the authentication tag $\text{MAC}(k_1, x_1)$. If it is successful, then the receiver accepts x_1 and stores k_1 .

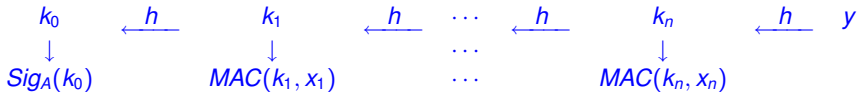
- 3 Assume that the receiver has successfully verified i messages and stored (k_0, k_{i-1}) . **At time instances** t_j : $t_0 < t_1 < \dots < t_j$, and $t_j + \Delta$ where $t_j > t_{j-1} + \Delta$, the entity A sends $(x_j, \text{MAC}(k_j, x_j))$ and releases k_j respectively. After receiving k_j , the receiver first confirms whether

$$k_{i-1} = h(k_j). \quad (4)$$

Since k_{i-1} has been authenticated, the validity of (4) results in an authentication of k_j . Then the receiver verifies the authentication tag $\text{MAC}(k_j, x_j)$.

- 4 **Continue this process** till finishing the authentication of message x_n . After used up all the values in the hash chain, the entity needs to produce a new hash chain for next n messages.

The process for **computing an authentication tag** for each message x_i :



The time differences for sending messages and releasing keys:

t_0	t_1	$t_1 + \Delta$	t_2	$t_2 + \Delta$	\dots	t_n	$t_n + \Delta$
k_0	x_1		x_2		\dots	x_n	
$Sig_A(k_0)$	$MAC(k_1, x_1)$		$MAC(k_2, x_2)$			$MAC(k_n, x_n)$	
		k_1		k_2	\dots		k_n

Hash Chain Based Access Authentication

This is a scenario of Problem (b).

There are three possible ways an attacker could learn the user's secret password and then impersonate him when interacting with the system:

- (1) By **intercepting** the user's communication with the system, e.g., eavesdropping on the line connecting the user's terminal with the system, or observing the execution of the password checking program.
- (2) By **gaining access** to the information stored inside the system, e.g., reading the system's password file.
- (3) By the user's inadvertent disclosure of his password, e.g., choosing an easily guessed password.

The third possibility cannot be prevented by any password protocol!
(why?)

A countermeasure for the first two attacks.

Hash Chain Based Password Authentication

- 1 A user generates a hash chain, stores k_0 in the system (committed to the system), and uses k_i as his i th password, $i = 1, \dots, n - 1$. In other words, the user will use different password to remote identify himself to the system at different time.
- 2 At time t_1 , the user uses k_1 as his password, the system verifies whether $k_0 = h(k_1)$.
- 3 Assume that the user has used the first $i - 1$ passwords and stored (k_0, k_{i-1}) . At time instance t_i : $t_0 < t_1 < \dots < t_i$, he sends k_i as his new password, and the previous used passwords are expired. Upon receiving it, the system verifies whether $k_{i-1} = h(k_i)$, and grant an access if it is successful.

Once all $n - 1$ passwords associated with the initial value y are used up, the user needs to generate a new hash chain, and submit the new end point value again to the system.

4. Merkle Trees for Authentication

- **The second solution** to Problem (b): Merkle trees.
- Let B be a set consisting of a group of users, which will be worked in a cooperated way. The n data $\{x_0, x_1, \dots, x_{n-1}\}$ are generated by B , and they will be authenticated by A .

Generation of Merkle Tree

- 1 The party B generates a complete tree with height $h = \lceil \log n \rceil$. The n leaves are ordered from left to right, and hash value $h(x_i)$ is attached to vertex leaf i .
- 2 Each parent vertex is attached with a hash value where an input to h is the concatenation of the hash values from two children vertexes, i.e.,

$$a_{i,j} = h(v_{left} || v_{right}) \quad (5)$$

where $x || y$ means the concatenation of x and y . This tree is referred to as a *Merkle tree*.

- 3 B submits the root value r to server A by either signing r or by a protected way.

System A authenticates any randomly chosen x_i

- 1 Find a path from the leaf x_i to the root. Let $Auth(x_i)$ be the set consisting of the vertex values in the path. We may assume that

$$Auth(x_i) = \{a_i, a_{1,j_1}, \dots, a_{h-1,j_{h-1}}, r\}. \quad (6)$$

Let $Sib(x_i)$ denote the set consisting of all siblings of the vertexes in $Auth(x_i)$ whose elements are given by

$$Sib(x_i) = \{a_t, a_{1,l_1}, \dots, a_{h-1,l_{h-1}}\} \quad (7)$$

where a_t is the sibling of a_i .

- 2 A requests B to submit all the sibling values given by (7). After receiving those values, A iteratively computes the following hash chain:

$$\begin{aligned} a_i &= h(x_i) \\ a_{1,j_1} &= h(a_i || a_t) \\ a_{2,j_2} &= h(a_{1,j_1} || a_{1,l_1}) \\ &\vdots \\ d &= h(a_{h-1,j_{h-1}} || a_{h-1,l_{h-1}}) \end{aligned}$$

Check whether $d = r$. If so, accept message x_i is from the party B.

A Merkle Tree of 8 Messages

