

Solutions to Assignment 5

Xinxin Fan

Solutions

1. (a) When $n = 7$, KDM server generates a logic key tree as shown in the following Figure 1.

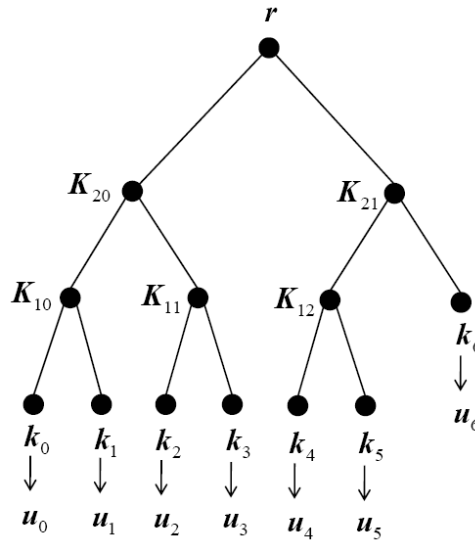


Figure 1: An LKT for a Group with Seven Entities

The key set for each entity is as follows:

$$\begin{aligned}
 K(u_0) &= \{k_0, K_{10}, K_{20}, r\}, & K(u_1) &= \{k_1, K_{10}, K_{20}, r\}, \\
 K(u_2) &= \{k_2, K_{11}, K_{20}, r\}, & K(u_3) &= \{k_3, K_{11}, K_{20}, r\}, \\
 K(u_4) &= \{k_4, K_{12}, K_{21}, r\}, & K(u_5) &= \{k_5, K_{12}, K_{21}, r\}, \\
 K(u_6) &= \{k_6, K_{21}, r\}.
 \end{aligned}$$

- (b) In the initial phase, KMD server prepares the following ciphertext for the entity u_6 :

$$\begin{array}{|l}
 c_1 = E_{k_6}(K_{21}) \\
 c_2 = E_{K_{21}}(r)
 \end{array}$$

- (c) Upon receiving (c_1, c_2) , the entity u_6 performs the following successive decryption using his individual key k_6 .

$$\begin{array}{|l}
 K_{21} = D_{k_6}(c_1) \\
 r = D_{K_{21}}(c_2)
 \end{array}$$

2. (a) Assuming that entity u_0 leaves the network, KMD server will perform the following rekeying process as described in Protocol 2:

(1) As shown in Figure 1, a path from u_0 to the root gives the key set of u_0 as below

$$K(u_0) = \{k_0, K_{10}, K_{20}, r\}.$$

(2) Find the sibling of u_0 , which is u_1 . KMD server attaches u_1 to its parent vertex $(1, 0)$, assigns the individual key k_1 of u_1 to the vertex, and removes the parent key K_{10} .

(3) KMD server generates $\{K'_{20}, r'\}$ for updating the parent keys $\{K_{20}, r\}$ in $K(u_0)$ except for K_{10} since that vertex is changed to a leaf vertex.

(4) *Paired encryption and multicast subsets:*

(i) Encrypt the new parent key K'_{20} using the individual key k_1 of u_1 and its sibling key K_{11} , i.e., compute $c_1 = E_{k_1}(K'_{20})$ and $c_2 = E_{K_{11}}(K'_{20})$, and multicast c_1 to $\{u_1\}$ and c_2 to $\{u_2, u_3\}$.

(ii) Find the sibling of the vertex $(2, 0)$, which is the vertex with index $(2, 1)$. Their parent is the root. Encrypt their new root key using the newly update key of the vertex $(2, 0)$ and the key of its sibling $(2, 1)$, i.e., compute $c_3 = E_{K'_{20}}(r')$ and $c_4 = E_{K_{21}}(r')$, and multicast c_3 to $\{u_1, u_2, u_3\}$ and c_4 to $\{u_4, u_5, u_6\}$.

The ciphertexts and multicast subsets are shown in the following diagram.

Vertex and its sibling	Paired ciphertext	Multicast set
(1, 0)	$c_1 = E_{k_1}(K'_{20})$	$\{u_1\}$
(1, 1)	$c_2 = E_{K_{11}}(K'_{20})$	$\{u_2, u_3\}$
(2, 0)	$c_3 = E_{K'_{20}}(r')$	$\{u_1, u_2, u_3\}$
(2, 1)	$c_4 = E_{K_{21}}(r')$	$\{u_4, u_5, u_6\}$

The siblings of the vertexes in the path from u_1 to the root are shown in Figure 2 in framed boxes. The vertex keys, framed by circles and boxes in Figure 2, are the keys involved in the rekeying process when entity u_0 leaves the network.

(b) u_1 will receive two ciphertexts c_1 and c_3 . Moreover, u_1 and u_6 must update the keys $\{K_{20}, r\}$ and $\{r\}$, respectively, since these keys are in the key set of the leaving entity u_0 .

(c) If u_0 leaves at time instance a and the KDM does not execute the rekeying protocol until the time instance b , the leaving entity u_0 still can decrypted the message which is encrypted by the old multicast key r of the group at time instance t ($a < t < b$). Therefore, to provide the confidentiality in the system, the KDM must perform the rekeying process immediately when some nodes leave the group.

(d) After the rekeying process for u_0 leaving, the LKT is changed to a new tree, shown in the following Figure 3.

(e) Assume that a new user, denoted by u_7 , will join the system after the rekeying process for u_0 's leaving. The KMD server will first perform the following rekeying process as described in Protocol 3 for u_7 's joining:

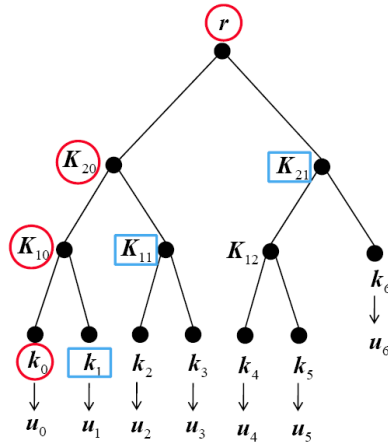


Figure 2: The vertex keys are involved in the rekeying process

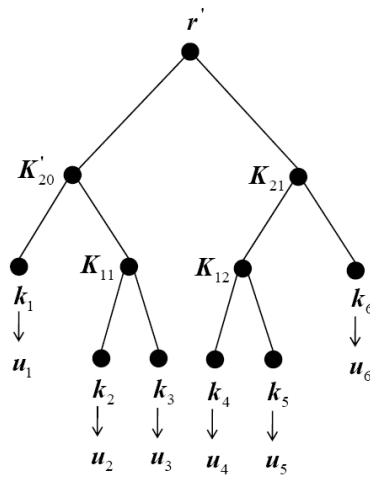


Figure 3: A new LKT after the rekeying process

- (1) From the LKT shown in Figure 3, u_1 is the first leaf with depth 2 from left to right order. Make the vertex where u_1 is attached as a parent vertex with index $(1, 0)$, and attach u_1 and u_7 as its children.
- (2) The original key set of u_1 is given by

$$K(u_1) = \{k_1, K'_{20}, r'\}.$$

- (3) The KMD server generates a key set for the new entity u_7 , denoted as

$$K(u_7) = \{k_7, K''_{10}, K''_{20}, r''\},$$

where k_7 is the individual key of the new entity u_7 and K''_{11} is a new parent key of entities u_1 and u_7 .

- (4) Compute $c_1 = E_{k_1}(K''_{10})$ and $c_1 = E_{k_7}(K''_{10})$, and send c_1 to u_1 and c_2 to u_7 . The key set of u_1 needs to be updated to

$$K(u_1) = \{k_1, K''_{10}, K''_{20}, r''\}.$$

The required updates are

$$\{K'_{20}, r'\} \longrightarrow \{K''_{20}, r''\}.$$

The KMD server runs Step 4 in Protocol 2 in a way that u_1 is treated as leaving the network for which the key set of u_7 is virtually defined as $K(u_7) = \{k_7, K''_{10}, 0, 0\}$ and u_1 is included in any subgroups of which u_7 belongs for the updates. the details of the updates are illustrated in the following diagram.

KMD	Multicast set
$c_3 = E_{K''_{10}}(K''_{20})$	$\{u_1, u_7\}$
$c_4 = E_{K_{11}}(K''_{20})$	$\{u_2, u_3\}$
$c_5 = E_{K''_{20}}(r'')$	$\{u_1, u_7, u_2, u_3\}$
$c_6 = E_{K_{21}}(r'')$	$\{u_4, u_5, u_6\}$

After the join of u_7 , the LKT becomes a new LKT, as shown in Figure Figure 4.

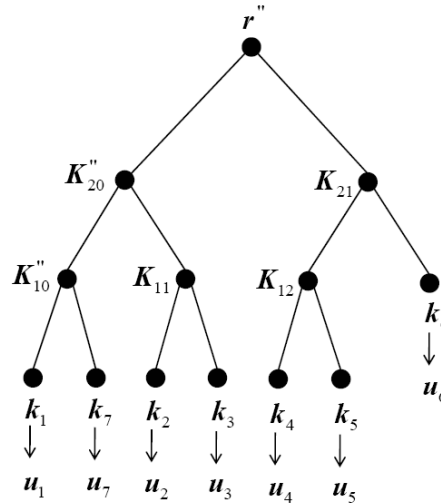


Figure 4: A new LKT after the join of the entity u_7

3. To authenticate 100 messages x_1, x_2, \dots, x_{100} , an entity A first generates a hash chain $\{k_0, k_1, \dots, k_{100}\}$ with length 101, i.e., we have $k_i = h^{101-i}(y)$ for $i = 0, 1, \dots, 100$. Then A will perform the following steps:

- (1) At time instance t_0 , the entity A signs the anchor k_0 of the hash chain and 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 $k_0 = h(k_1)$. If it is, 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 $i - 1$ messages and stored (k_0, k_{i-1}) . At time instances $t_i : t_0 < t_1 < \dots < t_i$, and $t_i + \Delta$ where $t_i > t_{i-1} + \Delta$, the entity A sends $(x_i, \text{MAC}(k_i, x_i))$ and releases k_i , respectively. After receiving k_i , the receiver first confirms whether $k_{i-1} = h(k_i)$. Since k_{i-1} has been authenticated, the validity of $k_{i-1} = h(k_i)$ implies the authenticity of k_i . Then the receiver verifies the authentication tag $\text{MAC}(k_i, x_i)$.
- (4) Continue this process until finishing the authentication of the message x_{100} . At this time point, all hash values in the hash chain are used up.
 - (a) If the key used in the generation of an authentication tag of the i th message is transmitted at the same time instance when the message and its authentication tag are transmitted, an attacker can intercept the key k_i and replace it by his key k'_i . In this way, the attacker can generate an valid authentication tag for any message he chooses.
 - (b) In modern computer networks, a centralized server or some number of computers can act as primary time servers to synchronize a much larger number of secondary servers and clients. In order to do this, a distributed network clock synchronization protocol is required which can read a server clock, transmit the reading to one or more clients and adjust each client clock as required. Protocols that do this include the Network Time Protocol (NTP), Digital Time Synchronization Protocol (DTSS) and many others proposed in the literature.