Simple and Efficient Time-Bound Hierarchical Key Assignment Scheme (Short Paper)

Naveen Kumar, Anish Mathuria, and Manik Lal Das

DA-IICT, Gandhinagar, India
{naveen_kumar,anish_mathuria,maniklal_das}@daiict.ac.in

Abstract. We propose a simple and efficient hash-based time-bound hierarchical key assignment scheme that requires a single key per user per subscription. It is more efficient than existing schemes in terms of public storage.

Keywords: Time-bound access, Hierarchical access control, Key management.

1 Introduction

There exist many on-line applications where data access is restricted to a specific time interval. The total system time is divided into distinct *time slots*. A time slot is the smallest possible subscription period. Each time slot has a set of data blocks associated with it. A contiguous sequence of time slots is called a *time interval*. A user can subscribe for one or more time intervals (subscription intervals).

A monthly *subscription hierarchy* with 3 time slots (one for each January, February and March) is shown in Figure 1. For each possible subscription interval, there is a distinct node in the hierarchy. Each leaf node represents a time slot. A directed edge in the subscription hierarchy represents an access to a subscription interval with a subsequence of time slots. For example, an edge from node $Jan - Mar$ to $Jan - Feb$ implies that a user with subscription of node $Jan - Mar$ can access to node $Jan - Feb$.

To cryptographically enforce time-bound access control, each node in the subscription hierarchy is assigned a key using appropriate Hierarchical Key Assignment Scheme ($HKAS$) [1]. Resources associated with a time slot are encrypted with its associated encryption key. User is given secret information using which one can easily compute all authorized encryption keys. The secret information must be distributed securely to the user by a trusted Central Authority (CA). An edge in the hierarchy represents direction of key derivation. A user subscribed for the month of Jan and Feb is given a single secret information $K_{Jan-Feb}$ through which he can compute encryption keys K_{Jan} and K_{Feb} for the nodes Jan and Feb respectively.

When designing an efficient scheme, the objective is to minimize the amount of secret key storage, public storage and key derivation time. Tzeng [2] proposed

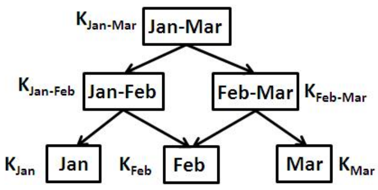

Fig. 1. An example subscription hierarchy

the first time-bound $HKAS$ based on RSA [3] cryptosystem. However, Yi and Ye [4] found a three party collusion attack on [2]. In a survey paper on time-bound $HKAS$, Zhu et al [5] suggest that hash-based constructions are more promising than expensive modular exponentiation based constructions.

Ateniese, Santis, Ferrara and Masucci [6][7] proposed two constructions, one based on bilinear pairing and another based on symmetric encryption. First requires large key storage (i.e. $O(z)$ where z is the number of time slots) to a user per subscription. Second uses worst case two-level structure to reduce key derivation cost at the expense of huge public storage ($O(z^3)$).

Atallah, Blanton and Frikken [8] proposed an improved construction that requires at most $z-1$ hash operations as key derivation cost and $z(z-1)$ public edge values. Using $2-Hop$ shortcut edge scheme [8], it requires key derivation cost as 4 hash operations with a cost of $O(z^2 \, log \, z)$ additional public shortcut edge values. Shortcut edges are used to reduce key derivation cost. A $2-Hop$ shortcut edge scheme is such that distance between any two nodes in a linear hierarchy is at most 2 edges (Hops). Using $log \, z - Hop$ shortcut edge scheme [8], it requires key derivation cost as $2 \, log \, z$ with $O(z \, log \, z)$ additional public shortcut edge values.

Crampton [9,10] exploits the fact that it is not necessary to derive keys for non-leaf nodes in a subscription hierarchy. He proposed an improved time-bound $HKAS$ with single key per user where he divides the nodes in the subscription hierarchy into triangle, rectangle and square blocks and edges are inserted between different blocks for key derivation. He was able to reduce key derivation cost up to $log \, z$ hash operations. Number of public edge values required with a subscription hierarchy is $z(z-1)$ similar as in Atallah et al. [8]. In [9], he discussed one more construction (for single key/user), with key derivation cost of 2 hash operations but with a significant increase in public edge values ($O(z^3)$, similar to [7]). Recently Crampton [11] proposed another similar (for single key per user) construction with a reduced key derivation cost of $log \, log \, z$ hash operations but with a public storage cost of $z^2(1+(1/6) \, log \, log \, z)$. The aforementioned constructions illustrate the trade-offs between key derivation cost and public storage cost.

In this work, we propose a new construction for time-bound $HKAS$ with single key storage per user per subscription. Our scheme uses indirect key derivation with dependent keys. As shown in Section 3, our scheme requires less public storage ($z(z-1) - \lceil (1/8)z(3z+2) \rceil$) and only one key private storage at CA. It requires at most $log\, z$ hash operations as key derivation cost. The security of our scheme relies on the one-way property of cryptographic hash functions.

In the next section, we describe our proposed time-bound $HKAS$. We analyze and compare performance of our proposed construction with other time-bound $HKAS$ in Section 3.

2 Proposed Scheme

We propose a time-bound $HKAS$ that uses xor and hash functions. A time-bound $HKAS$ generates system public information and, secret information for each existing subscription interval such that a user with a secret information can derive any authorize subscription node's key in the hierarchy. Time-bound $HKAS$ is said to be secure if it is sound, even if user's collude. A scheme is sound if a user will not have secret information using which he can compute any unauthorized encryption key in the system. Our scheme uses indirect key derivation with dependent keys. In indirect key derivation, a user can derive key of any immediate successor node directly and hence can compute key of any descendant node working along the path towards the target node ([12]). In case of dependent keys, key of a node is dependent on the keys of its predecessor nodes [1]. CA will generate, assign and maintain secret keys in the system.

Key Assignment. Keys are assigned using following steps.

Step 1. Let, system consists of z time slots $t_1, ..., t_z$. CA generates a subscription hierarchy structure with z leaf nodes, one for each time slot t_i, with $1 \leq i \leq z$. It contains a node for every possible subscription interval in a system of z time slots. Nodes are arranged in such a way that at each level l in the subscription hierarchy structure, there are $l+1$ nodes with subscription interval of size $z-l$ each. Therefore, at level $l = 0$ (root level) there is only one $(0+1)$ node with subscription interval size z, i.e., node with time interval (t_1, t_z) represented as $(1, z)$. An example subscription hierarchy structure (or a subscription hierarchy) for $z = 4$ time slots is shown in Figure 2(a).

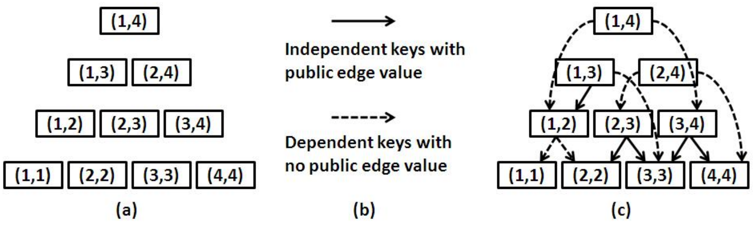
(1,4)
(1,3)
(2,4)
(1,2)
(2,3)
(3,4)
(1,1)
(2,2)
(3,3)
(4,4)
(a)
Independent keys with public edge value
Dependent keys with no public edge value
(b)
(c)

Fig. 2. (a) An example subscription hierarchy, (b) Arrow types in Fig.(c), (c) Key derivation structure corresponding to subscription hierarchy in Fig.(a)

Step 2. CA chooses a public cryptographic one-way hash function $h()$ for key generation. $h()$ is also used by the subscribers for key derivation purpose. A secret key K_s is generated and stored by the CA. CA assigns a random public label $l_{(a,b)}$ to each node (t_a, t_b) in the subscription hierarchy. In order to assign keys to the nodes in subscription hierarchy, CA will call procedure *Key_Assignment()*, defined in Algorithm 1. Step 1 in Algorithm 1, moves to each level from top to bottom in the subscription hierarchy. Step 2 initializes counter i to one, to point left most node in a level. Step 3 moves to each node from left to right in a level. If no key is assigned to the selected node, it compute and assign key to the node using $h()$ and K_s (Step $4-6$). Steps $7, 8$ computes two child nodes named $left$ and $right$ child nodes respectively of the selected node. In steps $9-13$, if key to $left$ child node is $Null$, then compute and assign dependent key $K_{(i,left)}$ as in Step 10. Else, compute public edge value $r_{(i,j),(i,left)}$ for key derivation between nodes (Step 12). Steps $14-18$ gives similar treatment to $right$ child. Step 19 will increment i by one, to move next node in same level.

Algorithm 1. $Key\_Assignment(SH, z, K_s, h())$

DESCRIPTION: Given a subscription hierarchy SH, number of time slots z, secret key K_s associated with SH and $h()$, assign keys to nodes in SH.

```
1:  for $l = 0 \to (z-2)$ do
2:      $i = 1$
3:      for $j = (z-l) \to z$ do
4:          if $(K_{(i,j)} = Null)$ then
5:              $K_{(i,j)} = h(K_s, l_{(i,j)})$
6:          end if
7:          $left = \lfloor (i+j)/2 \rfloor$
8:          $right = left + 1$
9:          if $K_{(i,left)} = Null$ then
10:             $K_{(i,left)} = h(K_{(i,j)}, l_{(i,left)})$
11:         else
12:             $r_{(i,j),(i,left)} = h(K_{(i,j)}, l_{(i,left)}) \oplus K_{(i,left)}$
13:         end if
14:         if $K_{(right,j)} = Null$ then
15:             $K_{(right,j)} = h(K_{(i,j)}, l_{(right,j)})$
16:         else
17:             $r_{(i,j),(right,j)} = h(K_{(i,j)}, l_{(right,j)}) \oplus K_{(right,j)}$
18:         end if
19:         $i = i + 1$
20:     end for
21: end for
22: return
```

Figure 2(c) shows output of *Key_Assignment()* (Algorithm 1) considering subscription hierarchy shown in Figure 2(a). In Figure 2(c), a smooth directed edge denotes a public edge value between two end nodes and dotted directed edge denotes a dependent key generation as shown in Figure 2(b). A dotted directed edge from node u to v shows that the key of node v is computed using key of node u. Table 1 shows output of *Key_Assignment()* (Algorithm 1) with respect to subscription hierarchy shown in Figure 2(c).

Table 1. Key assignment to the subscription hierarchy given in Figure 2(c)

node	Key	left child	right child	public edge values	
				left child	right child
$(1,4)$	$h(K_s, l_{(1,4)})$	$(1,2)$	$(3,4)$	$-$	$-$
$(1,3)$	$h(K_s, l_{(1,3)})$	$(1,2)$	$(3,3)$	$r_{(1,3),(1,2)}$	$-$
$(2,4)$	$h(K_s, l_{(2,4)})$	$(2,3)$	$(4,4)$	$-$	$-$
$(1,2)$	$h(K_{(1,4)}, l_{(1,2)})$	$(1,1)$	$(2,2)$	$-$	$-$
$(2,3)$	$h(K_{(2,4)}, l_{(2,3)})$	$(2,2)$	$(3,3)$	$r_{(2,3),(2,2)}$	$r_{(2,3),(3,3)}$
$(3,4)$	$h(K_{(1,4)}, l_{(3,4)})$	$(3,3)$	$(4,4)$	$r_{(3,4),(3,3)}$	$r_{(3,4),(4,4)}$
$(1,1)$	$h(K_{(1,2)}, l_{(1,1)})$	$-$	$-$	$-$	$-$
$(2,2)$	$h(K_{(1,2)}, l_{(2,2)})$	$-$	$-$	$-$	$-$
$(3,3)$	$h(K_{(1,3)}, l_{(3,3)})$	$-$	$-$	$-$	$-$
$(4,4)$	$h(K_{(2,4)}, l_{(4,4)})$	$-$	$-$	$-$	$-$

Key Derivation. A user with a subscription key can derive any authorize encryption key within its subscription using procedure $Key\_Derivation()$, defined in Algorithm 2. Suppose that there is a user with secret information $K_{(t_a,t_b)}$ corresponding to a subscription interval (t_a, t_b). To derive an encryption key $K_{(t,t)}$ with $t_a \leq t \leq t_b$, the user will do the following.

Step 1. Let $(t_a < t_b)$, user will find two nodes with subscription (t_a, t_{mid}) and (t_{mid+1}, t_b) where $t_{mid} = \lfloor (t_a + t_b)/2 \rfloor$.

(a). Let $t \in (t_a, t_{mid})$. To compute key $K_{(t_a,t_{mid})}$, if public edge value $r_{(t_a,t_b),(t_a,t_{mid})}$ exists then user will compute $K_{(t_a,t_{mid})} = h(K_{(t_a,t_b)}, l_{(t_a,t_{mid})}) \oplus r_{(t_a,t_b),(t_a,t_{mid})}$ where $h()$ and $l_{(t_a,t_{mid})}$ are public. Otherwise, user will compute $K_{(t_a,t_{mid})} = h(K_{(t_a,t_b)}, l_{(t_a,t_{mid})})$.

(b). Let $t \in (t_{mid+1}, t_b)$. To compute key $K_{(t_{mid+1},t_b)}$, if public edge value $r_{(t_a,t_b),(t_{mid+1},t_b)}$ exists then user will compute $K_{(t_{mid+1},t_b)} = h(K_{(t_a,t_b)}, l_{(t_{mid+1},t_b)}) \oplus r_{(t_a,t_b),(t_{mid+1},t_b)}$. Otherwise, he will compute $K_{(t_{mid+1},t_b)} = h(K_{(t_a,t_b)}, l_{(t_{mid+1},t_b)})$.

Step 2. User will repeat Step 1 using fresh computed key $K_{(t_x,t_y)}$ ((t_x, t_y) is either (t_a, t_{mid}) or (t_{mid+1}, t_b)) with $t_x \leq t \leq t_y$ until get target encryption key $K_{(t,t)}$.

Since, there is a path from each subscription node to its all descendant leaf nodes in the subscription hierarchy; one will surely get any authorize leaf node encryption key following above steps.

Algorithm 2. *Key_Derivation*$(K_{(i,j)}, i, j, t, Pub)$

DESCRIPTION: Given a subscription key $K_{(i,j)}$, subscription start time slot i, subscription expiry time slot j, target time slot t with $i \leq t \leq j$ and public information, it returns target node encryption key $K_{(t,t)}$.

```
1: if $(t < i)$ or $(t > j)$ or $(j < i)$ then
2:    return $Null$
3: end if
4: while $j > i$ do
5:    $m = \lfloor (i+j)/2 \rfloor$
6:    if $(t \leq m)$ then
7:       if $r_{(i,j),(i,m)}$ then
8:          $K_{(i,m)} = h(K_{(i,j)}, l_{(i,m)}) \oplus r_{(i,j),(i,m)}$
9:       else
10:         $K_{(i,m)} = h(K_{(i,j)}, l_{(i,m)})$
11:      end if
12:      $j = m$
13:   else
14:      if $r_{(i,j),(m+1,j)}$ then
15:         $K_{(m+1,j)} = h(K_{(i,j)}, l_{(m+1,j)}) \oplus r_{(i,j),(m+1,j)}$
16:      else
17:         $K_{(m+1,j)} = h(K_{(i,j)}, l_{(m+1,j)})$
18:      end if
19:      $i = m + 1$
20:   end if
21: end while
22: return $K_{(i,j)}$
```

3 Performance Analysis

Each node in the subscription hierarchy excluding leaf nodes has at most two outgoing edges and each edge has one associated public edge value. Hence, there are at most $z(z-1)$ public edge values. The nodes in lower half levels of the subscription hierarchy are having dependent keys and hence each such node has one incoming edge which does not have associated public edge value. The number of nodes (X) in lower half levels is computed below,

```
X = # nodes in full hierarchy - # nodes in upper half levels
  = (1/2)z(z+1) - (1/2)(z/2)((z/2)+1)
  = (1/8)z(3z+2)
```

Hence, out of $z(z-1)$ edges, up to $(1/8)z(3z+2)$ (nodes in lower half levels of subscription hierarchy) nodes have dependent incoming edges without public edge value. Hence, a total of $z(z-1) - (1/8)z(3z+2)$ public edge values are required.

In our scheme, each key in the subscription hierarchy is computed with single key (e.g. K_s). Hence, only one key is required with CA to derive all other keys. In [9,10], since keys are independently assigned to the nodes in the subscription hierarchy, there are many nodes whose parent does not exists (are root nodes). A key for each root node must be stored at CA. We can see in their hierarchy that at least upper half of the levels do not have parents (hence are root nodes. Therefore, $(1/2)(z/2)((z/2) + 1) = (z/8)(z + 2)$ keys must be stored at CA. In [8], since using root node key, CA can derive every key in the hierarchy, it requires only one key to store. In [7], there are two levels. Upper level will have a node corresponding to each subscription interval with more than one time slots. In other way, it includes all nodes in the considered subscription hierarchy other than leaf nodes (nodes in upper $z - 1$ levels) i.e. $(z/2)(z - 1)$.

Table 2 compares cost associated with a subscription hierarchy in the existing time-bound $HKAS$. Ateniese et al [7] scheme requires one decryption operation for key derivation with an expense of huge $(O(z^3))$ public storage. Atallah et al base scheme [8] requires at most z hash operations as a key derivation. When using $log\, z - Hop$ shortcut edge scheme, Atallah et al improved scheme reduces key derivation cost up to $2 log z$ with an expense of additional $(> z^2)$ public edge values. Crampton [9,10] was able to reduce key derivation cost up to $log z$ without using any additional (shortcut) public edge value by using independent keys. In our scheme, we use dependent keys and are able to further reduce public storage by a factor of $(1/8)z(3z + 2)$. Key derivation cost in our scheme is similar (at most $log\, z$ steps) to [9,11], as a user can jump half of the existing levels towards target leaf node in every step.

Table 2. Comparison of single key time-bound $HKAS$

Scheme	Type of keys	Public edge values	Secret storage at CA	Key derivation cost
Ateniese et al Scheme [7]	independent	$z(z-1)(z+4)/6$	$z(z-1)/2$	1 decryption
Atallah et al base Scheme [8]	independent	$z(z-1)$	1	z
Atallah et al Impv. Scheme [8] with $log\, z - Hop$ scheme	independent	$> z(z-1) + z^2$	1	$2.log\, z$
Crampton Scheme [9]	independent	$z(z-1)$	$> z/8(z+2)$	$log\, z$
Our proposed Scheme	dependent	$z(z-1) - \lceil(1/8)z(3z+2)\rceil$	1	$log\, z$

Note that, allowing more number of outgoing edges to a node in subscription hierarchy will reduce key derivation cost. There is a trade-off between outgoing edges to a node and key derivation cost. In our scheme, if we allow $log z$ outgoing edges to a node (with the same spirit as in [11]), key derivation cost will be reduced to $log\, log\, z$ as in [11]. Public storage cost in our construction will be still less with a factor of $(1/8)z(3z + 2)$ since these number of nodes are still require an incoming edge with dependent key derivation i.e. without any public edge value.

4 Future Work

Future direction to this work includes: extending system hierarchy to consider dynamic operations like add or delete user subscription in the system and formal security analysis with appropriate adversary model.

References

1. Akl, S.G., Taylor, P.D.: Cryptographic solution to a problem of access control in a hierarchy. ACM Trans. on Computer Systems 1(3), 239–248 (1983)
2. Tzeng, W.G.: A time-bound cryptographic key assignment scheme for access control in the hierarchy. IEEE Trans. on Know. and Data Eng. 14, 182–188 (2002)
3. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
4. Yi, X., Ye, Y.: Security of Tzeng time-bound key assignment scheme for access control in hierarchy. IEEE Trans. on Know. and Data Eng. 15(4), 1054–1055 (2003)
5. Zhu, W.T., Deng, R.H., Zhou, J., Bao, F.: Time-bound hierarchical key assignment: An overview. IEICE Trans. 93-D(5), 1044–1052 (2010)
6. Ateniese, G., De Santis, A., Ferrara, A.L., Masucci, B.: Provably-secure time-bound hierarchical key assignment schemes. In: ACM Conference on Computer and Communications Security, pp. 288–297 (2006)
7. Ateniese, G., De Santis, A., Ferrara, A.L., Masucci, B.: Provably-secure time-bound hierarchical key assignment schemes. Journal of Cryptology 25(2), 243–270 (2012)
8. Atallah, M.J., Blanton, M., Frikken, K.B.: Incorporating temporal capabilities in existing key mgmt. schemes. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 515–530. Springer, Heidelberg (2007)
9. Crampton, J.: Trade-offs in cryptographic implementations of temporal access control. In: Jøsang, A., Maseng, T., Knapskog, S.J. (eds.) NordSec 2009. LNCS, vol. 5838, pp. 72–87. Springer, Heidelberg (2009)
10. Crampton, J.: Time-storage trade-offs for cryptographically-enforced access control. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 245–261. Springer, Heidelberg (2011)
11. Crampton, J.: Practical and efficient cryptographic enforcement of interval-based access control policies. ACM Trans. on Inf. Syst. Secur. 14(1), 14 (2011)
12. Atallah, M.J., Frikken, K.B., Blanton, M.: Dynamic and efficient key management for access hierarchies. In: ACM Conference on Computer and Communications Security, pp. 190–202 (2005)