

Trade-Offs in Cryptographic Implementations of Temporal Access Control

Jason Crampton

Information Security Group, Royal Holloway, University of London
jason.crampton@rhul.ac.uk

Abstract. In recent years, we have seen the development of key assignment schemes that use cryptography to enforce time-based authorization policies. One of the most important aspects of these schemes is the balance between the time required to derive keys and the amount of storage required for the public information from which keys are derived. The derivation time and storage are dependent on the number of time periods used in the authorization policy. In this paper, we discuss novel schemes that achieve good trade-offs between these competing parameters and for which explicit bounds can be given in terms of the number of time periods.

1 Motivation

In some situations, we may wish to use cryptographic techniques to implement some form of access control. By (symmetrically) encrypting the contents of a data object, we restrict access to an object to those users who possess the corresponding encryption key. The problem we now need to solve is the efficient and accurate distribution of encryption keys to authorized users.

In recent years, there has been a considerable amount of interest in key encrypting schemes. In such schemes, a user is given a secret value – typically a single key – which enables the user to derive some collection of encryption keys. Key derivation is performed using the secret value and some information made publicly available by the scheme administrator.

The most widely studied situation assumes the existence of a partially ordered set (L, \leq) and an encryption key $\kappa(x)$ associated with each element $x \in L$. Each user is associated with a security label $\lambda(u) \in L$ and should be able to derive all keys in the set $\{\kappa(y) : y \leq \lambda(u)\}$. This scenario is based on the information flow policies for confidentiality that were widely studied in the 1970s [5, 12]. It is generally assumed that the trivial solution in which the user's secret is the set of keys $\{\kappa(y) : y \leq \lambda(u)\}$ is not of interest. Instead, the research literature has focused on so-called *key assignment schemes* in which each user is given a single key and additional public information is used to derive additional keys. The two objectives when designing such a scheme are to minimize the amount of public information and the time required to derive a key. Unsurprisingly, it is not possible to realize both objectives simultaneously, so trade-offs have been

sought. Crampton *et al.* provide a survey of, and taxonomy for, key assignment schemes, and the various factors that affect the parameters described above [9].

At the same time, we have seen the development of access control models in which time plays an important role in deciding whether access requests are authorized or not [6]. One particular application of such “temporal access control” systems is the protection of data that is made available periodically as (part of) a subscription-based service [7].

Crampton *et al.* suggested that temporal access control policies, in which there are number of time points and users were authorized to access data over a sequence of consecutive time points, could be implemented using key assignment schemes. Shortly after, Atallah *et al.* [2], Ateniese *et al.* [4] and De Santis *et al.* [11] described key assignment schemes for temporal access control using this representation for time. Their work focused on two particular aspects:

- the development of schemes that were secure against key recovery, and
- the reduction of the storage required for public information and the number of operations required for key derivation.

One shortcoming of their work is that the approaches used to tackle the second of these issues do not consider the actual requirements of the underlying access control model. Instead, generic techniques to reduce the diameter of a directed graph are applied. This has two consequences: application-specific optimizations are not considered and only the asymptotic behavior of the constructions is provided. Given that the number of time intervals is likely to be rather small in many practical applications, it is not clear that this kind of approach is the most appropriate.

In this paper, we consider application-specific optimizations that arise from two rather straightforward observations. This enables us to present concrete schemes with precise bounds on the amount of storage and the number of derivation steps required. One nice feature of this work is that we may still be able to exploit the techniques used in earlier work to obtain some further reduction in storage and derivation time.

In the next section, we describe some relevant background material. We then make a simple, yet powerful, observation and explore the schemes that can be derived as result of this observation. In Sect. 4, we explore the consequences of another simplification. Section 5 describes a different way of decomposing the partially ordered set of time intervals, which yields methods for fixing the number of derivation steps required. We conclude the paper with a summary of our contributions, a comparison with related work and some suggestions for future work.

2 Key Assignment Schemes

Let us suppose we have a partially ordered set of security labels (L, \leq) . An information flow policy requires that each user u and protected object o be assigned a security label; u is authorized to read o provided the security label

of u is greater than or equal to that of o . More formally, let $\lambda : U \cup O \rightarrow L$ be a labeling function that associates each entity with a security label. Then u is authorized to access o if and only if $\lambda(u) \geq \lambda(o)$.

A *key assignment scheme* may be used to enforce an information flow policy. A key assignment scheme comprises a set of keys $\{\kappa(x) : x \in L\}$ and a set of public information. Each object with security label x is encrypted with $\kappa(x)$. A user u with the key $\kappa(\lambda(u))$ must be able to derive $k(y)$ for any $y \leq \lambda(u)$ using $\kappa(\lambda(u))$ and public information. Hence, a user can decrypt any object with security label $\lambda(y)$, where $y \leq \lambda(u)$. The parameters that characterize the behavior of a key assignment scheme are [9]:

- the number of keys that a user requires;
- the amount of public information that is required;
- the amount of time taken to derive a key.

Generally speaking, we are only interested in schemes in which each user has a single key. (We could, trivially, give a user a key for each of the labels that are less than or equal to $\lambda(u)$ – that is, the set of keys $\{\kappa(y) : y \leq \lambda(u)\}$ – but this type of approach is rarely considered appropriate.) As a consequence we require either more public information or longer key derivation times (or both).

2.1 Key Assignment Schemes for Directed Graphs

A partially ordered set (L, \leq) can be represented by a graph (L, E) . There are two obvious choices for the edge set E : one is the full partial order relation \leq ; the second option is to omit all transitive and reflexive edges from \leq to obtain the *covering relation*, denoted $<$. The graph $(L, <)$ is called the *Hasse diagram* of L , and is the usual representation of L as a directed graph [10].

Note that we can easily generate a key assignment scheme for any directed acyclic graph (V, E) . Specifically, we associate a key $\kappa(x)$ with each $x \in V$, and, for each edge $(x, y) \in E$, we publish $Enc_{\kappa(x)}(\kappa(y))$, where $Enc_k(m)$ denotes the encryption of message m using key k .¹ Then any user in possession of $\kappa(x)$ can derive $\kappa(y)$ in one step, and for any z on a path from x containing m edges, $\kappa(z)$ can be (iteratively) derived in m steps.

2.2 Trade-Offs in Key Assignment Schemes

It can be seen that key assignment schemes for directed graphs can be used specifically for information flow policies. We may use the graph (L, \leq) , in which case key derivation can always be performed in one step. In contrast, key derivation may require a number of steps if we use the graph $(L, <)$. The trade-off here is that the second graph contains fewer edges and hence the number of items of public information that are required to support key derivation is smaller. The study of these kinds of trade-offs will be the focus of this paper.

¹ Atallah *et al.*, for example, use the “encryption” $\kappa(y) \oplus h(\kappa(x) \parallel y)$, where h is a hash function, \oplus is bit-wise XOR, and \parallel denotes concatenation of strings [3].

2.3 Security Considerations

A number of schemes exist in the literature that prevent a user from obtaining keys for which she is not authorized by colluding with other users. The first such scheme was developed by Akl and Taylor [1] specifically for partially ordered sets. Since then Atallah *et al.* [2] and, independently, Ateniese *et al.* [4] have shown that there exist key assignment schemes that are secure against “key recovery” and can be used with arbitrary directed graphs. We will assume henceforth that the schemes we propose, which only vary in the choice of edge set for the graph, will be implemented using a scheme that is secure against key recovery.

2.4 Key Assignment Schemes for Temporal Access Control

Let us now assume that users are authorized to access objects for specified periods of time. We assume that there exists a sequence of time “points” t_1, \dots, t_m and that there are cryptographic keys $\kappa_1, \dots, \kappa_m$ associated with each time point. Each key κ_i is used to encrypt data (using some appropriate symmetric encryption technique) released at time point i . Each time point is a collection of time instants; that is, a time interval. A sequence of consecutive time points represents a longer time interval.

We can use a key assignment scheme to enforce a temporal access control scheme. Specifically, the lattice of security labels is the set of intervals defined over the set $\{t_1, \dots, t_m\}$ ordered by subset inclusion, which we denote by I_m . That is, $I_m = \{[t_i, t_j] : 1 \leq i \leq j \leq m\}$, where $[t_i, t_j]$ denotes the set $\{t_i, t_{i+1}, \dots, t_{j-1}, t_j\}$. Clearly, we may represent $\{t_1, \dots, t_m\}$ simply as the set $\{1, \dots, m\}$, which we denote by $[m]$. Henceforth, we use this representation of, and notation for, the set of time points: a user authorized for time points i, \dots, j is assigned to label $[i, j]$; key (object) κ_i is assigned to label $[i, i]$.

Note that the number of edges in the graph $(I_m, <)$ contains $m(m-1)$ edges. Note also that the Hasse diagram of I_m can be represented as a triangular grid, with the apex of the triangle representing the interval $[1, m]$ and the long diagonal representing the intervals $[i, i]$.² This graphical interpretation of I_4 is illustrated in Fig. 3(a) on page 7. Using this graph, we can see that iterative key derivation requires precisely $m-1$ steps to get from vertex $[1, m]$ to any vertex of the form $[i, i]$. More generally, it requires $j-i$ steps to get from the vertex $[i, j]$ to a vertex $[l, l]$, where $l \in [i, j]$.

Henceforth, we will call the triangular pattern of side m an *m-triangle*, denoted by T_m . We will write D_m to denote an *m-diamond* – a diamond-shaped pattern of side m (obtained by joining two copies of T_m along the long diagonal). T_4 and D_4 are illustrated in Fig. 1. Nodes of the form $[i, i] \in T_m$ will be called *leaf nodes*.

To conclude this section, we summarize the problem we address in the remainder of the paper. We wish to construct a graph (T_m, E_m) , such that

² All edges in all graphs depicted in this paper are assumed to point down the page.



Fig. 1. T_4 and D_4

- for any non-leaf node $[i, j]$ there exists a path from $[i, j]$ to $[k, k]$ for every $k \in [i, j]$;
- $|E_m|$ is small;
- the length of the longest path in (T_m, E_m) is small.

The first criterion simply reflects that the graph must implement the desired access control policy. The second means that we wish to keep the public storage requirements small, while the final criterion means that we wish to make the worst-case key derivation time small. Note that T_m is fixed, so any scheme is defined by the choice of E_m . For brevity, we will simply refer to a scheme E_m (with T_m and the graph (T_m, E_m) left implicit).

3 An Immediate Improvement

We start by making the first of our application-specific observations. This immediately yields a number of significant improvements to the number of steps required.

Remark 1 *The only security labels for which a user needs to derive a key have the form $[i, i]$ (since no objects are associated with any interval of the form $[i, j]$, where $i < j$). Therefore, it is irrelevant whether a user with security label $[i, j]$ can derive $[i', j']$, where $[i', j'] \subseteq [i, j]$ and $i' \neq j'$.*

This means that we can seek to optimize the key assignment scheme used to derive keys by omitting some of the edges in the triangular grid I_m . Clearly, the triangular grid is “self-similar”, so it is natural to examine recursive methods for generating graphs in which the number of hops is reduced. Before we discuss such methods, we introduce a very simple 1-hop scheme.

Scheme 1 (1-hop derivation) *In our first scheme $E_m^{(1)}$, we simply insert an edge between every non-leaf node $[i, j]$ to leaf node $[k, k]$, for all $k \in [i, j]$.*

Scheme $E_4^{(1)}$ is illustrated in Fig. 2. We can see that 16 edges are required: 4 for the top node, 3 for each of the nodes in the second row, and 2 for each of the nodes in the third row.

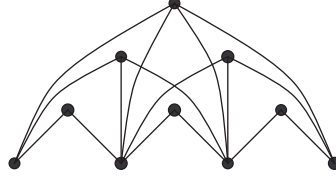


Fig. 2. The scheme $E_4^{(1)}$

Proposition 2 For all $m \geq 1$, $e_m^{(1)} = \frac{1}{6}m(m-1)(m+4)$, where $e_m^{(1)}$ denotes the cardinality of $E_m^{(1)}$.

Proof. It is easy to see that we require m edges for the uppermost node, $m-1$ edges for each of the two nodes in the next row, etc. Hence, we see that the number of edges required is given by

$$\begin{aligned}
 \sum_{i=1}^{n-1} i(n-i+1) &= (n+1) \sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2 \\
 &= \frac{1}{2}(n+1)(n-1)n - \frac{1}{6}n(n-1)(2n-1) \\
 &= \frac{1}{6}n(n-1)(3n+3-(2n-1)) \\
 &= \frac{1}{6}n(n-1)(n+4)
 \end{aligned}$$

□

Scheme 2 ($\log_2 m$ -hop derivation) To construct our second scheme $E_m^{(2)}$, we make the following observations:

- T_{2m} contains two copies of T_m and one copy of D_m ;
- no edges are required between nodes in D_m (by Remark 1);
- I_m contains two edges for each node in D_m
- for each node $[i, j] \in D_m$, $m, m+1 \in [i, j]$.

Hence, to construct $E_m^{(2)}$, we replace the two edges in I_m from each node $[i, j] \in D_m$ with two edges $([i, j], [i, m])$ and $([i, j], [m+1, j])$. In other words, each node in D_m is directly connected to a node in each copy of T_m .

Note that the number of derivation hops for T_{2m} is one more than that for T_m . Henceforth, we will write $h_m^{(i)}$ for the number of derivation hops in scheme $E_m^{(i)}$. In particular, we have $h_{2m}^{(2)} = h_m^{(2)} + 1$.

Figure 3(b) illustrates $E_4^{(2)}$. Note that it is not possible to get from the top node to a leaf node in 1 hop. Note, however, that only 12 edges are required.

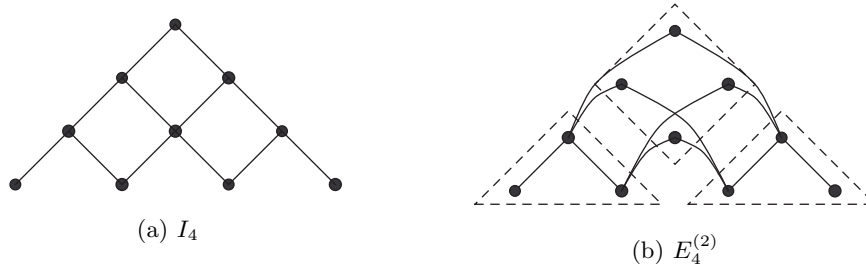


Fig. 3. Constructing the scheme $E_4^{(2)}$

In an effort to keep the notation simple, we will write h_m and e_m (rather than $h_m^{(i)}$ and $e_m^{(i)}$) where no confusion can arise about which scheme is under consideration.

Proposition 3 For scheme $E_n^{(2)}$, $n \geq 2$, we have

$$e_{2n} = 2e_n + 2n^2 \quad (1)$$

$$e_n = n(n-1) \quad (2)$$

$$h_n = \log_2 n \quad (3)$$

Proof. The first result follows immediately from the construction. Assuming we use scheme $E_n^{(2)}$, then we need two copies of T_n each containing e_n edges and we need two edges for each of the n^2 nodes in D_n .

To prove (2), recall that I_n contains $n(n-1)$ edges and note that the construction of $E_n^{(2)}$ replaces the two edges for each node in D_n with two different edges. Hence, e_n is the same as the number of edges in I_n .

Finally, note that $h_{2n} = h_n + 1$. Solving this recurrence relation, we obtain $h_n = \log_2 n$. \square

The important thing to note about $E_n^{(2)}$ is that it requires no more storage than I_n , but the upper bound on key derivation is $\log_2 n$ steps rather than $n-1$ steps. Finally, we note that our approach can also be applied to triangle T_{m+n} . We illustrate the scheme $E_5^{(2)}$ in Fig. 4.

4 Skipping a Level

Suppose we have a scheme for T_n which requires no more than h_n steps, and we have used Scheme 2 and the scheme for T_n to construct a scheme for T_{2n} with $h_n + 1$ steps. Now consider T_{4n} , which breaks down into two copies of T_{2n} and one copy of D_{2n} . Note that D_{2n} can be sub-divided into four copies of D_n . Nodes in each copy of D_n can then be attached to nodes in the appropriate copies of

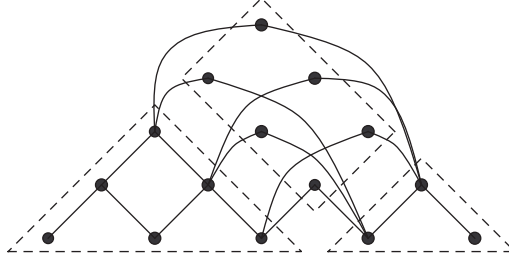


Fig. 4. Constructing $E_5^{(2)}$ using T_2 and T_3

T_n contained within each T_{2n} , skipping the intermediate nodes in T_{2n} . Hence, we do not increase the total number of hops required for derivation: a node in D_{2n} is connected directly to a node in T_n , as are nodes in each copy of T_{2n} . In other words, we have $h_{4n} = h_n + 1 = h_{2n}$. More formally, we have the following construction.

Scheme 3 Let $n = bm$. Then for T_{2n} , we split D_n into b^2 copies of D_m . We connect each node in each D_m to the appropriate nodes in copies of T_m . In particular, for $[i, j]$, where $i = \alpha_i m + \beta_i$ and $j = \alpha_j m + \beta_j$, we add edges from $[i, j]$ to nodes $[i, (\alpha_i + 1)m]$, $[(\alpha_i + 1)m + 1, (\alpha_i + 2)m], \dots, [\alpha_j m, j]$.

The (partial) construction of $E_8^{(3)}$ is illustrated in Fig. 5. Node $[2, 6]$ in D_4 has been connected to nodes $[2, 2]$, $[3, 4]$ and $[5, 6]$. Note that $[2, 6]$ would have been connected to nodes $[2, 4]$ and $[5, 6]$ in $E_8^{(2)}$. Note also that $h_8^{(3)} = h_4^{(3)} = h_2^{(3)} + 1$.

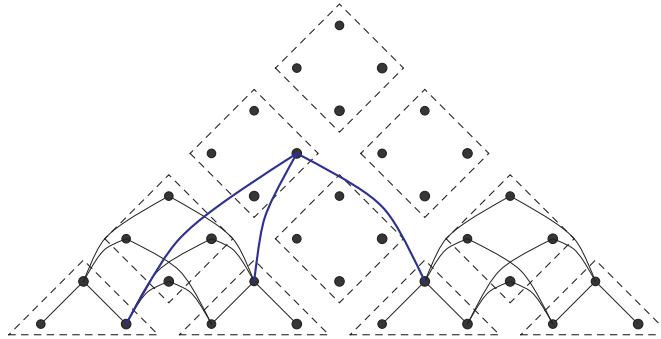


Fig. 5. The partial construction of $E_8^{(3)}$

n	Steps	Derivation of e_n	e_n
2	1		2
4	1		16
8	2	$2e_4 + (2^0 + 1).4^2$	64
16	2	$2e_8 + (2^1 + 1).8^2$	320
32	3	$2e_{16} + (2^0 + 1).16^2$	1152
64	3	$2e_{32} + (2^1 + 1).32^2$	5376
128	4	$2e_{64} + (2^0 + 1).64^2$	18944
256	4	$2e_{128} + (2^1 + 1).128^2$	87040

Table 1. Constructing a scheme with $\frac{1}{2} \log n$ derivation steps

Lemma 4 For scheme $E_n^{(3)}$, where $n = bm$, we have

$$e_{2n} = 2e_n + (b + 1)n^2.$$

Moreover, if $h_n > h_m$ then $h_{2n} = h_n$.

Proof. Omitted due to space constraints.

Notice the trivial case $b = 1$ corresponds to $e_{2n} = 2e_n + 2n^2$. Of course, in this case we add a step to the derivation cost, because we do not skip a level.

We now apply this construction to the case $b = 2$, in doing so deriving a scheme $E_m^{(4)}$ in which $h_n = \frac{1}{2} \log_2 n$. We define schemes for T_2 and T_4 , each of which use one derivation step. Then we can define a scheme for T_8 that requires two derivation steps using two copies of T_4 as the building blocks. We can define a scheme for T_{16} that also requires two derivation steps using four copies of T_4 as the building blocks. Extending further, we define a scheme for T_{32} that requires three derivation steps using two copies of T_{16} as the building blocks, and define a scheme for T_{64} that also requires three steps using four copies of T_{16} as building blocks, etc. This construction is illustrated in Table 1. Using the result of Lemma 4 and noting that $b = 1$ or $b = 2$ in this construction, we have $e_{2n} \leq 2e_n + 3n^2$.

Lemma 5 For $E_n^{(4)}$ and $n = 2^m$, we have $e_n \leq \frac{3}{2}n(n - 1)$.

Proof. We prove the result by induction on m . We choose a 1-hop scheme for T_4 with 16 edges. Hence, the result holds for $m = 2$. Now let us assume that the result holds for $m \leq M$ and consider $e_{2^{M+1}}$. Then, by construction,

$$\begin{aligned}
e_{2^{M+1}} &\leq 2e_{2^M} + \frac{3}{2}2^{2M} \\
&\leq 2 \left(\frac{3}{2}(2^{2M} - 2^M) \right) + \frac{3}{2}2^{2M} \quad (\text{by inductive hypothesis}) \\
&= \frac{3}{2}(3 \cdot 2^{2M} - 2^{M+1}) < \frac{3}{2}(2^{2M+2} - 2^{M+1}) = \frac{3}{2}2^{M+1}(2^{M+1} - 1)
\end{aligned}$$

as required. □

We conjecture that we can use $b = 2^r$, for any non-zero positive integer r , to derive a scheme with $h_n \approx \frac{1}{r+1} \log_2 n$ and $e_n \leq an(n-1)$, where a is a function of r . The sole obstacle to the direct construction of a scheme analogous to $E_n^{(4)}$ is the construction of a scheme for the base case T_{2^r} such that $e_{2^r} \leq \frac{1}{2}a2^r(2^r - 1)$. In particular, we will not be able to use a 1-hop scheme for T_{2^r} (unlike the case $b = 2$), because the number of edges in a 1-hop scheme for T_n grows with the cube of n . However, it should always be possible to find a scheme for T_{2^r} with a small number of hops so that the base case “works”. This will be the subject of future work.

We also note that we can construct a scheme that only requires $\log_2(\log_2 n)$ derivation steps. We define schemes for T_2 and T_4 , each requiring a single derivation step. Then for $T_8 = T_{2^3}$ and $T_{16} = T_{2^4}$ we build schemes using $T_4 = T_{2^2}$. For T_{2^5}, \dots, T_{2^8} , we build schemes using T_{2^4} . For $T_{2^9}, \dots, T_{2^{16}}$, we build schemes using T_{2^8} . This construction is illustrated in Table 2. A similar technique can be applied for any $n = 2^{2^k}$. In particular, we have the following result.

Lemma 6 *Let $n = 2^{2^k}$. Then we can construct a scheme for T_n such that*

$$e_n - \sqrt{n}e_{\sqrt{n}} = \frac{1}{6}n\sqrt{n}(\sqrt{n} - 1)(\sqrt{n} + 4) \quad \text{and} \quad h_n = \log_2(\log_2 n).$$

Proof. Omitted due to space constraints.

Using this formula, we have, for example,

$$e_{256} = 16e_{16} + \frac{1}{6}.256.16.15.20 = 209920,$$

which can be confirmed by Table 2. Clearly, the number of edges required is approximately $\frac{1}{6}n^2\sqrt{n}$, since the term in $n^2\sqrt{n}$ will dominate the term in $\sqrt{n}e_{\sqrt{n}}$.

5 A Multiplicative Decomposition of T_n

Let $n = ab$. Then we can split T_n into b copies of T_a and t_{b-1} copies of D_a , where $t_k = \frac{1}{2}k(k+1)$ is the k th triangle number. We treat the copies of D_a and T_a as “supernodes” in T_b . We can construct schemes for T_a and T_b and use a^2 copies of the scheme (one for each node in the non-leaf supernode) for T_b to connect the set of D_a s to the T_a s. We then use the scheme for T_a to derive keys within T_a . The construction is illustrated in Fig. 6 for T_{12} .

The simplest case arises when $b = 2$ and gives rise to Scheme 2. In this case, we have two copies of T_a and one copy of D_a . We use the obvious 1-hop scheme for T_2 . We then recursively apply this construction to T_a . This process ultimately yields a scheme which requires no more than $\log_2 n$ steps.

n	$\log_2 n$	$\lceil \log_2(\log_2 n) \rceil$	Derivation of e_n	e_n
2	1	1		2
4	2	1		16
8	3	2	$2e_4 + (2^0 + 1).4^2$	64
16	4	2	$2e_8 + (2^1 + 1).8^2$	320
32	5	3	$2e_{16} + (2^0 + 1).16^2$	1152
64	6	3	$2e_{32} + (2^1 + 1).32^2$	5376
128	7	3	$2e_{64} + (2^2 + 1).64^2$	31232
256	8	3	$2e_{128} + (2^3 + 1).128^2$	209920
512	9	4	$2e_{256} + (2^0 + 1).256^2$	550912
1024	10	4	$2e_{512} + (2^1 + 1).512^2$	1888256
2048	11	4	$2e_{1024} + (2^2 + 1).1024^2$	9019392
4096	12	4	$2e_{2048} + (2^3 + 1).2048^2$	55787520
8192	13	4	$2e_{4096} + (2^4 + 1).4096^2$	396787712
16384	14	4	$2e_{8192} + (2^5 + 1).8192^2$	3008167936
32768	15	4	$2e_{16384} + (2^6 + 1).16384^2$	23464640512
65536	16	4	$2e_{32768} + (2^7 + 1).32768^2$	185441976320

Table 2. Constructing a scheme with $\lceil \log_2(\log_2 n) \rceil$ derivation steps

5.1 A 2-Hop Scheme

Let us now consider the case where we would like to construct a scheme that requires no more than 2 steps. Then we need 1-hop schemes for T_a and T_b . Scheme 1 may be used to create such schemes requiring

$$\frac{1}{6}a(a-1)(a+4) \quad \text{and} \quad \frac{1}{6}b(b-1)(b+4)$$

edges, respectively. Hence, we can construct a scheme for T_{ab} that requires no more than 2 steps, and the number of edges is

$$a^2 \frac{1}{6}b(b-1)(b+4) + b \frac{1}{6}a(a-1)(a+4) = \frac{1}{6}ab(a(b-1)(b+4) + (a-1)(a+4)).$$

Returning to the construction of a 2-hop scheme for T_{12} as illustrated in Fig. 6, we divide T_{12} into copies of D_4 and T_4 , yielding a copy of T_3 in which the non-leaf supernodes are diamonds and leaf supernodes are triangles. We then construct a 1-hop scheme for T_3 using 6 edges (Fig. 6(b)). We duplicate this scheme for every node in the root supernode. Clearly, there are $4^2 = 16$ nodes in each supernode. Hence we require 6.16 such edges. Having done this, we can now get from any node that is contained in a copy of D_4 to a node in T_4 in one hop. It remains, therefore, to construct a scheme for each T_4 supernode such that we can get from any non-leaf node to a leaf node in one hop. Again, we can use Scheme 1 for this (as illustrated in Fig. 2). We require 16 edges for each of the three copies of T_4 . The scheme therefore requires a total of $16.6 + 3.16 = 16.9 = 144$ edges.

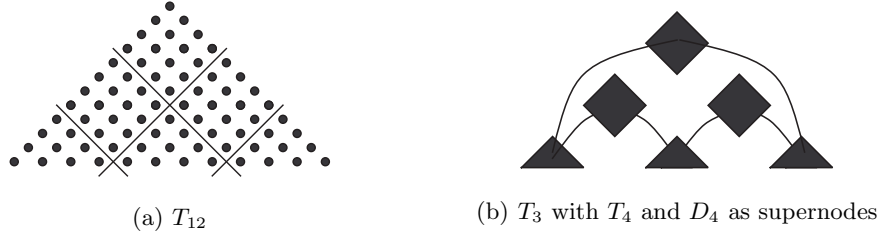


Fig. 6. Creating a scheme for T_{12} using schemes for T_3 and T_4

5.2 Minimizing the Number of Edges in the 2-Hop Scheme

The obvious question to consider is how we choose a and b so that the total number of edges e is minimized. The number of edges e is given by the formula

$$\begin{aligned}
 e &= \frac{1}{6}ab(a(b-1)(b+4) + (a-1)(a+4)) \\
 &= \frac{1}{6}ab(ab^2 + 3ab - a + a^2 - 4)
 \end{aligned}$$

Writing $b = n/a$, we obtain e as a function of n and a :

$$e = \frac{1}{6}n \left(\frac{n^2}{a} + 3n - a + a^2 - 4 \right)$$

Differentiating e with respect to a we obtain

$$\frac{1}{6}n \left(-\frac{n^2}{a^2} - 1 + 2a \right) = \frac{1}{6}n(-b^2 - 1 + 2a)$$

This expression evaluates to 0 when $2a = b^2 + 1$. Hence, the minimum number of edges occurs when $2a = b^2 + 1$. Table 3 shows how the number of edges varies for $n = 12$. (The values of $2a$ and $b^2 + 1$ are tabulated for convenience.) We see that the smallest number of edges occurs when $a = 4$. In this case, $b = 3$ and $2a \approx b^2 + 1$. Notice that $a = 1$ and $a = 12$ are boundary cases in which the whole scheme requires one hop (since any scheme for T_1 requires 0 hops). Hence the combined scheme is equivalent to a simple 1-hop scheme.

Let us now assume that $n = ab$ and $a = \frac{1}{2}(b^2 + 1)$. (In other words, we have chosen n so that the minimum number of edges will be attained.) Substituting

a	b	Edges	$2a$	$b^2 + 1$
1	12	352	2	145
2	6	212	4	37
3	4	172	6	17
4	3	160	8	10
6	2	172	12	5
12	1	352	24	2

Table 3. How the number of edges varies for different 2-hop schemes for T_{12}

$a = \frac{1}{2}(b^2 + 1)$ into e , we obtain

$$\begin{aligned}
e &= \frac{1}{24}n(3b^4 + 6b^3 + 2b^2 + 6b - 17) \\
&= \frac{1}{24}n(3b^2(b^2 + 1) + 6b(b^2 + 1) - b^2 - 17) \\
&\leq \frac{1}{24}n(6bn + 12n) \quad \text{since } n = ab = \frac{1}{2}b(b^2 + 1) \\
&= \frac{1}{4}n^2(b + 2)
\end{aligned}$$

Therefore, when $n = \frac{1}{2}b(b^2 + 1)$, there exists a 2-hop scheme for T_n with no more than $\frac{1}{4}n^2(b + 2)$ edges. When $b = 3$, for example, we have $a = 5$ and $n = 15$. In other words, the best 2-hop scheme for T_{15} arises when we use copies of T_5 and D_5 as supernodes and use a 1-hop scheme for T_3 to move between the supernodes; for this construction, we have $e = 265$. The upper bound derived above is $\frac{1}{4}15 \cdot 15 \cdot 5 = \frac{1125}{4}$ which clearly exceeds 265.

Since $n = \frac{1}{2}b(b^2 + 1)$, we have $b < \sqrt[3]{2n}$. Notice that a 1-hop scheme requires $\frac{1}{6}n(n-1)(n+4)$ edges – that is, approximately $\frac{1}{6}n^2(n+4)$ edges – while a 2-hop scheme requires approximately $\frac{1}{4}n^2(\sqrt[3]{2n} + 2)$. The 1-hop scheme for T_{15} , for example, requires 665 edges, compared to 265 for the 2-hop scheme described in the previous paragraph.

Note that \sqrt{n} , while perhaps being the natural choice for a (and b), is not the optimal choice. To see this, note that the number of edges e for such a 2-hop scheme is given by

$$\begin{aligned}
e &= \frac{1}{6}n(\sqrt{n}(\sqrt{n} - 1)(\sqrt{n} + 4) + (\sqrt{n} - 1)(\sqrt{n} + 4)) \\
&= \frac{1}{6}n(\sqrt{n} - 1)(\sqrt{n} + 4)(\sqrt{n} + 1) \\
&= \frac{1}{6}n(n - 1)(\sqrt{n} + 4)
\end{aligned}$$

However, $\sqrt{n} + 4 > \sqrt[3]{2n} + 2$. Hence, this scheme requires more edges than the optimal choice. For $n = 256$, for example, we find the best choice of a is 32 (and

$b = 8$) not $a = b = 16$. This is what our requirement $2a = b^2 + 1$ would suggest, since $2 \cdot 32 \approx 8^2 + 1$. The 2-hop scheme in which $a = 32$ requires 162304 edges, whereas the scheme in which $a = 16$ requires 217600.

5.3 Other Possibilities

Of course, we may decide that we wish to bound the number of derivation steps by some number $h \neq 2$. One obvious reason why we might do this in practice is if the storage costs of the 2-hop scheme are too high. In this case, we might split T_n into copies of T_a and D_a within T_b and define schemes that require h_a and h_b hops, where $h = h_a + h_b$. We can use any of the techniques described elsewhere in this paper to construct schemes that provide the desired trade-off between worst-case derivation time and the amount of public information required. A more systematic exploration of these constructions is beyond the scope of this paper and will be the subject of future work.

A second possibility is that we may require a 3-hop scheme. In this case, providing n has three factors, a , b and c , say, we can split T_n into bc copies of T_a , derive a 1-hop scheme for T_a , and then develop a 2-hop scheme for T_{bc} as above.

6 Concluding Remarks

In this paper we have developed a number of schemes for reducing the derivation time required for keys in cryptographic schemes for implementing temporal access control. This reduction has been achieved at the expense of the addition of a small amount of public information (although the reduction from $m - 1$ steps to $\log_2 m$ described in Scheme 2 requires no additional storage). The characteristics of a number of the schemes we have described are summarized in the bottom section of Table 4. In the remainder of this section we consider related and future work.

6.1 Related Work

There are two strands of closely related work. Both strands include schemes in which users may have up to three keys. We focus on schemes in which users have a single key, as they are directly comparable to our schemes.³ The relevant characteristics of comparable schemes in the literature are summarized in Table 4.

- Atallah *et al.* [2] propose a number of schemes that reduce the number of edges and the maximum number of hops for I_m , using techniques they had previously developed for total orders [3].

³ We noted in Sect. 2 that schemes in which users have more than one key are usually considered to be undesirable. Naturally, increasing the number of keys is sure to reduce either the public storage or the key derivation time or both.

- Ateniese *et al.* [4] and De Santis *et al.* [11] propose a number of schemes that take a quite different approach, using existing work due to Thorup [14] and to Dushnik and Miller [13] to reduce the diameter of I_m .

Scheme		Storage	Derivation
Atallah <i>et al.</i>	[2, §4]	$\mathcal{O}(n^2 \log_2 n)$	4
	[2, §4]	$\mathcal{O}(n^2)$	$\mathcal{O}(\log_2^* n)$
Ateniese <i>et al.</i>	[4, §4.2]	$\mathcal{O}(n^3)$	1
De Santis <i>et al.</i>	[11, §3.1]	$\mathcal{O}(n^2)$	$\mathcal{O}(\log_2 n \log_2^* n)$
	[11, §3.1]	$\mathcal{O}(n^2 \log_2 n)$	$\mathcal{O}(\log_2^* n)$
	[11, §3.1]	$\mathcal{O}(n^2 \log_2 n \log_2(\log_2 n))$	3
Our work	Scheme 1	$\frac{1}{6}n(n-1)(n+4)$	1
	Scheme 2	$n(n-1)$	$\log_2 n$
	Scheme 3 ($b=2$)	$\frac{3}{2}n(n-1)$	$\frac{1}{2} \log_2 n$
	§5.1	$\frac{1}{4}n^2(b+2)$	2

Table 4. A comparison of related work

The main distinguishing feature of our work is the focus on improved schemes that are directly relevant to the problem at hand, whereas prior work has simply applied existing techniques without considering the particular characteristics of the graph I_m and its application to temporal access control. As a consequence, we are able to define tight bounds on storage and derivation costs, whereas related work only describes asymptotic behavior. For large values of n , such a description may be useful, but, for smaller (and arguably more relevant) values of n , our approach is more informative. Moreover, without knowing the “magic” constants hidden by the \mathcal{O} notation, it is difficult to ascertain which scheme in the literature is the best to use for a particular value of n . Finally, we may still be able to apply some of the techniques used by De Santis *et al.* to further reduce the storage and derivation time for our schemes.

6.2 Future Work

There are two obvious areas for future work. The first is to investigate how to use the techniques described in Sect. 5 to construct h -hop schemes for a specified value of h . At the moment, we have only considered the case $h=2$. We also hope to investigate recursive schemes that use the factorization of n to produce h -hop schemes, where h is the number of prime factors in n .

The second strand of research would consider the application of the techniques to lattices of the form $L \times I_m$, where L is a security lattice in the usual information flow sense. These types of lattices – which provide a cryptographic

way of enforcing temporal information flow policies – have been considered by Crampton [8], as well as De Santis *et al.* [4, 11] and by Atallah *et al.* [2].

References

1. S.G. Akl and P.D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–248, 1983.
2. M.J. Atallah, M. Blanton, and K.B. Frikken. Incorporating temporal capabilities in existing key management schemes. In *Proceedings of the 12th European Symposium on Research in Computer Security*, pages 515–530, 2007.
3. M.J. Atallah, K.B. Frikken, and M. Blanton. Dynamic and efficient key management for access hierarchies. In *Proceedings of 12th ACM Conference on Computer and Communications Security*, pages 190–202, 2005.
4. G. Ateniese, A. De Santis, A.L. Ferrara, and B. Masucci. Provably-secure time-bound hierarchical key assignment schemes. In *Proceedings of 13th ACM Conference on Computer and Communications Security*, pages 288–297, 2006.
5. D.E. Bell and L. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report MTR-2997, Mitre Corporation, Bedford, Massachusetts, 1976.
6. E. Bertino, P.A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security*, 4(3):191–223, 2001.
7. E. Bertino, B. Carminati, and E. Ferrari. A temporal key management scheme for secure broadcasting of XML documents. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 31–40, 2002.
8. J. Crampton. Cryptographically-enforced hierarchical access control with multiple keys. *Journal of Logic and Algebraic Programming*, 2009. To appear: electronic preprint available from doi:10.1016/j.jlap.2009.04.001.
9. J. Crampton, K. Martin, and P. Wild. On key assignment for hierarchical access control. In *Proceedings of 19th Computer Security Foundations Workshop*, pages 98–111, 2006.
10. B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, United Kingdom, 1990.
11. A. De Santis, A.L. Ferrara, and B. Masucci. New constructions for provably-secure time-bound hierarchical key assignment schemes. In *Proceedings of 12th ACM Symposium on Access Control Models and Technologies*, pages 133–138, 2007.
12. D.E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.
13. B. Dushnik and E.W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63(3):600–610, 1941.
14. M. Thorup. Shortcutting planar digraphs. Technical Report 93-60, DIMACS, 1993.