Cryptographic Enforcement of Information Flow Policies without Public Information

Jason Crampton[1], Naomi Farley[1], Gregory Gutin[1],
Mark Jones[1], and Bertram Poettering[2]

[1] Royal Holloway, University of London
[2] Ruhr University Bochum

Abstract. The enforcement of access control policies using cryptographic primitives has been studied for over 30 years. When symmetric cryptographic primitives are used, each protected resource is encrypted and only authorized users are given the decryption key. Hence, users may require many keys. In most schemes in the literature, keys are derived from a single key explicitly assigned to the user and publicly available information. Recent work has challenged this design by developing schemes that do not require public information, the trade-off being that a user may require more than one key. However, these new schemes, which require a chain partition of the partially ordered set on which the access control policy is based, generally require more keys than necessary. Moreover, no algorithm is known for determining the best chain partition to use. In this paper we define the notion of a tree-based cryptographic enforcement scheme, which, like chain-based schemes, requires no public information but simultaneously has lower storage requirements. We formally establish that the strong security properties of recent chain-based schemes are preserved by tree-based schemes, and provide an efficient construction for deriving a tree-based enforcement scheme from a given policy that minimizes the number of keys required.

1 Introduction

Access control is a fundamental security service in modern computing systems. Informally, an access control system filters attempts by users to interact with protected resources, only allowing those interactions that are *authorized* by a *policy*, which is configured by the resource owner(s). Implementations of access control in software are vulnerable to compromise of the machine hosting the software. Moreover, such enforcement mechanisms do not work when protected resources are stored by an untrusted or semi-trusted third party, as is increasingly common.

In some situations, therefore, we may wish to use cryptographic techniques to enforce some form of access control. Such an approach is useful when data objects have the following characteristics: read often, by many users; written once, or rarely, by the owner of the data; and transmitted over unprotected networks. In

such circumstances, protected data (objects) are encrypted and authorized users are given the appropriate cryptographic keys. When cryptographic enforcement is used, the problem we must address 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* or *key assignment* 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 which decrypt the objects for which she is authorized. Key derivation is performed using the secret value and some information made publicly available by the scheme administrator. These schemes are particularly suitable for policies that can be represented in terms of information flow.

Ideally, such a scheme should 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. Most schemes in the literature assume that each user is supplied with a single key from which other keys are derived with the help of some information published by the scheme administrator (see [10] for a survey of such schemes). In 2010, Crampton *et al.* [9] introduced a new type of scheme in which users may receive several keys. The significant advantage of this scheme is that no public information is required. Moreover, the simplicity of the underlying structure of the scheme makes it possible to prove the scheme possesses very strong security properties [12].

An information flow policy is defined by a partially ordered set X and a function mapping users and resources to elements in X. Most key assignment schemes are derived directly from X. The innovation introduced by Crampton *et al.* was to consider a partition of X into chains (or total orders). It is particularly easy to work with chains, but the partition breaks some of the "connectivity" of the partial ordering. These breaks are "repaired" by issuing more than one key to some users. However, one question that remains open is how best to choose the chain partition of a partially ordered set: there may be many such partitions and different choices may lead to chain partition schemes with different characteristics.

In this paper, we show that it is possible to work with trees, rather than chains, without reintroducing the need for public information, resulting in much more space-efficient key assignment. We define a tree-based, cryptographic enforcement scheme and provide a rigorous construction for such schemes from a given partially ordered set. We identify a number of different parameters that may be important in the context of a tree-based enforcement scheme. In particular, we consider the total number of keys that may be required in such a scheme and prove that a tree-based enforcement scheme with a minimal number of keys can be constructed in time $O(|X|^2)$. We show that a tree-based enforcement scheme for a given X will typically require fewer keys than a chain-based scheme. Moreover, we present an efficient algorithm for computing the best choice of tree from the information flow policy, in contrast to chain-based methods (which assume that a chain partition is given).

Our approach is based on constructing a weighted directed acyclic graph from X and then constructing a minimum weight spanning out-tree from the graph. We establish a number of results about this out-tree that are likely to provide the foundation for further study of tree-based enforcement schemes.

In the next section, we introduce notation, relevant background material and related work. Then, in Sec. 3, we define a tree-based enforcement scheme, provide a method for constructing such schemes for a given information flow policy, and prove that all the resulting schemes have the property of strong key indistinguishability. In Sec. 4, we address the problem of finding a tree-based enforcement scheme that minimizes the total number of keys required to enforce a given policy, culminating in a polynomial-time algorithm for computing such a scheme. We conclude the paper with a summary of our contributions and some suggestions for future work. Those proofs that are useful in understanding our constructions are given in the body of the paper. The remainder, including the security proof for our construction (which extends an earlier proof by Freire *et al.* [12]), are in the appendix.

2 Background and Related Work

In this paper, we consider the cryptographic enforcement of access control policies. In particular, we focus on the enforcement of information flow policies using symmetric cryptographic primitives.[3]

2.1 Definitions and Notation

A *directed graph* (or *digraph*) $G = (V(G), E(G))$ is defined by a *vertex set* $V(G)$ and an *arc set* $E(G) \subseteq V(G) \times V(G)$. An arc in $E(G)$ is written in the form xy, where $x, y \in V(G)$. A *directed path* is a sequence of arcs $v_1 v_2, v_2 v_3, \ldots, v_{p-2} v_{p-1}, v_{p-1} v_p$, which we may also write as the sequence of vertices $v_1 v_2 \ldots v_p$ through which the path passes. We write $x \rightsquigarrow_G y$ if there exists a directed path from x to y in G. For all $x \in V(G)$, we define $x \rightsquigarrow_G x$.

The *in-degree* of a vertex $v \in V(G)$ is defined to be the number of arcs of the form uv in $E(G)$. Given an undirected rooted tree, we may orient each edge in such a way that the root has in-degree 0 and all other vertices have in-degree 1; the resulting (acyclic) digraph is called an *out-tree*. Thus if a directed path exists between a pair of two vertices in an out-tree then it is unique. H is a *spanning subgraph* of a graph G if $V(H) = V(G)$. A *spanning out-tree* is a spanning subgraph that is an out-tree.

A *partially ordered set* or *poset* is a pair $(X, \leqslant)$, where $\leqslant$ is a binary, reflexive, anti-symmetric, transitive relation. Given a poset $(X, \leqslant)$, we write $x < y$ if $x \leqslant y$ and $x \neq y$; and we may write $x \geqslant y$ if $y \leqslant x$. We write $x \lessdot y$ and say y *covers* x if $x < y$ and there does not exist $z \in X$ such that $x < z < y$. We say x is

---

[3] There exists a large body of work on the enforcement of attribute-based policies using asymmetric cryptographic primitives, notably attribute-based encryption [6, 13].

incomparable to y, denoted $x \parallel y$, if $x \nleqslant y$ and $y \nleqslant x$. We say $Y \subseteq X$ is an *antichain* if for all $x, y \in Y$, either $x = y$ or $x \parallel y$: Y is a *maximum* antichain if $|Y| \geqslant |Z|$ for every other antichain $Z \subseteq X$; the *width* of X is the cardinality of a maximum antichain.

Given a poset $(X, \leqslant)$, we define the graph $H = (X, E_0)$, where $xy \in E_0$ if and only if $x \gtrdot y$. H is called the *Hasse diagram* of $(X, \leqslant)$ and is a directed acyclic graph. A Hasse diagram of a simple poset is shown in Fig. 1 (on page 5). We may also define the graph $H^* = (X, E_0^*)$, where $xy \in E_0^*$ if and only if $x > y$. The graph H^* is obtained by taking the transitive closure of H.

An *information flow policy* is defined by a partially ordered set of security labels $(X, \leqslant)$, a set of users U, a set of (protected) objects O, and a security function $\lambda : U \cup O \to X$. We say $u \in U$ is *authorized* to read $o \in O$ if $\lambda(u) \geqslant \lambda(o)$ [5].

2.2   Basic Methods of Cryptographic Enforcement

A natural way to enforce an information flow policy is to define a cryptographic key $\kappa(x)$ for each $x \in X$, encrypt object o with $\kappa(\lambda(o))$ and give u (or enable u to derive) all keys $\kappa(x)$ such that $x \leqslant \lambda(u)$. More specifically, let $G = (X, E(G))$ be an acyclic directed graph such that $E_0 \subseteq E(G) \subseteq E_0^*$. Then the transitive closure of G is equal to H^* and $x \rightsquigarrow_H y$ if and only if $x \rightsquigarrow_G y$. By publishing key derivation information for each arc in $E(G)$, it is possible to derive $\kappa(y)$ from $\kappa(x)$ if $x \rightsquigarrow_G y$. Thus, the total amount of key derivation information required is proportional to $|E(G)|$, while the number of key derivations will depend on the lengths of the directed paths in G. We provide a more formal account of the functionality required of a cryptographic enforcement scheme in Sec. 2.4.

Typically, key derivation information is generated using an appropriate symmetric cryptographic algorithm [1]: for arc $xy \in E(G)$, the inputs to the cryptographic algorithm will include $\kappa(x)$ and $\kappa(y)$. We write $Enc(m, \kappa)$ to denote the encryption of message m with key κ. There are three very well known ways to implement cryptographic enforcement of information flow policies [10]:

Basic – give u the set of keys $\{\kappa(x) : x \leqslant \lambda(u)\}$;
Iterative – give u a single key $\kappa(\lambda(u))$ and publish $\{Enc(\kappa(x), \kappa(y)) : x \lessdot y\}$;
**Direct** – give u a single key $\kappa(\lambda(u))$ and publish $\{Enc(\kappa(x), \kappa(y)) : x < y\}$.

We may evaluate different implementations by considering a number of parameters. Let $k(x)$ be the number of keys required by a user associated with x. Then we write k to denote the maximum value of $k(x)$ taken over all x and K to denote $\sum_{x \in X} k(x)$. We write p to denote the number of items of public information,[4] and d to denote the number of key derivation operations a user may be required to perform to derive a key. Let n denote the cardinality of X. Then the characteristics of the three schemes described above are summarized in Table 1.

---

[4] It is assumed that the structure of the poset $(X, \leqslant)$ is known to all participants of a cryptographic enforcement scheme.

Scheme	Keys for u	K	k	p	d
Basic	$\{\kappa(x) : x \leqslant \lambda(u)\}$	$n + \|E_0^*\|$	$O(n)$	0	0
Iterative	$\{\kappa(\lambda(u))\}$	n	1	$\|E_0\|$	$O(n)$
Direct	$\{\kappa(\lambda(u))\}$	n	1	$\|E_0^*\|$	1

Table 1: How the parameters of various key assignment schemes vary

(a) Hasse diagram   (b) A chain partition
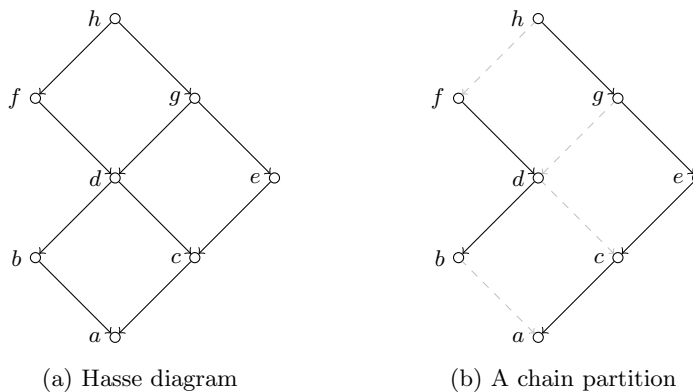
Fig. 1: The Hasse diagram of a simple poset $(X, \leqslant)$ and a chain partition

Naturally, there is a trade-off between the amount of public information we need to compute and store centrally, and the number of key derivation operations that are required. The direct scheme, for example, minimizes the cost of key derivation at the expense of an increase in public information. Consider the example in Fig. 1: the Hasse diagram of the poset has 8 vertices and 10 arcs, and the width of the poset is 2; the graph of the transitive closure has 23 arcs.

More complex schemes have been devised to reduce the number of derivation operations by increasing $|E(G)|$ [2, 8, 11]. In particular, Atallah *et al.* introduced a scheme for policies where X is a total order, in which the number of derivation operations was no greater than 2 and $|E(G)| = O(|X| \log |X|)$ [2]. Crampton extended these ideas to arbitary interval-based access control policies [8].

2.3 Chain Partition Techniques

We may consider other ways of enforcing an information flow policy. Crampton *et al.* observed that one possibility is to decompose a partially ordered set $(X, \leqslant)$ into disjoint chains and then use one-way functions to derive keys [9]. In this case, the arc set $E(G) \subseteq E_0$ and the transitive closure of G (the graph representing the chain partition) is not necessarily equal to H^* (as illustrated in Fig. 1b, in which deleted arcs are shown as gray dashed lines).

The advantage of such a scheme is that no public information is required. We simply select a key for the top element in each chain and then use a (public)

one-way function F to iteratively compute the keys for the remaining elements in each chain. In particular, if $x \lessdot y$ in a chain, then $\kappa(x) = F(\kappa(y))$.[5] Thus a user can simply derive keys by repeated applications of the one-way function. The trade-off in this case is that the user may need as many as w keys, one for each of w chains. In Fig. 1b, for example, a user assigned to vertex d will require $\kappa(d)$ and $\kappa(c)$. In short, it may be advantageous to eliminate public information, in which case each user may require multiple keys to support key derivation.

2.4 Formalization and Constructions

Recent work has formalized the security properties required of a cryptographic enforcement scheme (CES) for information flow policies [1, 3, 12]. Atallah *et al.* introduced the concepts of *key recovery* and *key indistinguishability* [1]. The former, informally, is the requirement that a coalition of users $V \subseteq U$ (the "adversary") can derive $\kappa(x)$ only if there exists $v \in V$ such that $\lambda(v) \geqslant x$. In other words, compromising users cannot lead to non-derivable keys being compromised. This is, essentially, the weakest security requirement that one might require of a CES. The schemes described in Sec. 2.2 have this property (provided the encryption scheme has reasonable properties).

However, in the interests of integrating a CES with other cryptographic tools, the stronger notion of indistinguishability was introduced. This property requires that the adversary cannot distinguish between $\kappa(x)$ and a random string (of the same length). The schemes discussed in Sec. 2.2 do not have this property (see [1], for example).

Informally, treating encryption keys as "just another encrypted data object" cannot be the basis for a robust cryptographic enforcement scheme. Specifically, the derivation of keys has to be separated from the decryption of data objects. We achieve this by introducing a secret value $\sigma(x)$ for each $x \in X$ from which $\kappa(x)$ may be derived. More formally, a CES for $(X, \leqslant)$ comprises the SetUp and Derive algorithms, the first being used to generate keys and the data used to derive keys, and the second to derive keys. Let $\mathcal{K}$ denote an arbitrary key space (typically $\mathcal{K} = \{0, 1\}^l$ for some $l \in \mathbb{N}$).

- SetUp takes as input a security parameter ρ and a poset $(X, \leqslant)$ associated with an information flow policy. It outputs, for each element $x \in X$, a pair $(\sigma(x), \kappa(x))$: $\sigma(x)$ is used to derive keys $\kappa(y) \in \mathcal{K}$, where $y \leqslant x$; and $\kappa(x)$ is used to encrypt data objects associated with security label x. The SetUp algorithm also outputs a set of public information Pub, which is used to support key derivation.[6]
- Derive takes as input $(X, \leqslant)$, Pub, start and end points $x, y \in X$ and $\sigma(x)$. It outputs $\kappa(y) \in \mathcal{K}$ if and only if $y \leqslant x$. (In particular, $\kappa(x)$ can be derived from $\sigma(x)$.)

---

[5] This method is not appropriate for arbitrary posets because we may have $y \lessdot x$ and $y \lessdot z$ [10].

[6] In some schemes, it may be the case that $\kappa(y) = \sigma(y)$ for all $y \in X$; and in some schemes, it may be that the set of public information is empty.

Atallah *et al.* described a CES in which two keys $\tau(x)$ and $\kappa(x)$ are derived from $\sigma(x)$ using a pseudorandom function and $(\tau(y), \kappa(y))$ is directly derivable from $\tau(x)$ only if $y \lessdot x$. (Thus, $\kappa(y)$ is iteratively derivable from $\sigma(x)$ if $x \rightsquigarrow y$.) The main innovation here is to separate the derivation and encryption functions of $\kappa(x)$, meaning that knowledge of the object decryption key $\kappa(x)$ does not help in deriving $\kappa(y)$. (Of course, exposure of $\tau(x)$ will allow for the derivation of $\tau(y)$ and hence $\kappa(y)$.)

Freire *et al.* introduce a security property called *strong key indistinguishability* [12], which we define formally in Fig. 4 and Definition 5 (on page 13). The adversary selects a vertex x to attack and may then learn $\{\sigma(y) : y \not\geqslant x\}$ (as in the security model for key indistinguishability) and $\{\kappa(y) : y \neq x\}$; the adversary's task is to distinguish $\kappa(x)$ from random. They then define a CES for total orders that has the property of strong key indistinguishability, in which a key $\kappa(x)$ is derived from $\sigma(x)$ using a pseudorandom function and $\sigma(y)$ is directly derivable from $\sigma(x)$ only if $y \lessdot x$. Finally, they demonstrate how this CES can be extended to arbitrary posets using the chain partition construction described in Sec. 2.3.

3 Tree-Based Enforcement Schemes

In this work, we are interested in enforcing an information flow policy, defined in terms of the Hasse diagram of a partially ordered set $(X, \leqslant)$, using cryptographic primitives. We may enforce the policy in any way we see fit. We may, for example, increase the number of arcs (by including some subset of the transitive arcs), thereby decreasing the lengths of the directed paths in the graph and the number of key derivations that are required. Thus there is a trade-off between (increasing) the number of arcs and (decreasing) the amount of storage required for public information. In particular, we could include all transitive arcs, so that all paths are of length 1 (as in the direct scheme). Alternatively, we may increase the number of keys given to each user and reduce the derivation time (keeping the number of arcs constant). This corresponds to allowing the user to start from multiple points in the graph.

In practice, there may be constraints that will dictate what kind of cryptographic enforcement schemes will be appropriate. There may be constraints, for example, on the computational power and/or storage of the end-user devices; or it may not be possible to provide an on-line server to store public information. As noted in Table 1, there are four parameters that are likely to be of interest: k, K, p, and d. We may wish to minimize or impose an upper bound on one or more of these parameters. Certain choices have been well studied, particularly those for which $k = 1$ (when each user is given exactly one key and $E_0 \subseteq E(G) \subseteq E_0^*$). Alternatively, we can eliminate public information (by ensuring that every node has at most one in-arc), at the expense of an increase in the number of keys assigned to each vertex. It is these types of schemes that we consider in the remainder of this paper. In particular, we consider the problem of minimizing K, the total number of keys required.

In the special case that the Hasse diagram $H = (X, E_0)$ is a spanning out-tree, we may use simpler cryptographic primitives to enforce an information flow policy. Specifically, we know there is a unique directed path from x to y whenever $y < x$. Hence, for all $x, y \in X$ such that $y \lessdot x$, we define $\kappa(y)$ to be $F(\kappa(x) \parallel y)$, where F is an appropriate one-way function [16] and $\parallel$ denotes string concatenation. In other words, keys are determined by the vertices, rather than the arcs, through which a directed path passes. In this case, we require no public information (apart from a description of the poset), because keys are derived only from a (secret) key and a (public) vertex label.

In general, of course, H is not an out-tree. We may assume without loss of generality, however, that our poset has a maximum element. If $(X, \leqslant)$ has more than one maximal element then we add a new element to X which is defined to be greater than all elements in X. (In this case, no user or object would be assigned to such an element.) Thus, we may assume that H^* has only one vertex of in-degree zero and so has a spanning out-tree [4, Prop. 1.7.1].

3.1 Constructing an Enforcement Scheme

In this paper, then, we investigate ways of constructing a spanning out-tree from $H^* = (G, E_0^*)$ (in order to eliminate the need for public information) by selecting an arc set that is a subset of E_0^*. However, we have to "repair" the Hasse diagram by allocating some users more than one key (because some of the paths will have been "broken" by the deletion of arcs). Thus it is interesting to consider how to select the arcs for deletion in such a way that the increase in the number of keys is minimized (either on a per-vertex basis or in total).

Figure 2 illustrates three out-trees derived from the poset in Fig. 1a. Removing arcs to create an out-tree inevitably means that certain paths are broken. The out-tree in Fig. 2a, for example, means that a user associated with vertex h only requires a single key and derivation requires no more than one hop. However, every other vertex (except a) requires additional keys in order to bridge the gaps. The above observations motivate the following definition.

Definition 1. *Given an information flow policy $(X, \leqslant)$, $E(T) \subseteq X \times X$ defines a derivation out-tree $T = (X, E(T))$ if (i) T is a spanning out-tree; (ii) $xy \in E(T)$ implies $y < x$.*

Lemma 1. *Let $D = (V, E)$ be an acyclic digraph with only one vertex r of in-degree zero. Then by selecting one in-bound arc for each vertex $x \neq r$ we obtain a spanning out-tree of D. Furthermore, any spanning out-tree of D can be constructed in this way.*

Proof. First, let us prove that T is a spanning out-tree. Clearly, T has no directed cycle and every vertex of $x \neq r$ has in-degree 1. It remains to show that T is connected and contains r. Consider a vertex $y_1 \neq r$ and a longest directed path of T terminating at y_1: $P = y_t y_{t-1} \ldots y_1$. Since T has no directed cycle all vertices of P are distinct and since P is longest, $y_t = r$. Thus, every vertex of T is reachable from r showing that T is connected and contains r.
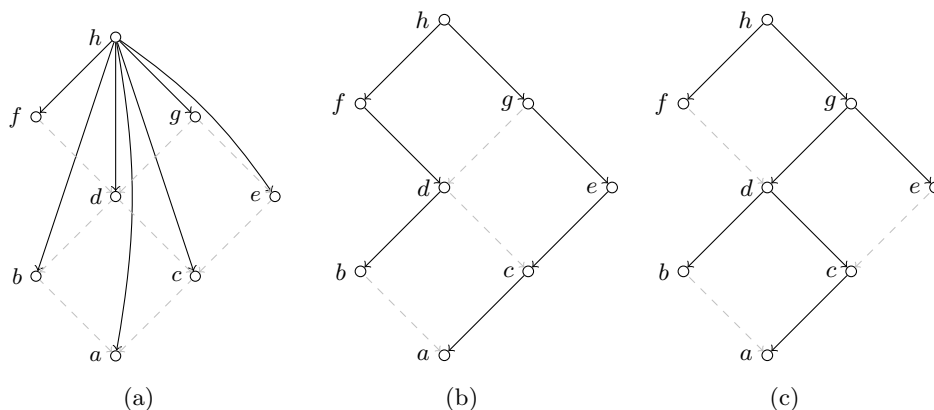
Fig. 2: Spanning out-trees derived from the poset in Fig. 1 by arc deletion

Now let T be a spanning out-tree. Note that for every vertex $x \neq r$ there is exactly one arc to x. Thus, T can be constructed by the procedure of the lemma. □

If $T = (X, E)$ is a derivation out-tree and $x \not\geqslant u$, then $x \not\leadsto_T u$. However, we may have $u < x$ but $x \not\leadsto_T u$. Thus, the problem with a derivation out-tree, in the context of cryptographic enforcement schemes, is that some authorized labels will no longer be reachable. Accordingly, we extend the notion of a derivation out-tree to a tree-based enforcement scheme.

Definition 2. *Given an information flow policy* $(X, \leqslant)$*, a* tree-based enforcement scheme *is a pair* (T, ϕ)*, where* T *is a derivation out-tree and* $\phi : X \rightarrow 2^X$ *is a* key allocation function *such that:*

- $x \in \phi(x)$*;*
- *if* $u \leqslant x$ *then there exists* $z \in \phi(x)$ *such that* $z \leadsto_T u$*;*
- *if* $u \not\leqslant x$ *then for all* $z \in \phi(x)$*,* $z \not\leadsto_T u$*.*

In a tree-based enforcement scheme (T, ϕ), directed paths in T are used to derive secrets (and hence keys): $E(T)$ determines the paths and ϕ determines the starting points of those paths (and hence the set of secrets that should be given to each user). In particular, $\phi(x) \setminus \{x\}$ is a set of vertices that were reachable from x in H^* that are no longer reachable in T. Thus, informally, $\phi(x)$ identifies a set of starting places in T from which all (and only those) nodes that were accessible in $(X, \leqslant)$ from x remain accessible in T, and $|\phi(x)| - 1$ is the number of *additional* secrets that will be required by a user with security label x.

Given a poset $(X, \leqslant)$ with maximum element r and a derivation out-tree $T = (X, E)$, define $\phi_E : X \rightarrow 2^X$, where

$$\phi_E(x) = \begin{cases} \{x\} & \text{if } x = r, \\ \{z \in X : \exists y \in X \text{ such that } yz \in E, x \geqslant z, x \not\geqslant y\} & \text{otherwise.} \end{cases}$$

We now establish that ϕ_E is the "best" tree-based enforcement scheme. First, we show that (T, ϕ_E) is indeed a tree-based enforcement scheme. We then show that for a given tree $T = (X, E)$, any tree-based enforcement scheme (T, ϕ), and any $x \in X$, $\phi(x) \supseteq \phi_E(x)$.

Lemma 2. *For any poset $(X, \leqslant)$ and any derivation out-tree $T = (X, E)$, (T, ϕ_E) is a tree-based enforcement scheme.*

Proof. We first show that $x \in \phi_E(x)$. This is trivially the case for $x = r$. If x is not the root vertex, there exists $y \in X$ such that $yx \in E$ (since T is a derivation out-tree). Moreover, $x \geqslant x$ and $x \not\geqslant y$ (since $yx \in E$ implies $x < y$). Hence, by definition, $x \in \phi_E(x)$.

Now consider the case $u < x$. Since T is a derivation out-tree, there exists a path $z_\ell z_{\ell-1} \ldots z_0$ in T, with $r = z_\ell$, $u = z_0$ and $\ell > 0$. If $z_i = x$ for some i then we are done (since $x \in \phi_E(x)$). Hence, we may assume that $z_i \neq x$ for all i. However, there exists a smallest integer $m < \ell$ such that $x \geqslant z_m$ and $x \not\geqslant z_{m+1}$. (If no such integer existed, we would have to conclude $r > x$.) By definition, $z_m \in \phi_E(x)$ and also $z_m \rightsquigarrow_T u$.

Finally, consider the case $u \not\leqslant x$ and suppose (in order to obtain a contradiction) there exists $z \in \phi_E(x)$ such that $z \rightsquigarrow_T u$. Then $u \leqslant z$ (by definition of a derivation out-tree and $\rightsquigarrow_T$) and $z \leqslant x$ (by definition of $\phi_E(x)$). By transitivity, $u \leqslant x$, the desired contradiction. $\square$

Lemma 3. *For any tree-based enforcement scheme $(T = (X, E), \phi)$ and every vertex $x \in X$, $\phi(x) \supseteq \phi_E(x)$.*

Proof. Clearly $\phi(r) \supseteq \phi_E(r)$, by definition. Given $x \neq r$, suppose (in order to obtain a contradiction) that $z \in \phi_E(x)$ and $z \notin \phi(x)$. Then, by definition of ϕ_E, there exists $y \in X$ such that $yz \in E$, $x \geqslant z$ and $x \not\geqslant y$. Now, since $z \leqslant x$ and (T, ϕ) is an enforcement scheme, there exists $t \in \phi(x)$ such that $t \rightsquigarrow_T z$. Hence $t \rightsquigarrow_T y$ (since T is a tree and $yz \in E$). Therefore, $y \leqslant t$ and $t \leqslant x$, since (T, ϕ) is an enforcement scheme and $t \rightsquigarrow_T t$. By transitivity, $x \geqslant y$ (the desired contradiction). $\square$

Thus, for a given tree T, (T, ϕ_E) is the enforcement scheme that minimizes, for each $x \in X$, the number of secrets required by a user assigned to x. Hence, for a given derivation out-tree $T = (X, E)$, it is reasonable to assume that we will always use the enforcement scheme (T, ϕ_E). Accordingly, we define

$$K(T) = \sum_{x \in X} |\phi_E(x)|.$$

That is $K(T)$ represents the total number of secrets required by a tree-based enforcement scheme based on the derivation out-tree T. Note also that $|\phi_E(x)|$ denotes the number of secrets required by a user assigned to security label x. Henceforth, given a derivation out-tree $T = (X, E)$, we will assume we will use the enforcement scheme (T, ϕ_E). Accordingly, we will write ϕ in preference to ϕ_E.

Let $T = (X, E)$ be a derivation out-tree. Then, for $y, z \in X$ such that $yz \in E$, define

$$\gamma(yz) = \{x \in X : x \geqslant z, x \not\geqslant y\}.$$

As we will see in Lemma 4 and Sec. 4, there is a strong connection between ϕ and γ, which we can use to compute a tree-based enforcement scheme efficiently.

Lemma 4. *Let $(X, \leqslant)$ be an information flow policy and let $T = (X, E)$ be a derivation out-tree. Then ϕ can be computed in time $O(|X|^2)$.*

Proof. By definition, $\phi(x) = \{z \in X : \exists y \in X \text{ such that } yz \in E, x \geqslant z, x \not\geqslant y\}$, for any x not equal to r in X. Moreover, there is a single arc in E of the form yz, for any $z \in X$, since T is a derivation out-tree. Thus, an algorithm to compute ϕ comprises an outer loop which iterates through the elements of X and an inner loop that iterates through the elements of E, where each iteration of the inner loop for arc yz tests whether $x \geqslant z$ and $x \not\geqslant y$. We can compute the adjacency matrix of H^* in time $O(|X|^2)$, which we can use to test whether $x \geqslant z$ (and $x \not\geqslant y$) in constant time. Moreover, $|E| = |X| - 1$ (since every vertex except the root has in-degree 1). Thus our algorithm runs in time $O(|X|^2)$. $\square$

3.2 Generating Keys

We now describe how to instantiate a tree-based enforcement scheme for $(X, \leqslant)$, given a derivation out-tree $T = (X, E)$, using a pseudorandom function (PRF). The scheme is a natural extension of the one used by Freire *et al.* for total orders [12].[7] Let ρ be a security parameter and $F \colon \{0,1\}^\rho \times \{0,1\}^* \to \{0,1\}^\rho$ be a PRF (as formally introduced in Section 3.3).

SetUp: The inputs to the algorithm are ρ and a derivation out-tree $T = (X, E)$ for $(X, \leqslant)$, with root vertex r.
Select secret value $s(r)$ uniformly at random from $\{0,1\}^\rho$. Set

$$\kappa(r) \stackrel{\text{def}}{=} F(s(r), r) \tag{1}$$

and, recursively, if y is a child of vertex x (in T), set

$$s(y) \stackrel{\text{def}}{=} F(s(x), y) \tag{2}$$

$$\kappa(y) \stackrel{\text{def}}{=} F(s(y), y) \tag{3}$$

Thus, for $xy \in E$, $s(y)$ is derived from $s(x)$ and the label of y, while $\kappa(y)$ is derived from $s(y)$ and the label of y.
Finally, define $\sigma(x) = \{s(y) : y \in \phi(x)\}$.

---

[7] In the special case of a total order, we obtain the scheme of Freire *et al*, modulo some differences in the choice of the second input to the PRF.
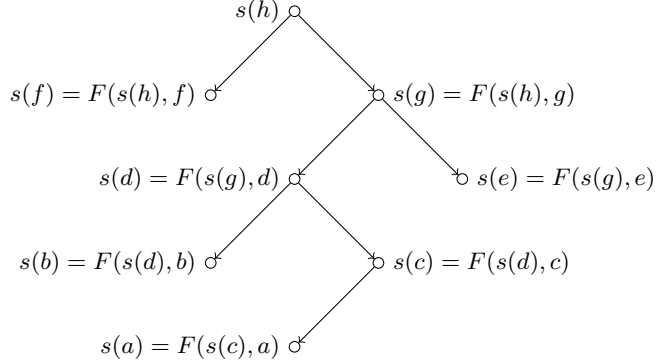
Fig. 3: The secrets generated for the spanning-out-tree in Fig. 2c

Derive: Given y, x and $\sigma(x)$, with $y \leqslant x$, there (uniquely) exists $z \in \phi(x)$ such that $z \rightsquigarrow_T y$.

If $z = y$, then (since $s(z) \in \sigma(x)$), compute $\kappa(z) = F(s(z), z)$. If $z \neq y$, then for each intermediate vertex t_i on the path $t_1 \ldots t_m$ between $t_1 = z$ and $t_m = y$, compute $s(t_i) = F(s(t_{i-1}), t_i)$. Finally, compute $\kappa(y) = F(s(y), y)$.

Our method for generating secrets is illustrated in Fig. 3.

3.3 Security Analysis

We start by specifying what we understand by a PRF. Our definition is not the most general possible and is tailored to the requirements of our construction (as described in Sec. 3.2); specifically, we assume that the keyspace and range of the PRF are the same set.

Definition 3. *A* pseudorandom function $(F_\rho)_{\rho \in \mathbb{N}}$ *is a family of efficient functions* $F_\rho \colon \mathcal{K} \times \{0,1\}^* \to \mathcal{K}$, *where we understand ρ as a security parameter and* $\mathcal{K} = \{0,1\}^\rho$ *as the keyspace.*

We will usually write $F_{\rho,K}(x)$ to denote $F_\rho(K, x)$ for any $K \in \mathcal{K}$. To further simplify the notation, we will omit ρ when no confusion can arise. We write $\mathcal{D}^O \Rightarrow 1$ to denote a configuration where $\mathcal{D}$ is a probabilistic poly-time Turing machine that has oracle access to a function O and outputs a bit with value 1.

Definition 4. *Given a pseudorandom function F, we define the* advantage *of a distinguisher $\mathcal{D}$ to be*

$$\mathrm{Adv}_{\mathcal{D}}^F(\rho) = \left| \Pr[K \leftarrow_R \mathcal{K}; \mathcal{D}^{F_K(\cdot)} \Rightarrow 1] - \Pr[\varphi \leftarrow_R \langle \{0,1\}^* \to \mathcal{K} \rangle; \mathcal{D}^{\varphi(\cdot)} \Rightarrow 1] \right|,$$

where $\langle \{0,1\}^* \to \mathcal{K} \rangle$ denotes the universe of all functions mapping $\{0,1\}^*$ to $\mathcal{K}$. We say F is indistinguishable *from a random function if the advantage of any efficient distinguisher $\mathcal{D}$ is negligible.*

$\mathrm{Expt}_{X,x,\mathcal{A}}^{\mathsf{kist},b}(1^\rho)$:

1   $(\mathsf{Pub}, \bar{\sigma}, \bar{\kappa}) \leftarrow_R \mathsf{SetUp}(1^\rho, (X, \leqslant))$
2   $K_0 \leftarrow_R \mathcal{K}$
3   $K_1 \leftarrow \kappa(x)$
4   $b' \leftarrow_R \mathcal{A}((X, \leqslant), x, \mathsf{Pub}, \mathit{Corrupt}_{X,x}, \mathit{Keys}_{X,x}, K_b)$
5   Return b'

Fig. 4: Security experiment for strong key indistinguishability

We next make precise the level of security that we target. We refer to [1, 12] for recent discussions and comparisons of security models that are specific enough to allow the analysis of CESs using the formalisms of provable security. We reproduce here the strongest model from [12]; that is, the one formalising the highest level of security, which is based on the security experiment $\mathrm{Expt}_{X,x,\mathcal{A}}^{\mathsf{kist},b}(1^\rho)$ defined in Fig. 4. We write $\bar{\sigma}$ and $\bar{\kappa}$ to denote, respectively, vectors that list the values $\sigma(x)$ and $\kappa(x)$ for all $x \in X$.

Definition 5. *Let $(X, \leqslant)$ be an arbitrary poset. A CES for $(X, \leqslant)$ is strongly key indistinguishable with respect to static adversaries if, for all $x \in X$, the advantage of all efficient adversaries $\mathcal{A}$ that interact in experiment $\mathrm{Expt}_{X,x,\mathcal{A}}^{\mathsf{kist}}$ is negligible, where we define*

$$\mathrm{Adv}_{X,x,\mathcal{A}}^{\mathsf{kist}}(\rho) = \left| \Pr\left[\mathrm{Expt}_{X,x,\mathcal{A}}^{\mathsf{kist},1}(1^\rho) \Rightarrow 1 \right] - \Pr\left[\mathrm{Expt}_{X,x,\mathcal{A}}^{\mathsf{kist},0}(1^\rho) \Rightarrow 1 \right] \right|$$

and set $\mathit{Corrupt}_{X,x} = \{\sigma(v) : v \in X, x \not\leqslant v\}$ and $\mathit{Keys}_{X,x} = \{\kappa(v) : v \in X \backslash \{x\}\}$.

Observe that in this definition, and in contrast to other models discussed in [1, 12], the adversary obtains, in principle, *all* secrets embedded in the system (that is, all $\sigma(x)$ and $\kappa(x)$ values), excluding only those that would allow distinguishing the target key by trivial means (e.g., by invoking the Derive algorithm).[8]

The final step of our analysis is to prove that our tree-based enforcement scheme from Sec. 3.2 is strongly key indistinguishable. Observe that this implies that our scheme is secure in all the models considered in [1, 12]. More formally, we have the following result.

Theorem 1. *Our tree-based enforcement scheme is strongly key indistinguishable in the sense of Definition 5. More precisely, for any poset $(X, \leqslant)$, $x \in X$, and efficient adversary $\mathcal{A}$, there exists a constant $0 \leqslant c \leqslant |X|$ and efficient distinguishers $\mathcal{D}_1^0, \ldots, \mathcal{D}_c^0, \mathcal{D}_1^1, \ldots, \mathcal{D}_c^1$ against the underlying PRF such that*

$$\mathrm{Adv}_{X,x,\mathcal{A}}^{\mathsf{kist}} \quad \leqslant \quad \mathrm{Adv}_{\mathcal{D}_1^0}^F + \cdots + \mathrm{Adv}_{\mathcal{D}_c^0}^F + \mathrm{Adv}_{\mathcal{D}_1^1}^F + \cdots + \mathrm{Adv}_{\mathcal{D}_c^1}^F \ .$$

---

[8] A variant of Definition 5 would consider dynamic adversaries: such an adversary is able to choose the challenge label x *during* the experiment, rather than having it fixed as one of the experiment's parameters. However, it has been shown that static and dynamic definitions of key indistinguishability are polynomially equivalent [12]. To simplify the exposition, therefore, we restrict our attention to the static case.

4 Minimizing K in a Tree-based Enforcement Scheme

So far, we have shown that it is possible to construct a tree-based enforcement scheme for an information flow policy $(X, \leqslant)$ that is strongly key indistinguishable. As we observed before, we will usually require our tree-based enforcement scheme to have some particular properties, such as minimizing the total number of keys or ensuring that all derivation paths are no longer than some threshold value. Hence, we require an algorithm to compute a derivation out-tree that satisfies the desired requirements, since, by Lemma 4, we can then compute the associated key allocation function ϕ in polynomial time.

In this section, we consider two questions: how to minimize K, the total number of keys allocated to vertices (by the key allocation function ϕ); and how to minimize $\widehat{K}$, the total number of keys distributed to users. The second question is interesting because, in practice, we might want to reduce the exposure of keys by ensuring that very few keys are associated with vertices to which many users are assigned. We solve both questions, demonstrating that it is surprisingly efficient to compute the required tree-based enforcement schemes in polynomial time. This is possible because of the connection between ϕ and γ, which leads to Theorem 2. We then state and prove Theorem 3, the main result of this section.

Our basic approach is to define a weight for each arc in E_0^* and construct a minimum weight spanning out-tree. Accordingly, given an information flow policy $((X, \leqslant), \lambda, U, O)$, where $\lambda : U \cup O \to X$, let $U(x) = \{u \in U : \lambda(u) = x\}$, and let $H = (X, E_0)$ be the Hasse diagram of X. Then we define the *weight function* $\omega : E_0^* \to \mathbb{N}$, where

$$\omega(yz) \stackrel{\text{def}}{=} \sum_{x \in \gamma(yz)} |U(x)|.$$

Theorem 2. *Let* $(T = (X, E), \phi)$ *be any tree-based enforcement scheme for* $(X, \leqslant)$. *Then*

$$\sum_{\substack{x \in X \\ x \neq r}} |U(x)| \cdot |\phi(x)| = \sum_{e \in E} \omega(e).$$

Proof. By definition, we have, for every $x \neq r$,

$$|\phi(x)| = |\{yz \in E : x \in \gamma(yz)\}|$$

and so

$$|U(x)| \cdot |\phi(x)| = |U(x)| \cdot |\{yz \in E : x \in \gamma(yz)\}|.$$

Hence

$$\sum_{\substack{x \in X \\ x \neq r}} |U(x)| \cdot |\phi(x)| = \sum_{\substack{x \in X \\ x \neq r}} |U(x)| \cdot |yz \in E : x \in \gamma(yz)|$$

and, since $r \not\in \gamma(yz)$ for any $yz \in E$, we have

$$\sum_{\substack{x \in X \\ x \neq r}} |U(x)| \cdot |\phi(x)| = \sum_{yz \in E} \sum_{x \in \gamma(yz)} |U(x)| = \sum_{yz \in E} \omega(yz).$$

$\square$

Theorem 3. *Given an information flow policy $((X, \leqslant), U, O, \lambda)$, we can compute a tree-based enforcement scheme (T, ϕ) such that $\widehat{K}$ is minimized in time $O(|E_0^*| + |X|^2)$.*

Proof. For brevity, we write E for $E(T)$. By Theorem 2,

$$\widehat{K} = |U(r)| + \sum_{e \in E} \omega(e).$$

An algorithm to compute the weight function ω iterates through the arcs in E_0^* and, for a given arc yz, iterates through all x in X testing whether $x \geqslant z$ and $x \not\geqslant y$. In other words, we swap the inner and outer loops in the algorithm used in the proof of Lemma 4. Thus, we can compute ω in time $O(|X|^2)$.

Since $|U(r)|$ is fixed, we minimize $\widehat{K}$ by computing a derivation out-tree that minimizes $\sum_{e \in E} \omega(e)$. By Lemma 1, we can achieve this by selecting, for each non-root vertex $x \in X$, the minimum weight arc to x, where the weights are given by ω. We need only consider each arc (in E_0^*) once, which takes time $O(|E_0^*|)$. The resulting set of arcs forms a spanning out-tree of minimum weight and the number of additional keys required is $\sum_{e \in E} \omega(e)$. We can derive the associated key allocation function in time $O(|X|^2)$, by Lemma 4; the result follows. $\square$

Corollary 1. *Given an information flow policy $((X, \leqslant), U, O, \lambda)$, we can compute a tree-based enforcement scheme such that K is minimized in time $O(|E_0^*| + |X|^2)$.*

Corollary 2. *We can find, in time $O(|E_0^*| + |X|^{3/2} |E_0^*|^{1/2})$, a minimum weight spanning out-tree that has the minimum number of leaves among such trees.*

It is useful to find a minimum weight spanning out-tree with a minimum number of leaves because the number of leaves will impose an upper bound on $|\phi(x)|$. Note, however, that $|\phi(x)|$ may be greater than the width of X (and it is not difficult to construct such an example). This is because the set of arcs in the graph that is input to MinLeaf – the algorithm used to construct the spanning out-tree – will, in general, be a strict subset of E_0^*. Thus, the size of the maximal independent set in the graph that is input to MinLeaf can exceed the width of the poset (which is the equal to the size of the maximal independent set in $G = (X, E_0^*)$).

We now prove some further properties of γ. This enables us to reduce the running time of our algorithm because we show it is sufficient to consider only arcs in E_0 (rather than E_0^*) when constructing the minimum weight spanning out-tree.

Lemma 5. *Let $(X, \leqslant)$ be a partially ordered set. Then for all $x, y, z \in X$ such that $z < y < x$,*

$$\gamma(xy) \cap \gamma(yz) = \emptyset \quad and \quad \gamma(xz) \supseteq \gamma(yz) \cup \gamma(xy)$$

Corollary 3. *Let $(X, \leqslant)$ be a partially ordered set with Hasse diagram $H = (X, E_0)$. Then, for any path $x_1 x_2 \ldots x_p$ in H^*, $p > 2$, we have*

$$\omega(x_1 x_p) \geqslant \sum_{i=1}^{p-1} \omega(x_i x_{i+1}).$$

Corollary 4. *Let $(X, \leqslant)$ be a partially ordered set with Hasse diagram $H = (X, E_0)$. Then there exists a minimum weight spanning out-tree $T = (X, E)$ with $E \subseteq E_0$.*

Corollary 5. *We can compute a tree-based enforcement scheme for information flow policy $(X, \leqslant)$ in time $O(|E_0| + |X|^2)$.*

Remark 1. In practice, we expect that $|U(x)| > 0$, although our proofs do not make this assumption. If we do make this assumption, it is possible to strengthen the statement in Corollary 4 and assert that a minimum weight spanning out-tree can *only* contain arcs from the Hasse diagram.

Fig. 5 illustrates the construction of the minimum weight spanning out-tree for the poset in Fig. 1 (assuming there is a single user for each vertex). The weight on arc ec is 3, for example, because $\gamma(ec) = \{c, d, f\}$. (The effect of retaining arc ec would be that $\kappa(c)$ would be required for each of c, d and f. Equivalently, $c \in \phi(d)$ and $c \in \phi(f)$ if we were to choose ec to belong to our derivation out-tree.) To construct a minimum weight spanning out-tree, we must select arcs ca and dc (and we select one or other of fd and gd). One possible scheme, when gd is retained rather than fd is illustrated in Fig. 5b; the scheme requires a total of 11 keys, being the sum of the weights on the retained arcs plus an extra one for the root vertex.

Remark 2. Our construction will almost always require fewer keys than a scheme based on chain partitions. This follows by noting that any vertex x, such that $x > y$, $x > z$ and $\{y, z\}$ is an antichain, necessarily requires (at least) two keys in a chain partition scheme, but this is not necessarily true of our construction (since the derivation tree may include many antichains). Consider the chain partition in Fig. 1b and the derivation tree in Fig. 5b. The former would require 13 keys, while the latter only 11.

5 Conclusion

In this paper, we have introduced a new form of cryptographic scheme for the enforcement of information flow policies. Our scheme has the advantage that no

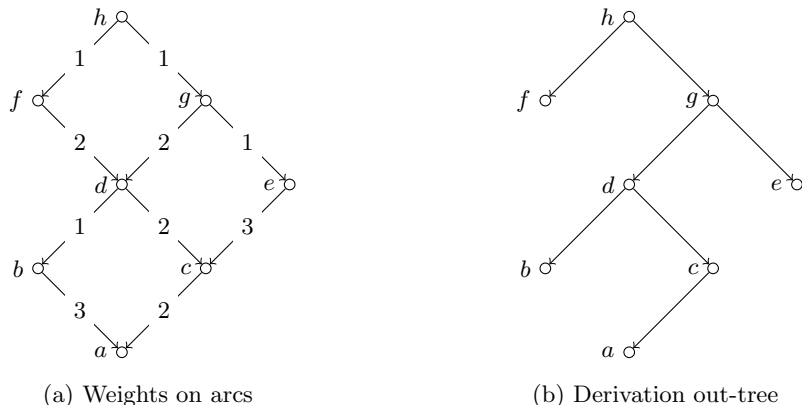(a) Weights on arcs　　　　　　　(b) Derivation out-tree

Fig. 5: The minimum weight derivation tree for Fig. 1

public information is required for the derivation of decryption keys. Moreover, our tree-based scheme requires fewer keys (when X is not a total order), compared to existing chain-based approaches, to enforce a given policy. Nevertheless, our scheme retains the strong security properties that have recently been established for chain-based schemes [12]. From a practical perspective, we provide an efficient algorithm for computing an optimal derivation tree, in the sense that it requires the smallest number of keys. This is in sharp contrast to chain-based approaches, which provide no guidance on how best to select a chain partition of the poset (of which there may be many) nor provide a way of computing the number of keys required for a given partition. Thus, there are particular practical advantages to using a tree-based approach.

There are several interesting opportunities for future work. From a mathematical perspective, it would be interesting to establish the minimum total number of keys required by a chain-based scheme and, if possible, to quantify the benefits offered by a tree-based scheme. This is, however, likely to be non-trivial, as it is not clear that there exists a weight function for chain-based schemes that can be used to formulate a result analogous to Theorem 2. From a more practical perspective, it would be interesting to find an algorithm that can compute a derivation tree such that (i) no user requires more than w keys, where w is the width of the poset (ii) the total number of keys is as small as possible. In particular, such a construction may be useful in scenarios where the user devices have limited secure storage for keys. Our preliminary work on this problem suggests that no efficient algorithm exists, but whether it is an NP-hard problem remains open. We also intend to investigate whether a forest-based enforcement scheme, which would share some of the characteristics of tree- and chain-based schemes, would offer advantages in terms of reducing (i) the maximum number of steps required for key derivation (ii) the administrative effort required following key revocation (since we can limit key updates to those vertices within a tree in the forest). In Fig. 5b, for example, we could delete arc gd to yield a forest of two

trees: each user assigned to vertex h or g would require an additional key $(\kappa(d))$ but worst-case key derivation would require two, rather than four, hops.

References

1. M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3), 2009.
2. M. J. Atallah, M. Blanton, and K. B. Frikken. Incorporating temporal capabilities in existing key management schemes. In J. Biskup and J. Lopez, editors, *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 515–530. Springer, 2007.
3. G. Ateniese, A. De Santis, A. L. Ferrara, and B. Masucci. Provably-secure time-bound hierarchical key assignment schemes. In Juels et al. [15], pages 288–297.
4. J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications.* Springer, 2nd edition, 2009.
5. D. Bell and L. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report MTR-2997, Mitre Corporation, Bedford, Massachusetts, 1976.
6. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.
7. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 3rd edition, 2009.
8. J. Crampton. Practical and efficient cryptographic enforcement of interval-based access control policies. *ACM Trans. Inf. Syst. Secur.*, 14(1):14, 2011.
9. J. Crampton, R. Daud, and K. M. Martin. Constructing key assignment schemes from chain partitions. In S. Foresti and S. Jajodia, editors, *DBSec*, volume 6166 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2010.
10. J. Crampton, K. M. Martin, and P. R. Wild. On key assignment for hierarchical access control. In *CSFW*, pages 98–111. IEEE Computer Society, 2006.
11. A. De Santis, A. L. Ferrara, and B. Masucci. New constructions for provably-secure time-bound hierarchical key assignment schemes. *Theor. Comput. Sci.*, 407(1-3):213–230, 2008.
12. E. S. V. Freire, K. G. Paterson, and B. Poettering. Simple, efficient and strongly KI-secure hierarchical key assignment schemes. In E. Dawson, editor, *CT-RSA*, volume 7779 of *Lecture Notes in Computer Science*, pages 101–114. Springer, 2013.
13. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Juels et al. [15], pages 89–98.
14. G. Gutin, I. Razgon, and E. J. Kim. Minimum leaf out-branching and related problems. *Theor. Comput. Sci.*, 410(45):4571–4579, 2009.
15. A. Juels, R. N. Wright, and S. D. C. di Vimercati, editors. *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006.* ACM, 2006.
16. R. S. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Inf. Process. Lett.*, 27(2):95–98, 1988.

A Proofs

Proof (of Theorem 1). The argument proceeds using sequences of $|X| = n$ hybrid games that interpolate between experiments $\text{Expt}^{\text{kist},0}$ and $\text{Expt}^{\text{kist},1}$. In each hybrid step, if specific conditions are met, we replace one PRF instance by a random function; from the point of view of the adversary, the distance between each two consecutive hybrids is not greater than Adv^F, for an appropriate PRF distinguisher.

Fix a poset $(X, \leqslant)$, a derivation out-tree $T = (X, E(T))$ for X, a label $x \in X$, and an efficient adversary $\mathcal{A}$. Let $x_n \prec x_{n-1} \prec \cdots \prec x_2 \prec x_1 = r$ be any (reverse) linear extension of X; that is x_n is a smallest element in X and x_1 is the root.[9] For $b \in \{0, 1\}$, we set $G_0^b = \text{Expt}_{X,x,\mathcal{A}}^{\text{kist},b}$ and define games $G_1^b, \ldots, G_n^b$ such that, if $x \not\leqslant x_k$ then G_k^b and G_{k-1}^b are identical, and if $x \leqslant x_k$ then the difference between games G_k^b and G_{k-1}^b is precisely that all PRF invocations with key $\sigma(x_k)$ are replaced by assignments with values in $\mathcal{K}$ drawn uniformly at random. Let S_k^b denote $\Pr[G_k^b \Rightarrow 1]$ for all b, k.

Observe that we replace PRF invocations by random assignments for precisely those labels x that do not have a corresponding entry in $Corrupt_{X,x}$. Observe also that, as we consider labels $x \in X$ in a suitable order, for all switchings from a PRF to a random function we have that the corresponding PRF key $\sigma(x)$ was replaced with a uniform random value before. Hence, by a standard reductionist argument, in the cases $x \leqslant x_k$ we have

$$|S_k^b - S_{k-1}^b| = |\Pr[G_k^b \Rightarrow 1] - \Pr[G_{k-1}^b \Rightarrow 1]| \leqslant \text{Adv}_{\mathcal{D}}^F \ , \tag{4}$$

for a specific (efficient) distinguisher $\mathcal{D}$; in addition, whenever $x \not\leqslant x_k$ we have $G_k^b = G_{k-1}^b$ and hence $|S_k^b - S_{k-1}^b| = 0$. Now, by repeated application of the triangle inequality and (4), we have

$$\left| S_0^b - S_n^b \right| \leqslant \sum_{i=1}^{n} \left| S_{i-1}^b - S_i^b \right| \leqslant \sum_{i=1}^{c} \text{Adv}_{\mathcal{D}_i^b}^F \ ,$$

where $c = |\{x' \in X : x \leqslant x'\}|$ and distinguishers $\mathcal{D}_i^b$ are constructed as specified. We now consider games G_n^0 and G_n^1. In both cases $\kappa(x)$ is picked uniformly at random, thus lines 2 and 3 in the experiment implement the same operation. Hence G_n^0 is identical to G_n^1 and $\left| S_n^0 - S_n^1 \right| = 0$. Thus, we obtain

$$\text{Adv}_{X,x,\mathcal{A}}^{\text{kist}} = |S_0^1 - S_0^0| \leqslant |S_0^1 - S_n^1| + |S_n^1 - S_n^0| + |S_n^0 - S_0^0|$$
$$\leqslant \text{Adv}_{\mathcal{D}_1^1}^F + \ldots + \text{Adv}_{\mathcal{D}_c^1}^F + 0 + \text{Adv}_{\mathcal{D}_1^0}^F + \ldots + \text{Adv}_{\mathcal{D}_c^0}^F$$

as required. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

---

[9] That is, if $x \leqslant y$ (in X) then $x \preccurlyeq y$ (in the linear extension). Every (finite) partial order has at least one linear extension, which may be computed, in linear time, by representing the partial order as a directed acyclic graph and using a topological sort [7, §22.3].

Proof (of Corollary 1). We simply set $|U(x)| = 1$ and apply Theorems 2 and 3. $\qquad\square$

Proof (of Corollary 2). Replace H^* by its subgraph $D = (X, E)$ obtained as follows: for each vertex $x \neq r$ delete all arcs to x apart from those of minimum weight (among arcs to x). Observe that D can be constructed in time $O(|E_0^*|)$. Find an out-tree with minimum number of leaves using algorithm MINLEAF [14]. It remains to observe that MINLEAF's runtime is $O(|E| + |X|^{3/2} |E|^{1/2})$. $\qquad\square$

Proof (of Lemma 5). Suppose $t \in \gamma(xy) \cap \gamma(yz)$. Since $t \in \gamma(yz)$, we have $t \geqslant z$ and $t \not\geqslant y$; since $t \in \gamma(xy)$, we have $t \geqslant y$, immediately leading to the desired contradiction.

Now suppose $t \in \gamma(xy)$. Then $t \geqslant y$ and $t \not\geqslant x$. Hence, we have $t > z$, by transitivity; thus $t \in \gamma(xz)$ and $\gamma(xy) \subseteq \gamma(xz)$. Finally, suppose $t \in \gamma(yz)$. Then $t \geqslant z$ and $t \not\geqslant y$. Now $t \not\geqslant x$ (otherwise, we would have $t > y$ by transitivity) and hence $t \in \gamma(xz)$; thus $\gamma(yz) \subseteq \gamma(xz)$. $\qquad\square$

Proof (of Corollary 3). Consider the case $p = 3$, with $x > y > z$. Using Lemma 5 and the fact that $|U(t)| \geqslant 0$ for all t, we have

$$\omega(xz) = \sum_{t \in \gamma(xz)} |U(t)|$$
$$\geqslant \sum_{t \in \gamma(xy)} |U(t)| + \sum_{t \in \gamma(yz)} |U(t)|$$
$$= \omega(xy) + \omega(yz).$$

Now suppose the result holds for all $p < P$ and consider a path $x_1 \ldots x_P$ containing P vertices. Then $x_1 x_{P-1} \in E_0^*$ and, by Lemma 5 and the inductive hypothesis, respectively, we have

$$\omega(x_1 x_P) \geqslant \omega(x_1 x_{P-1}) + \omega(x_{P-1} x_P)$$
$$\geqslant \omega(x_1 x_2) + \cdots + \omega(x_{P-2} x_{P-1}) + \omega(x_{P-1} x_P)$$
$$= \sum_{i=1}^{P-1} \omega(x_i x_{i+1})$$

Thus the result holds by induction. $\qquad\square$

Proof (of Corollary 4). Let $T' = (X, E')$ be a minimum weight spanning out-tree for $(X, \leqslant)$, and suppose arc xy is in E' but not in E_0. Then $x \rightsquigarrow_H y$ and let zy be the last arc in this path. Since $\omega(uv) \geqslant 0$ for each arc uv and by Corollary 3, $\omega(zy) \leqslant \omega(xy)$. Therefore by removing xy from E' and adding zy, we have a spanning out-tree with weight at most that of T'. By replacing every arc in $E' \setminus E_0$ in this way, we have a spanning out-tree $T = (X, E)$ of weight at most that of T', and therefore of minimum weight. $\qquad\square$

Proof (of Corollary 5). By Corollary 4, we may restrict our attention to arcs in the Hasse diagram. Thus we can compute the minimum weight derivation tree in time $O(|E_0|)$ and we can compute ϕ in time $O(|E_0| + |X|^2)$. $\qquad\square$