

Cryptographically-Enforced Hierarchical Access Control with Multiple Keys[☆]

Jason Crampton

Information Security Group, Royal Holloway, University of London, UK

Abstract

Hierarchical access control policies, in which users and objects are associated with nodes in a hierarchy, can be enforced using cryptographic mechanisms. Protected data is encrypted and authorized users are given the appropriate keys. Lazy re-encryption techniques and temporal hierarchical access control policies require that multiple keys may be associated with a node in the hierarchy. In this paper, we introduce the notion of a multi-key assignment scheme to address this requirement. We define bounded, unbounded, synchronous, and asynchronous schemes. We demonstrate that bounded, synchronous schemes provide an alternative to temporal key assignment schemes in the literature, and that unbounded asynchronous schemes provide the desired support for lazy re-encryption.

Key words: Information flow policy, cryptographic access control, key assignment scheme, multi-key assignment scheme

1. Introduction

One of the fundamental security services in computer systems is *access control*, a mechanism for constraining the interaction between (authenticated) users and protected resources. Generally, access control is implemented by an authorization service, which includes an authorization decision function for deciding whether a user request to access a resource should be permitted or not. The output of an authorization decision function is usually determined by evaluating the access request with respect to an authorization policy.

It is sometimes appropriate, however, to enforce an authorization policy using cryptographic techniques. In such circumstances, protected data (objects) are encrypted and authorized users are given the appropriate cryptographic keys. Such an approach is useful when objects are: read often, by many users; written once, or rarely, by the owner of the data; and transmitted over unprotected networks. Fu *et al* [10] cite content distribution networks, such as Akami and BitTorrent, as examples where some kind of cryptographic access control is particularly suitable.

A no-read-up information flow policy [9] controls access to resources by comparing security labels assigned to users and objects; a user may have read access to an object (causing an information flow from object to user) if and only if the user's security label is at least as high as

[☆]A shorter version of this paper appeared in the Proceedings of the 12th Nordic Workshop on Secure IT Systems (NordSec 2007). Programme committee co-chair Úlfar Erlingsson acted as guest editor for this revised version.

Email address: `jason.crampton@rhul.ac.uk` (Jason Crampton)

Preprint submitted to Journal of Logic and Algebraic Programming

April 1, 2009

that of the object. Such a policy can be implemented by associating a symmetric encryption key with each security label and encrypting objects with the appropriate key. A user is given (or can derive) the encryption key for each label less than or equal to his security label. Since the seminal paper of Akl and Taylor [1], there has been a considerable amount of research into so-called *key assignment schemes*, which seek to minimize the number of keys required by each user (see the recent survey paper [7] for an overview of such schemes).

Motivation and related work. One of the main practical problems with cryptographic access control in general, and with key assignment schemes in particular, is that huge amounts of data may need to be re-encrypted when a key is changed. There are a number of reasons why keys might change, of which the most important are user revocation, key compromise and access control policy requirements.

One way of alleviating the burden of object re-encryption following a user revocation is to re-encrypt objects with the new key only when the content changes. This type of approach, sometimes called *lazy re-encryption*, has been used in cryptographic file systems [4, 13]. A consequence of this is that a user may require two or more keys for objects in the same protection domain: one or more keys for objects that have not yet been re-encrypted and one for those that have. The burden of object re-encryption is particularly onerous in key assignment schemes: if a user with security label l is revoked, then the key for every security label $l' \leq l$ has to be changed and every object with security label l' has to be re-encrypted. Therefore, lazy re-encryption is a particularly appropriate technique for key assignment schemes and one that has not previously been studied.

Some key assignment schemes incorporate a temporal element (see [3] for a survey of such schemes): Bertino *et al.*, for example, have used cryptographic access control to restrict access to subscription-based content, with each time period using a different key to encrypt content [6]. In this case, a user may require two or more keys for objects in the same protection domain: one key for objects encrypted in the current period and one for each of the preceding periods. All such research has assumed that all keys (including those for future time intervals) are issued when the scheme is instantiated. Although this simplifies key distribution, there is, however, a very good reason for not doing so: if a user pays subscriptions periodically and defaults on the payments, it will be necessary to update all the keys in the scheme in order to revoke the defaulter's authorization.

Contributions. In this paper, we examine extensions to existing key assignment schemes that permit several keys to be associated with each security label and illustrate how such schemes can be used to support lazy re-encryption and temporal access control policies. The main contributions of the paper are: to extend lazy revocation schemes to hierarchical access control policies (in contrast to the Unix-like "owner-group-world" policies of previous work), and to permit the timed release of new encryption keys for temporal hierarchical access control policies. The schemes presented in this paper are derived from techniques used for existing key assignment schemes, in which each label is associated with a single encryption key. They also make use of some of the ideas of Fu *et al* [10] and work on one-time passwords [11].

Outline. In the next section, we summarize the characteristics of information flow policies and existing key assignment schemes. We then describe our new schemes, how they support multiple keys for each security label, and discuss the potential applications of each type of scheme. We conclude the paper with a summary of our contributions and some ideas for future work.

2. Preliminaries

A *partially ordered set* (or *poset*) is a pair (L, \leq) , where \leq is a reflexive, anti-symmetric, transitive binary relation on L . We say x *covers* y , denoted $y \triangleleft x$, if $y < x$ and there does not exist $z \in L$ such that $y < z < x$. L is a *total order* if for all $x, y \in L$, either $x \leq y$ or $y \leq x$.

The *Hasse diagram* of a poset is the directed graph (L, \triangleleft) [8]. A simple Hasse diagram is shown in Figure 1(a) (on page 5). Note that $d \leq a$, but this “transitive edge” is not included in the graph. Note also that all edges are assumed to be directed upwards.

An *order ideal* $I \subseteq L$ has the property that if $x \in I$ and $y \leq x$ then $y \in I$. Similarly, an *order filter* $F \subseteq L$ has the property that if $x \in F$ and $y \geq x$ then $y \in F$. The order ideal generated by $x \in L$, denoted $\downarrow x$, is defined to be $\{y \in L : y \leq x\}$. It is easy to show that the set complement of an order ideal in L is an order filter (and vice versa) [8].

Henceforth we adopt the following conventions: given a poset L and $x, y \in L$, we may write $y < x$ if $y \leq x$ and $y \neq x$, and we may write $x \geq y$ if $y \leq x$; x , when used as input to some (cryptographic) function, will denote a string identifying $x \in L$; E denotes a symmetric encryption algorithm, and $E_k(p)$ denotes the encryption of plaintext p with key k ; $s \parallel t$ denotes the concatenation of strings s and t ; $m \mid n$ means that integer m divides integer n without remainder; (m, n) denotes the greatest common divisor of integers m and n ; in particular, $(m, n) = 1$ means that m and n are co-prime.

We assume the existence of an RSA key generator, a randomized algorithm that takes a security parameter k as input and outputs a triple (n, e, d) such that:

- $n = pq$, where p and q are distinct odd primes;
- $e \in \mathbb{Z}_{\phi(n)}^*$, where $\phi(n) = (p-1)(q-1)$, $e > 1$, and $(e, \phi(n)) = 1$;
- $d \in \mathbb{Z}_{\phi(n)}^*$, where $ed \equiv 1 \pmod{\phi(n)}$.

We also assume the existence of an associated hash function, which maps elements of \mathbb{Z}_n^* (in some standard representation) to bit-strings of some desired length.

2.1. Information flow policies

Definition 1. An information flow policy is a tuple (L, \leq, U, O, λ) , where: (L, \leq) is a (finite) partially ordered set of security labels; U is a set of users; O is a set of (protected) objects; $\lambda : U \cup O \rightarrow L$ is a security function that associates users and objects with security labels.

Such policies were of particular interest in the mid-1970s through to the mid-1980s, and formed part of the Bell-LaPadula security model [5]. The basic semantics of the policy is that a user u can read an object o if $\lambda(u) \geq \lambda(o)$. Henceforth, we will represent an information flow policy (L, \leq, U, O, λ) as a pair (L, \leq) with the tacit understanding that U, O and λ are given.

Clearly, one way of implementing such a policy is to encrypt an object with security label $y \in L$ with a key $\kappa(y)$, and to provide all users with security label $x \geq y$ with the key $\kappa(y)$. Henceforth we assume that the policy will be implemented by encrypting objects and distributing keys to users, enabling them to decrypt the objects to which they should have access.

A consequence of using cryptographic methods to control access is that users may cache decrypted data objects locally. Hence, revoking a user’s assignment to a security level does not necessarily have an immediate effect on that user’s ability to read documents he has previously accessed. For this reason, lazy re-encryption – only re-encrypting data objects when they are modified – is a reasonable strategy to employ in this context.

2.2. Key assignment schemes

A *key assignment scheme* (KAS) is a means of implementing an information flow policy using cryptographic techniques. The simplest such scheme provides the user u the set of keys $\{\kappa(y) : y \leq \lambda(u)\}$. More sophisticated KASs seek to minimize the number of keys that need to be distributed to users. This entails either making certain additional information publicly available or providing each user with additional secret information (or both).

A recent paper formalized the characteristic features of a key assignment scheme [7]. In simple terms, there must be a way of choosing keys for each $x \in L$, a method for constructing public information, and a method for deriving keys. The paper also identified five generic constructions for key assignment schemes and compared these schemes according to the following criteria: amount of public storage required, amount of private storage required, complexity of key derivation, and complexity of key updates. We discuss two of these schemes in the next two sections. In both schemes a user u has a single key $\kappa(\lambda(u))$.

2.2.1. IKE KAS

An *iterative key encrypting* (IKE) KAS uses public information to enable a user to iteratively derive keys for which he is authorized. The trusted center:

- chooses $\kappa(x)$, $x \in L$, at random from the key space;
- publishes $\{E_{\kappa(x)}(\kappa(y)) : y \leq x : x, y \in L\}$.

Informally, the public information “encrypts” edges in the Hasse diagram: for any edge (x, y) , $\kappa(y)$ will be encrypted with $\kappa(x)$ and stored in the public information. Key derivation is performed by traversing a path in the Hasse diagram decrypting keys along the way. More formally, if $y < x$, then there exists a path $y = z_0 < z_1 \cdots < z_n = x$, $n \geq 1$, so a user with security label x can successively derive keys $\kappa(z_{n-1}), \dots, \kappa(z_0) = \kappa(y)$. Assuming a suitable encryption method is chosen, no user can feasibly compute keys for which he is not authorized.

The AFB scheme, due to Atallah, Frikken and Blanton is an example of an IKE KAS [2]. The key space is $\{0, 1\}^l$, for some integer l , and the public information is defined to be $\{\kappa(y) \oplus h(\kappa(x) \parallel y) : y \leq x\}$, where $h : \{0, 1\}^* \rightarrow \{0, 1\}^l$ is a hash function.

2.2.2. The Akl-Taylor scheme

The second generic scheme is a *node-based key assignment scheme*. Key derivation in a node-based KAS is direct and the user is only required to store a single key. The disadvantage of a node-based KAS is that key updates may be more disruptive than in an IKE KAS, because keys are defined in terms of public information. The earliest KAS (of any type) is due to Akl and Taylor; it remains the best known, simplest, and most effective node-based KAS [1]. The trusted center

- runs the RSA key generator to obtain (n, e, d) , of which only n is used;
- chooses a secret s at random from \mathbb{Z}_n^* ;
- chooses a mapping $v : L \rightarrow \mathbb{N}^*$ such that $v(x) \mid v(y)$ if and only if $y \leq x$;
- defines $\kappa(x) = s^{v(x)} \pmod n$;
- publishes $\{n\} \cup \{v(x) : x \in L\}$.

A user with security label x can derive $\kappa(y)$, $y \leq x$, using private information $\kappa(x)$ and public information $\nu(x)$ and $\nu(y)$, by computing

$$(\kappa(x))^{\frac{\nu(y)}{\nu(x)}} = (s^{\nu(x)})^{\frac{\nu(y)}{\nu(x)}} = s^{\nu(y)} = \kappa(y).$$

Akl and Taylor proved that no user (or set of users) can derive a key for which he is not authorized, provided ν is chosen appropriately. We discuss the security of key assignment schemes in Section 2.2.4. The canonical way of constructing ν is to associate a distinct prime $\alpha(x)$ with each $x \in L$ and to define

$$\nu(x) = \prod_{y \not\leq x} \alpha(y). \quad (1)$$

An example of this construction is shown in Figure 1. L is shown in Figure 1(a); the small italic numbers in Figure 1(b) denote the primes associated with each element of L ; and the larger numbers denote the value of $\nu(x)$ for each $x \in L$. Note that $\nu(x) \mid \nu(y)$ if and only if $x \geq y$.

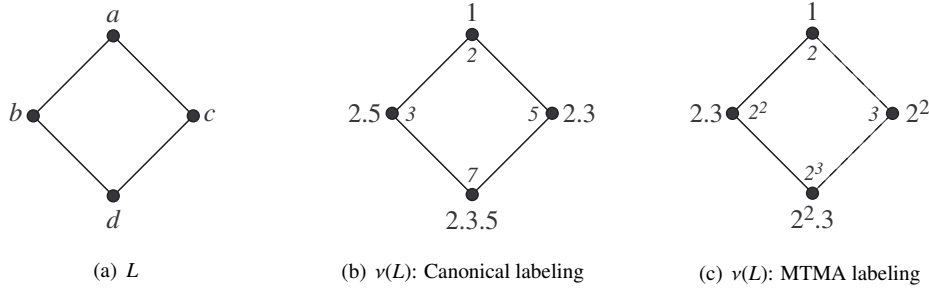


Figure 1: An example of the Akl-Taylor scheme

MacKinnon *et al* identified a different initial labeling $\alpha : L \rightarrow \mathbb{N}$, based on a decomposition of L into disjoint chains [14]. In other words, every element in L is a member of precisely one chain. We associate a distinct prime α_i with the i th chain. Then for $x \in L$, the j th element in the i th chain (where the maximal element in each chain is the first element), we define

$$\alpha(x) = \alpha_i^j \quad \text{and} \quad \nu(x) = \text{lcm}\{\alpha(y) : y \not\leq x\}.$$

An example of this labeling, which we call the MTMA labeling (after its creators, MacKinnon, Taylor, Meijer and Akl), is shown in Figure 1(c). The solid lines denote a chain ($a > b > d$) associated with the prime 2; the remaining node (c) forms a trivial chain associated with the prime 3. Note that the ν values satisfy condition (2).

2.2.3. Implementation issues

It has long been claimed that key derivation in the Akl-Taylor scheme may be expensive in practice (see [2, 12], for example) because it appears that a user must compute

$$\kappa(y) = \kappa(x)^{\frac{\nu(y)}{\nu(x)}}$$

to obtain $\kappa(y)$ from $\kappa(x)$, where $\nu(y)$ and $\nu(x)$ are the products of primes. This means that an expensive division operation is required followed by an exponentiation. However, it was noted

by Crampton *et al* [7] that the ν values were simply used to encode the structure of the poset, and that this encoding could be done in such a way as to avoid expensive division operations during key derivation.

Consider the definition of ν in the canonical labeling:

$$\nu(x) = \prod_{y \not\leq x} \alpha(x).$$

The product is taken over all elements that are not less than or equal to x . The set of elements less than or equal to x is (by definition) the order ideal generated by x . Hence, the product used to compute $\nu(x)$ is taken over the complement of the order ideal $\downarrow x$, which is an order filter.

We now choose some enumeration of the elements of $L = \{x_1, \dots, x_{|L|}\}$ and for each $x \subseteq L$, define the *characteristic tuple* (of x) $\chi_x = (b_1, \dots, b_{|L|})$, where $b_i = 0$ if $x_i \leq x$ and $b_i = 1$ otherwise. Hence, we may represent the elements of the filter $L \setminus \downarrow x$ by χ_x . Moreover, we may publish this information and the mapping α associating each $x_i \in L$ with a prime $\alpha(x_i)$. Now,

$$\nu(x) = \prod_{i=1}^{|L|} (\alpha(x_i))^{b_i}$$

where b_i is the i th element of χ_x . Hence,

$$\frac{\nu(y)}{\nu(x)} = \frac{\prod_{i=1}^{|L|} (\alpha(x_i))^{b'_i}}{\prod_{i=1}^{|L|} (\alpha(x_i))^{b_i}},$$

where b'_i is the i th element of χ_y . Hence

$$\frac{\nu(y)}{\nu(x)} = \prod_{i=1}^{|L|} (\alpha(x_i))^{b''_i},$$

where $b''_i = b'_i - b_i$. In other words, no division is required. We simply take the co-ordinatewise difference of the characteristic tuples of x and y and then multiply the appropriate primes together.

Taking our running example in Figure 1 and enumerating the elements of L in alphabetical order, the order ideal generated by b is $\{b, d\}$ and the order ideal generated by d is $\{d\}$. The characteristic tuples of b and d are $(1, 0, 1, 0)$ and $(1, 1, 1, 0)$, respectively. To compute $\kappa(d)$ from $\kappa(b)$ we compute $(1, 1, 1, 0) - (1, 0, 1, 0)$ to obtain $(0, 1, 0, 0)$, which means that we have to raise $\kappa(b)$ to the power of the prime associated with b to obtain $\kappa(d)$. This can be confirmed by looking at Figure 1(b). In short, no division of exponents is required. That is, there exist efficient implementations of the Akl-Taylor scheme.

We now observe that a similar type of representation can be used for the MTMA labeling. Recall that L is partitioned into i chains, the i th chain being associated with a distinct prime α_i , $\alpha(x) = \alpha_i^j$ and $\nu(x) = \text{lcm}\{\alpha(y) : y \not\leq x\}$. In other words, $\nu(x) = \prod_{i=1}^w \alpha_i^{a_i}$, where $a_i \in \mathbb{N}$. In this context, the characteristic tuple of x is defined to be $(a_1, \dots, a_w) \in \mathbb{N}^w$. In our example, we have two chains and we have

$$\chi_a = (0, 0), \quad \chi_b = (1, 1), \quad \chi_c = (2, 0), \quad \chi_d = (2, 1).$$

As before, division of exponents corresponds to subtraction of tuples. To derive $\kappa(d)$ from $\kappa(b)$, for example, we compute $(2, 1) - (1, 1) = (1, 0)$, which implies that we raise $\kappa(b)$ to the power of the prime associated with the first chain.

2.2.4. Security considerations

The fact that $v(x) \mid v(y)$ if, and only if, $x \geq y$ implies a user u with security label x cannot derive a key $\kappa(y)$, $y \not\leq x$, unless u can compute roots modulo n . This is known as the *RSA problem* and is believed to be as hard as factoring n : it is this *RSA assumption* on which the security of the RSA cryptosystem is based [15]. Akl and Taylor proved that no *set* of users can collude to derive a key for which none of them is authorized, provided v is chosen appropriately. In particular, we have the following result.

Theorem 2 (Akl and Taylor). $\kappa(y)$ can be feasibly computed from a set of keys $\{\kappa(x_1), \dots, \kappa(x_m)\}$ if, and only if, $\gcd\{v(x_1), \dots, v(x_m)\} \mid v(y)$.

The result is proved by showing that if this property did not hold then it would be feasible to solve the RSA problem. Hence, we choose v in such a way that for all $x \in L$,

$$\gcd\{v(y) : y \not\leq x\} \nmid v(x). \quad (2)$$

The canonical labeling defined in equation (1), $v(x) = \prod_{y \not\leq x} \alpha(y)$, satisfies property (2), as does the MTMA labeling.

Atallah *et al* examined the security of IKE KASs [2], introducing the notions of *key recovery* and *key indistinguishability*, which are analogous to plaintext recovery and plaintext indistinguishability in conventional security analyses. Informally, a scheme is secure against key recovery if it is infeasible for a group of users to pool key information to obtain a key for which none of them is authorized. In this sense, the Akl-Taylor scheme is secure against key recovery.

Key indistinguishability is believed to be an important consideration in IKE KASs [10]. Informally, if an attacker a is given a key k that purports to be $\kappa(x)$ for some $x \in L$ and a is able to decide (under reasonable computational assumptions) whether k is actually $\kappa(x)$ with some probability of success significantly greater than $1/2$, then the scheme does not have the key indistinguishability property. Clearly the basic IKE KAS does not have this property, since an attacker that knows $\kappa(y)$, for some $y \leq x$, can determine whether k is indeed $\kappa(x)$ by trying to derive $\kappa(y)$ from k and the public information. If he succeeds in deriving $\kappa(y)$, then he knows that $k = \kappa(x)$.

The IKE KASs described in this paper do not have the key indistinguishability property, although Atallah *et al* [2] and, independently, Fu *et al* [10] have developed straightforward modifications to existing schemes to obtain this property.¹ We do not make similar modifications to our schemes for ease of exposition.

Ateniese *et al* [3] extended the security analysis of Atallah *et al* to temporal key assignment schemes, and also considered active and passive adversaries. They showed that an active adversary has no advantage over a passive one for either key recovery or key indistinguishability. In other words, the security properties of an IKE KAS appear to be dependent only on the security properties of the algorithm used to encrypt keys, the results of which are made public. We hope to confirm these types of properties for our schemes in our future work.

¹Informally, users with security label x are provided with a secret $\sigma(x)$, rather than an encryption key $\kappa(x)$. The secret $\sigma(x)$ may be used to derive $\kappa(x)$ (using some suitable publicly known one-way function) and $\sigma(y)$ for all $y \leq x$ (using existing techniques for key assignment schemes). The point being that an attacker cannot deduce $\sigma(x)$ from $\kappa(x)$ and hence is unable to use key derivation (as described above) to distinguish the genuine $\kappa(x)$ from a random value.

3. Multi-key assignment schemes

We now introduce the concept of a *multi-key assignment scheme* (MKAS), which is a key assignment scheme that supports multiple keys for each security label. We assume there are a number of update events indexed by the natural numbers, and that the i th key is used to

- encrypt objects created between the i th and $(i + 1)$ th updates, and
- re-encrypt existing objects whose contents change between the i th and $(i + 1)$ th updates.

The i th time interval is the period between the i th and $(i + 1)$ th update.

If a user u is currently authorized to access an object o , then u is also authorized to access any object o' such that $\lambda(o) \geq \lambda(o')$. This implies that u must be able to compute each key that has been used for $\lambda(o')$. We also require that a user v revoked at the j th key update can not access any object created or amended since the j th update. We write $\kappa_i(x)$ to denote the i th key for security label x .² In other words, an authorized user u in time interval i can derive any key in the set $\{\kappa_j(l) : l \leq \lambda(u), 0 \leq j \leq i\}$.

There are two main considerations when designing a multi-key assignment scheme.

- *Is the number of updates determined in advance?* We say an MKAS is *bounded* if the number of updates is pre-determined, and *unbounded* otherwise.

Bounded MKASs are well suited for temporal access control policies, because the number of time periods will usually be fixed and known in advance. They are less appropriate for schemes intended to support lazy re-encryption, since the number of updates will not be predictable. Unbounded MKASs are far more suitable for lazy re-encryption.

- *Can $\kappa(x)$ be updated independently of $\kappa(y)$?* We say an MKAS is *synchronous* if all keys are updated at the same time, and *asynchronous* otherwise.

Synchronous MKASs are particularly suitable for temporal access control policies, because each time interval requires a fresh set of keys, one for each security label. Asynchronous MKASs may well be more suitable for implementing lazy re-encryption, because it may only be necessary to update a small number of keys following the revocation of a user.

The key observation to make prior to introducing the schemes we have devised is to note that the requirements can be regarded as having two dimensions: a “policy dimension”, determined by L ; and a “temporal dimension”, which can be modeled as a chain of (update) events. In the next section we consider three different ways of realizing bounded schemes, each having different properties and advantages. In Section 3.2 we introduce two unbounded synchronous schemes. In Section 3.3 we describe two unbounded asynchronous schemes. We conclude with a summary and comparison of the respective advantages of our schemes.

3.1. Bounded MKASs

We assume that there are a maximum of m updates, where m is chosen in advance by the scheme administrator, and that the keys for $L' \subseteq L$ are updated. The basic technique is to construct an IKE KAS for (L, \leq) for each time interval. In addition, we provide a mechanism for computing $\kappa_j(x)$ from $\kappa_i(x)$ for any $j < i$. Note that the schemes described can be either synchronous or asynchronous; a synchronous scheme is obtained by setting $L' = L$ at each update.

²The initialization of the scheme is assumed to be the 0th “update event”.

3.1.1. Node-based iterative MKAS

In this scheme we choose the keys $\kappa_0(x), \dots, \kappa_m(x)$ in such a way that $\kappa_j(x)$ can be derived directly from $\kappa_i(x)$, for all $j < i$. The construction is based on a very simple instance of the Akl-Taylor scheme in which L is a chain of m elements.

The trusted center first obtains a large compound integer n using an RSA key generator. For each $x \in L$, the trusted center

- chooses a secret $s(x) \in \mathbb{Z}_n^*$, such that for all $y \in L$, $(s(x), s(y)) \neq 1$ if and only if $x = y$;
- defines $\kappa_j(x) = (s(x))^{2^{m-j}} \pmod n$;
- defines an IKE for L and publishes $\{E_{\kappa_0(x)}(\kappa_0(y)) : y \in L, y \preceq x\}$.

We call the set of keys $\kappa_0(x), \dots, \kappa_m(x)$ an *Akl-Taylor key chain*.

Let the set of keys changed at the i th update be L'_i . Following the i th update, for each $x \in L'_i$, the trusted center obtains the next key for x and updates the public information. In particular, the trusted center re-computes and replaces $\{E_{\kappa_i(x)}(\kappa_i(y)) : x \in L', y \in L, y \preceq x\}$ and $\{E_{\kappa_i(x)}(\kappa_i(y)) : y \in L', x \in L, y \preceq x\}$. As a consequence, the amount of public information required by this scheme remains constant.

In order to recover $\kappa_j(y)$, given $\kappa_i(x)$, with $y \preceq x$ and $j \leq i$, the user first computes $\kappa_i(y)$ using the public information. The user then computes

$$(\kappa_i(y))^{2^{i-j}} = (\kappa_i(y))^{\frac{2^{m-j}}{2^{m-i}}} = (s(y)^{2^{m-i}})^{\frac{2^{m-j}}{2^{m-i}}} = s(y)^{2^{m-j}} = \kappa_j(y).$$

In other words, the user first uses the current IKE KAS to compute the current key for the desired security label, and then computes the key for the desired time interval.

Note that the node-based KAS that is used for the keys $\kappa_0(x), \dots, \kappa_m(x)$ has the property defined in (2). Hence, it is not possible for a user to compute a future key from an existing one.

3.1.2. Hash-chain iterative MKAS

In this scheme, keys are chosen in such a way that $\kappa_j(x)$ can be derived iteratively from $\kappa_i(x)$. For each $x \in L$, the trusted center chooses a key $\kappa_m(x) \in \{0, 1\}^l$ and defines $\kappa_{i-1}(x) = h(\kappa_i(x))$, where $h : \{0, 1\}^* \rightarrow \{0, 1\}^l$ is a hash function. We call

$$\kappa_m(x), \kappa_{m-1}(x) = h(\kappa_m(x)), \dots, \kappa_i(x) = h^{m-i}(\kappa_m(x)), \dots, \kappa_0(x) = h^m(\kappa_m(x))$$

a *hash chain* for x . This type of approach has been used in one-time password schemes [11], and means that a user cannot compute a future key from an existing one.

In order to recover $\kappa_j(y)$, given $\kappa_i(x)$, with $y \preceq x$ and $j < i$, the user first computes $\kappa_i(y)$ using the public information for the i th IKE KAS, as before. The user then iteratively derives $\kappa_j(y)$, by computing $\kappa_{i-1}(y) = h(\kappa_i(y)), \dots, \kappa_j(y) = h(\kappa_{j+1}(y))$.

3.1.3. Example

Figure 2 illustrates how an asynchronous scheme would evolve over time, using the poset depicted in Figure 1(a). Labels for which new keys are generated are represented by unfilled nodes; existing keys are represented by filled nodes; edges that need to be updated are represented by broken edges. At any moment in time, the public information is used to compute current keys; previous keys are computed using an Akl-Taylor or hash chain.

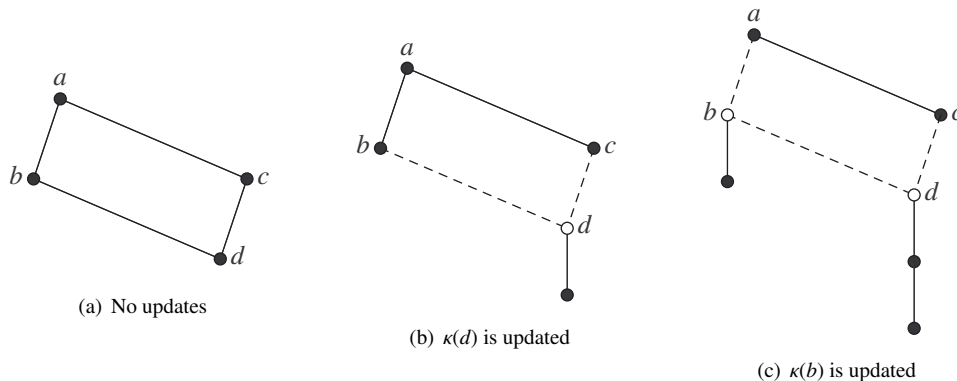


Figure 2: Key updates in an asynchronous scheme

Prior to any updates, a single key exists for each security label. A user with security label d is then removed from the scheme, meaning that $\kappa(d)$ has to be refreshed. Since d is the minimal element in the poset, no other keys need to be changed. In the asynchronous scheme we must recompute the public information $E_{\kappa(b)}(\kappa(d))$ and $E_{\kappa(c)}(\kappa(d))$, the “edge encryptions” for (d, b) and (d, c) . This is depicted in Figure 2(b).

Then a user with security label b is removed from the scheme, meaning that $\kappa(b)$ has to be refreshed. Since $d < b$, we must also update $\kappa(d)$. In addition, we must re-compute all edges from elements immediately greater than either b or d . This is shown in Figure 2(c). Notice that if, instead of removing the user with security label b from the scheme, we changed her security label to c , then the update set is $\{b\}$ rather than $\{b, d\}$ (since $d < c$); in this case, we would only update $\kappa(b)$ and re-compute $E_{\kappa(a)}(\kappa(b))$ and $E_{\kappa(b)}(\kappa(d))$.

3.1.4. MTMA MKAS

Finally, we introduce a bounded scheme based on the MTMA labeling. We note that we can regard the set of keys $\{\kappa_i(l) : l \in L, 0 \leq i \leq m\}$ as being associated with the nodes in the poset $L \times T$, where T is a chain of m elements. We write (x, i) for the i th element in the chain for node x . It is obvious, therefore, that we can decompose $L \times T$ into $|L|$ chains, each containing $m + 1$ elements.

Hence, we can define an MKAS in which all keys are defined using the Akl-Taylor scheme with the MTMA labeling, where each node is associated with a different prime. More specifically, the trusted center first obtains a large compound integer n and chooses a prime $\alpha(x)$ for each $x \in L$. Then the trusted center

- selects a secret $s \in \mathbb{Z}_n^*$;
- defines $\alpha(x, i) = \alpha(x)^{m+1-i}$, $0 \leq i \leq m$;
- computes and publishes

$$v(x, i) = \prod_{\substack{y \not\leq x \\ j > i}} \alpha(y, j);$$

- defines $\kappa_i(x) = s^{v(x,i)}$.

If a user u has $\kappa_i(x)$, then she can directly compute

$$\kappa_j(y) = \kappa_i(x)^{\frac{v(y,j)}{v(x,i)}}$$

for any $y \leq x$ and any $j \leq i$. In other words, this scheme has the advantage that key derivation is always performed in a single step, at the expense of requiring $\mathcal{O}(m|L|)$ storage for public information. Moreover, u cannot compute $\kappa_j(y)$ for any $y \not\leq x$ or any $j > i$ since this scheme is just an instance of the Akl-Taylor scheme (using the MTMA labeling) for the poset $L \times T$.

3.2. Unbounded synchronous MKASs

We now assume that the number of updates is not known in advance, and that all keys are updated at the same time. Note that any unbounded synchronous MKAS can be used as a bounded synchronous MKAS.

A feature of the Akl-Taylor KAS is that it is possible to update all the keys simply by choosing a new system secret. No public information needs to change, unlike any IKE KAS. Using this observation, we propose two synchronous MKAS. In the first scheme, user u is required to keep all earlier versions of $\kappa(\lambda(u))$. In the second scheme, u only requires a single key.

3.2.1. Node-based MKAS

Initially, the trusted center

- runs the RSA key generator to obtain n ;
- chooses a system secret $s_0 \in \mathbb{Z}_n^*$ at random;
- defines a mapping $\nu : L \rightarrow \mathbb{N}^*$ such that $\nu(x) \mid \nu(y)$ if and only if $y \leq x$;
- publishes $\{n\} \cup \{\nu(x) : x \in L\}$;
- defines $\kappa_0(x) = s_0^{\nu(x)} \pmod n$.

Following the i th update event, the trusted center

- chooses a new system secret $s_i \in \mathbb{Z}_n^*$, where $(s_i, s_j) = 1$ for all $j < i$;
- defines $\kappa_i(x) = s_i^{\nu(x)} \pmod n$.

A user with security label x can derive $\kappa_j(y)$, $y \leq x$, using $\kappa_j(x)$ and the public information $\nu(x)$ and $\nu(y)$.

Two significant advantages of this scheme are that the number of update events does not need to be defined before the scheme is instantiated, and the public information does not need to be updated. Of course, the user now accumulates keys as the scheme develops, so private storage requirements increase and the user is required to manage a number of different keys.

3.2.2. Node-based RSA MKAS

A refinement of the scheme described above is to use a deterministic method for generating new system secrets. Of course, we must ensure that future keys can not easily be computed from existing ones. The scheme is initialized in the same way as in Section 3.2.1, except that the trusted center publishes e as well as n . Following the i th update event, the trusted center

- computes a new system secret $s_i = s_{i-1}^d \pmod n$;
- defines $\kappa_i(x) = s_i^{v(x)} \pmod n$.

Note that

$$(\kappa_i(y))^e = (s_i^{v(y)})^e = (s_i^e)^{v(y)} = ((s_{i-1}^d)^e)^{v(y)} = (s_{i-1}^{de})^{v(y)} = s_{i-1}^{v(y)} = \kappa_{i-1}(y).$$

Hence the user now only needs to retain the most recent key: he can derive $\kappa_j(y)$ from $\kappa_i(x)$ ($y \leq x$, $j \leq i$) by first computing $\kappa_i(y)$ from the public information, and then iteratively computing $\kappa_{j'}(y)$, $i > j' \geq j$. The advantages of this scheme are that the user only requires a single key and the public information is fixed. The disadvantage is that key derivation is iterative if the key is for an earlier time interval.

Note that a user cannot feasibly compute a future key from an existing one, as this would require the ability to compute d , which in turn requires the ability to factorize n . In other words, the security of future keys relies on the same assumptions as the RSA cryptosystem.

3.3. Unbounded asynchronous MKASs

It would be convenient if we could adapt the schemes described in Section 3.2 to construct an unbounded asynchronous MKAS. However, such an adaptation is not possible, in general, because keys are a function of public information and a secret value.

As we have seen in Section 3.2, it is easy to change all keys in an Akl-Taylor scheme by changing the secret value. However, changing the prime associated with one element of L may cause changes to a number of different keys. To see this, consider the poset in Figure 1 and the effect of changing the prime associated with b to 11. This has the effect of changing both $\kappa(c)$ and $\kappa(d)$ because $v(c)$ and $v(d)$ are both defined in terms of the prime associated with b .

3.3.1. Iterative RSA MKAS

Hence, we need to adopt an alternative approach to develop an unbounded asynchronous MKAS. Informally, we use a method similar to that in Section 3.2 to create key chains for each $x \in L$, and link these chains using an IKE KAS. More formally, the trusted center

- runs the RSA key generator to produce (n, e, d) ;
- for each label $x \in L$, chooses secret $\kappa_0(x)$ at random from \mathbb{Z}_n^* ;
- publishes $\{n, e\}$;
- constructs an IKE KAS for L .

Following the i th key revocation for security label x , the trusted center

- computes, $\kappa_i(x) = \kappa_{i-1}^d \pmod n$;

- replaces $\kappa_{i-1}(x)$ in the IKE KAS with $\kappa_i(x)$;
- updates the public information for the IKE KAS.

Note that

$$(\kappa_i(y))^e = ((\kappa_{i-1}(y))^d)^e = \kappa_{i-1}(y). \quad (3)$$

Suppose now that user u with $\kappa_i(x)$ wishes to derive $\kappa_j(y)$, $x \geq y$, $i > j$. Then u can compute $\kappa_i(y)$ using the public information in the IKE KAS, and can now iteratively compute $\kappa_j(y)$ using the computation defined in (3).

3.3.2. Iterative MKAS

The second scheme employs a single IKE KAS. The difference is that L now evolves over time. Using the example depicted in Figure 2, each of the three Hasse diagrams would represent different posets, rather than the same poset with different numbers of keys associated with each of the nodes. Following an update to $\kappa(d)$, we would construct an IKE KAS for the poset depicted in Figure 3(a). Similarly, following an update to $\kappa(b)$, we would construct an IKE KAS for the poset depicted in Figure 3(b).

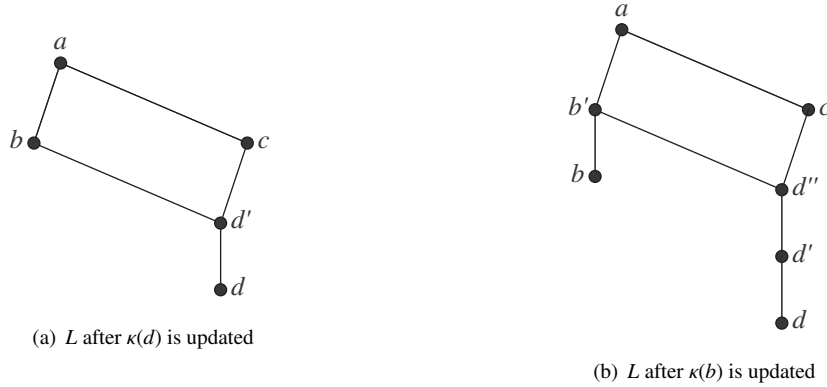


Figure 3: Dynamic information flow policies for an unbounded asynchronous MKAS

More formally, for each node x in the update set, we need to:

- insert a new node x' ;
- add a new edge (x, x') ; and
- for each edge (x, y) , we need to insert a new edge (x', y) and delete (x, y) .

A key for x' needs to be selected and the public information for the new edges computed and published.

The advantage of this scheme is its simplicity. The disadvantage is that the number of nodes and edges, and hence the amount of public information, grows over time.

3.4. Summary

We summarize the characteristics of our schemes in Figure 4. The basic design features of each scheme are described in Figure 4(a). Note, in particular, the MTMA scheme (Section 3.1.4), which uses a single method for both the policy and temporal dimensions. We only indicate whether a scheme is unbounded or not and whether a scheme is asynchronous or not: any unbounded scheme can be used as a bounded scheme, and any asynchronous scheme can be used as a synchronous scheme.

Scheme	Dimension		Unbounded	Asynchronous
	Policy	Temporal		
§3.1.1	IKE	Akl-Taylor	No	Yes
§3.1.2	IKE	Hash chain	No	Yes
§3.1.4	Akl-Taylor (MTMA labeling)		No	Yes
§3.2.1	Akl-Taylor	n/a	Yes	No
§3.2.2	Akl-Taylor	RSA	Yes	No
§3.3.1	IKE	RSA	Yes	Yes
§3.3.2	IKE	IKE	Yes	Yes

(a) Design characteristics

Scheme	Derivation		Public Update	Storage	
	Policy	Temporal		Public	Private
§3.1.1	Ind	Dir	Yes	$ E $	1
§3.1.2	Ind	Ind	Yes	$ E $	1
§3.1.4	Dir		No	$m L $	1
§3.2.1	Dir	n/a	No	$ L $	u
§3.2.2	Dir	Ind	No	$ L $	1
§3.3.1	Ind	Ind	Yes	$ E $	1
§3.3.2	Ind	Ind	Yes	$u E $	1

(b) Operational characteristics

Figure 4: Characteristics of our schemes

Figure 4(b) summarizes a number of operational characteristics, including those identified by Crampton *et al* [7] as being of interest when assessing key assignment schemes. The derivation column includes the key derivation method within the policy and temporal dimensions: we write “Dir” to denote direct key derivation and “Ind” to denote indirect key derivation. Some schemes require public information to be updated following a change of keys; this is indicated in the column headed “Public Update”. Storage requirements indicate the number of items of data that need to be stored in the worst case following u updates; E denotes the edge set in the Hasse diagram of L . The storage required is proportional to the value given in the table (the constant of proportionality being determined by the cryptographic primitives chosen to implement the scheme).

3.5. Discussion

It is difficult to make a definitive statement about which of these schemes is the best, because “best” may depend on the context in which a scheme is to be deployed. In an ideal world,

a scheme would: have direct key derivation and low storage costs; be unbounded and asynchronous; not require updates to public information; and each user would have a single key. Of course, none of these schemes have all these features.

It is not clear whether asynchronous schemes offer any significant advantages: again, this will probably depend on context. It should be noted that many updates for label l will require updates for all $l' \leq l$. In some scenarios, therefore, it may be simpler to use a synchronous scheme. Nevertheless, if we can assume (or know) that most of the update activity will be for relatively junior security labels, then there may well be a distinct advantage to using this incremental approach. This may well be a reasonable assumption: there will probably be fewer users with relatively senior labels, and those users are probably less likely to be re-assigned to new security labels or removed from the scheme.

Of the bounded schemes, the one based on the MTMA labeling has some appealing features, particularly compared to the other two bounded schemes: key derivation is direct and no updates are required for public information. As we showed in Section 2.2.3, there exist implementations of the Akl-Taylor scheme with the MTMA labeling that permit fast key derivation.

As we have noted, temporal access control policies do not require an asynchronous scheme and generally will not require the scheme to be unbounded. Hence, suitable choices for enforcing such policies would be the schemes described in Sections 3.1.4 and 3.2.2. The trade-off here is between partial iterative key derivation and amount of public data. Generally, the storage required for public data will be an insignificant fraction of the storage available (see [16] for a discussion of such considerations), so direct key derivation is likely to be preferred. In contrast, if we wish to support lazy re-encryption, then we should choose an asynchronous scheme. We will also probably require an unbounded scheme, so a suitable choice in these circumstances would be one of the schemes described in Section 3.3.

4. Conclusion

Updating keys is known to be an issue in cryptographic access control when the same encryption key is used to encrypt many different data objects. This problem occurs in cryptographic storage file systems, where techniques such as key regression have been used so that re-encryption of data objects is not necessarily performed when the encryption key is refreshed. The problem is even more pressing in key assignment schemes for hierarchical access control, in which a number of different keys may need to change at the same point in time, meaning that even higher volumes of data may need to be re-encrypted at some point. We have introduced the idea of multi-key assignment schemes and shown that such schemes are useful in supporting lazy re-encryption and temporal access control policies. These are the first such schemes in the literature, and represent a considerable advance on existing approaches to key updates in KASs.

Nevertheless, a number of opportunities for further work exist. One obvious shortcoming of our approach to enforcing temporal access control policies is that a user with a key for $x \in L$ can derive all previous keys for x . In some cases, it may be that a user can only access material from the point at which she starts paying for content. In such cases, it must not be possible for the user to derive keys for time points before she started paying. It is not clear whether a simple modification to our schemes is sufficient to enforce such requirements. A priority, then, will be to investigate this question.

We noted in Section 2.2.4 that formal security analyses of KASs are beginning to appear in the literature. An informal analysis suggests that the schemes presented in this paper are secure

against key recovery. However, none of the schemes has the property of key indistinguishability. Existing work suggests that a relatively simple extension to our schemes will be sufficient to obtain the key indistinguishability property. An important part of our future work will be to undertake a formal analysis of our schemes and to modify them, where necessary, using similar methods to those found in [2, 3, 10].

On a more practical note, it would be of considerable practical interest to adapt the architecture used in cryptographic file systems and implement a key assignment scheme for a hierarchical file system. Similarly, it would be interesting to use XML encryption to implement hierarchical access control for XML data.

Acknowledgements. The author would like to thank the anonymous referees for their constructive and instructive comments.

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, N. Fazio, and K.B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Transactions on Information and System Security*, 12(3):1–43, 2009.
- [3] G. Ateniese, A. De Santis, A.L. Ferrara, and B. Masucci. Provably-secure time-bound hierarchical key assignment schemes. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 288–297, 2006.
- [4] M. Backes, C. Cachin, and A. Oprea. Secure key-updating for lazy revocation. In *Proceedings of 11th European Symposium on Research in Computer Security*, pages 327–346, 2006.
- [5] D.E. Bell and L. LaPadula. Secure computer systems: Mathematical foundations. Technical Report MTR-2547, Volume I, Mitre Corporation, Bedford, Massachusetts, 1973.
- [6] 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.
- [7] 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.
- [8] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, United Kingdom, 1990.
- [9] D.E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.
- [10] K. Fu, S. Kamara, and Y. Kohno. Key regression: Enabling efficient key distribution for secure distributed storage. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2006*, 2006.
- [11] N.M. Haller. The S/KEY one-time password system. In *Proceedings of the 1994 Symposium on Network and Distributed System Security*, pages 151–157, 1994.
- [12] L. Harn and H.Y. Lin. A cryptographic key generation scheme for multilevel data security. *Computers and Security*, 9(6):539–546, 1990.
- [13] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *Proceedings of the FAST '03 Conference on File and Storage Technologies*, pages 29–42, 2003.
- [14] S.J. MacKinnon, P.D. Taylor, H. Meijer, and S.G. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Transactions on Computers*, C-34(9):797–802, 1985.
- [15] R. Rivest and B. Kaliski. The RSA problem. In H.C.A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*, pages 532–535. Springer, 2005.
- [16] H. Rowe and J. Crampton. Avoiding key redistribution in key assignment schemes. In *Proceedings of the Fourth International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 127–140, 2007.