

Avoiding Key Redistribution in Key Assignment Schemes

Harry Rowe* and Jason Crampton

Royal Holloway, University of London
{a.h.rowe, jason.crampton}@rhul.ac.uk

Abstract. A key assignment scheme is a model for enforcing an information flow policy using cryptographic techniques. Such schemes have been widely studied in recent years. Each security label is associated with a symmetric encryption key: data objects are encrypted and authorised users are supplied with the appropriate key(s). However, updates to encryption keys pose a significant problem, as the new keys have to be issued to all authorised users. In this paper, we propose three generic approaches to key assignment schemes that remove the problem of key redistribution following key updates. We analyse the overheads incurred by these approaches and conclude that these overheads are negligible in practical applications.

Keywords: key assignment schemes, key redistribution, hierarchical access control.

1 Introduction

There are a number of situations in which access control is implemented using cryptographic techniques. Such an approach is useful when the (protected) 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. [1] cite content distribution networks, such as Akamai and BitTorrent, as examples where this kind of technique might be useful.

A key assignment scheme is a form of cryptographic access control [2] that seeks to implement a no-read-up information flow policy [3]. Users and objects are assigned security labels; a user may have read access to an object if and only if the user's security label is at least as high as 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 the user's security label.

The data owner encrypts protected objects and supplies authorised users with the appropriate keys. Hence, many users may have a copy of the same key,

* The research of Harry Rowe is supported by Microsoft Research through its PhD Scholarship programme.

enabling each user to decrypt a number of different protected objects. One of the main practical problems with key assignment schemes arises when a user's authorisation is revoked: each of the keys that this user has been given or can derive is "compromised". Hence it is necessary to create new encryption keys and to securely redistribute the new keys to all remaining authorised users. (Of course all objects also need to be re-encrypted.)

In this paper we examine the assumptions that have typically been made when designing key assignment schemes. In particular, we are interested in the practical implications of the design choices. We conclude that there are number of ways in which we can design key assignment schemes so that key redistribution is not required. Of course, one would expect such a scheme to have certain disadvantages compared to existing schemes; there is always a trade-off. However, we believe that in purely practical terms, the additional overheads incurred by our schemes will be perfectly acceptable in practice, and are more than offset by the fact that we no longer need to be concerned with key distribution.

The rest of this paper is organised as follows: Sect. 2 reviews existing KASs and explores their shortcomings. We introduce our new schemes in Sect. 3 and analyse their performance. Sect. 4 discusses the optimality of our preferred scheme and we discuss proposed optimisations for existing KASs in Sect. 5. Finally, Sect. 6 concludes this 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 may write $x \geq y$ whenever $y \leq x$. We say x *covers* y , denoted $y < x$, 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 $(L, <)$ [4]. A simple Hasse diagram is shown in Fig. 1(a). We will write e to denote the cardinality of the covering relation $<$ (that is, the number of edges in the Hasse diagram of L), and e^* to denote the cardinality of the partial order relation \leq . In general e and e^* are $\mathcal{O}(l^2)$, where l is the cardinality of L , although in certain special cases (as when L is a total order, for example) e is $\mathcal{O}(l)$ and e^* is $\mathcal{O}(l^2)$.

Henceforth we adopt the following conventions: x , when used as input to some (cryptographic) function, will denote a string identifying the security label x ; H denotes a hash function; E denotes a symmetric encryption algorithm; and $E_k(m)$ denotes the encryption of message m with key k .

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 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, and formed part of the famous Bell-LaPadula security model. The basic operation of the policy is that a user u can read an object o if $\lambda(u) \geq \lambda(o)$. Clearly, one way of implementing such a policy is to encrypt an object with security label $y \in L$ with a key $k(y)$ and provide all users with security label $x \geq y$ with the key $k(y)$. 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. 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.

2.1 Key Assignment Schemes

Most key assignment schemes seek to minimise the number of keys that need to be distributed to users. This entails either making certain additional information public or providing each user with additional secret information (or both). A recent paper formalised the characteristic features of a key assignment scheme [2]. We summarise this approach below.

In general, a *key assignment scheme* (or *scheme*) for an information flow policy (L, \leq) defines three algorithms, the first two being run by the *scheme administrator*, the third by end users:

- **makeKeys** returns a labelled set of encryption keys $(\kappa(x) : x \in L)$, which we denote by $\kappa(L)$;
- **makePublicData** returns some set of data Pub that is made public by the scheme administrator;
- **getKey** takes $x, y \in L, \kappa(x)$ and the public data, and returns $\kappa(y)$ whenever $y \leq x$.

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. It was concluded that there were two generic constructions that offered the best trade-offs with regard to these criteria. We discuss these constructions in the next two sections. In both schemes a user u has a single key $\kappa(\lambda(u))$.

IKE KAS. An *iterative key encrypting* (IKE) key assignment scheme (KAS) uses public information to enable a user to iteratively derive keys for which he is authorised. An IKE KAS has the following characteristic features.

- $\kappa(x), x \in L$, can be chosen at random from the key space;
- Public information $Pub_I = \{E_{\kappa(x)}(\kappa(y)) : y \leq x : x, y \in L\}$;
- If $y < x$, a user with security label x can use $\kappa(x)$ to obtain $\kappa(y)$ by decrypting the public information $E_{\kappa(x)}(\kappa(y))$;
- If $y < x$, there exists a path $y = z_0 \leq z_1 \leq \dots \leq z_n = x, n \geq 1$, so a user with security label x can successively derive keys $\kappa(z_{n-1}), \dots, \kappa(z_0) = \kappa(y)$.

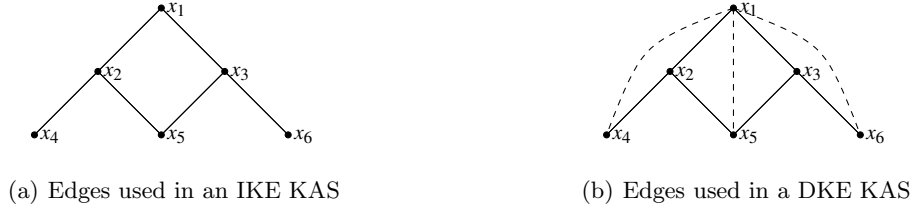


Fig. 1. The directed graphs used in key assignment schemes; the transitive relationships in the partially ordered set are denoted by broken lines

In all the KASs we consider in this paper, each datum of public information corresponds to an edge in some graph. In the case of an IKE KAS, this graph is $(L, <)$. Hence, the public information storage requirements for a KAS is proportional to the total number of edges in the graph, and we use the terms synonymously. The edges used in an IKE KAS are illustrated in Fig. 1(a),

DKE KAS. A *direct key encrypting* (DKE) KAS uses public information to enable a user to directly derive keys for which he is authorised. Such a scheme is shown in Fig. 1(b), and has the following characteristic features.

- $\kappa(x)$, $x \in L$ can be chosen at random from the key space;
- Public information $Pub_D = \{E_{\kappa(x)}(\kappa(y)) : y < x : x, y \in L\}$;
- If $y < x$, a user with security label x can use $\kappa(x)$ to obtain $\kappa(y)$ by decrypting the public information $E_{\kappa(x)}(\kappa(y))$.

The Atallah, Frikken and Blanton (AFB) KAS. The basic AFB scheme, proposed by Atallah et al. [5], is an example of an IKE KAS, which is secure against collusion, simple to set up and maintain, allows easy modifications to the hierarchical structure, and is efficient in terms of its storage requirements and key derivation time. In our notation, the AFB scheme can be represented in the following way:

- keys are chosen at random from $\{0, 1\}^m$, where m is a positive integer;
- $Pub = \{\kappa(y) - H(\kappa(x), y) : y < x : x, y \in L\}$, where $H : \{0, 1\}^* \rightarrow \{0, 1\}^m$ is a hash function.

Note that the AFB scheme can be transformed into a DKE KAS by setting $Pub = \{\kappa(y) - H(\kappa(x), y) : y < x : x, y \in L\}$, allowing a single step key derivation.

2.2 Implementation Considerations

In order to meaningfully evaluate a KAS, it is necessary to consider its proposed applications. Without reference to particular implementations, the expected purpose of a KAS is to facilitate controlled access to data resources, such as a *cryptographically protected file system* (CPFS) hosted on a remote file server.

The use of cryptography to enforce access control policies is most applicable in systems with distributed components whose security cannot be guaranteed and in which the data must be transmitted over untrusted communication channels.¹

Thus, we assume that (i) objects must be decrypted, and (ii) a publicly accessible resource server exists, hosting an information store of significant size. With these points in mind, it becomes clear that certain properties of KASs are less important than might be expected from a purely abstract analysis.

Key Derivation Time. The *key derivation time* is the average number of key derivations the user is required to compute to obtain a particular key. In an IKE KAS, this is the average path length in the Hasse diagram of the poset (L, \leq) , which is $\mathcal{O}(l)$ in general. Naively, this would seem to be a significant disadvantage when compared to the DKE KAS, where key derivation always requires a single step.

However, the process of deriving a key requires the use of only the most efficient cryptographic primitives – typically one-way hash functions as in the AFB scheme [5] – and involves decrypting keys of ~ 256 bits. Once the key is obtained, slower symmetric primitives would be used to decrypt the actual data objects, the size of which will be orders of magnitude larger than the encryption keys. That is, the time taken to derive encryption keys, even in an IKE KAS, will be negligible compared to the time taken to actually decrypt the data object.

Public Storage Requirements. The *public storage requirements* for a KAS can be expressed as the number of edges required to be stored publicly: this is proportional to e^* in the case of a DKE KAS and proportional to e in the case of an IKE KAS. The difference between e^* and e is $\mathcal{O}(l^2)$ in general, so this could be regarded as an important factor in deciding whether to implement an IKE KAS or DKE KAS.

However, the edges in question are the same size as the keys, ~ 256 bits. If we take an extreme example, where L is a totally ordered set of 100 security levels, the DKE KAS would require no more than ~ 150 kB of storage space ($\binom{100}{2}$ edges + 100 node labels), compared with the IKE KAS requirements of ~ 3 kB. The actual cost, in real terms, to the organisation of storing the extra 150 kB of information is negligible, particularly when compared with the overall storage cost necessarily incurred by the target file system. Thus, the extra public storage requirements incurred by a DKE KAS are effectively irrelevant to the analysis.

Complexity of Key Updates. Whether through key compromise, user revocation or routine security precautions, it will be necessary to change some or all of the keys at certain points in time. Updating the public information is a relatively simple task, involving changes to a single public information store.

The complexity is essentially identical in an IKE KAS and a DKE KAS. In order to revoke a single user with security level x , we must update $\kappa(y)$ for all

¹ Significant en/decryption overheads are inevitably incurred and are likely to be unacceptable in a localised system, in which all components are trusted and where secure channels exist between each component.

$y \leq x$ (since we must assume that the user may have derived and cached any of these keys). Hence, every user with a security label $y \leq x$ will need to receive the new key for y .

Note, that this is still a major improvement on a *trivial key assignment scheme* (TKAS) [2], in which user u with security label x is given the set of keys $\{\kappa(y) : y \leq x\}$. In a TKAS, revoking u means that any user v such that $\lambda(v) \geq x$ or $\lambda(v) \leq x$ will require at least one new key. The advantage of having information (edges) stored publicly (as in IKE and DKE KASs), is that each user has a single key and only users v such that $\lambda(v) \leq x$ require a new private key when $\kappa(x)$ is updated; the remainder of the key update process is achieved by updating the public information.

2.3 Remaining Difficulties and Motivation

It is clear that there are advantages in using a KAS to control access to a CPFS over traditional methods of access control in distributed file systems. Somewhat ironically, less cryptographic overhead is incurred in a CPFS, since there is no need to establish a secure channel (which requires the server to encrypt the data and the client to decrypt) as opposed to a single decryption by the client when using a CPFS. There is also no need for an online *authorisation server* (AS) and (logically distinct) *reference monitor*, thus removing these potential bottlenecks and shrinking the *trusted computing base* [6] to just the client machine.

Although issues such as key derivation time and public storage requirements are largely irrelevant in practical terms, as we have explained in the previous sections, the private key redistribution required following key updates is a serious consideration in practical CPFS implementations. Although the scale of this problem is reduced in schemes such as the AFB KAS, it is still far more complex (from the scheme administrator's point of view) than simply removing a particular user's authorisation from the AS (which has no effect on other legitimate users).

Secure key distribution has always been a fundamental issue in applied cryptography. The development of asymmetric cryptographic techniques in the 1970s was expected to remove or at least simplify this problem. However, it has become apparent that the robust implementation of such techniques require a dedicated infrastructure to guarantee the authenticity of public keys. Public key infrastructures are costly to implement and manage. Our motivation is therefore to design a KAS with similar security and efficiency properties as existing schemes, but one which obviates all private key redistribution.

3 Avoiding Key Redistribution in KASs

We assume that each user $u \in U$ shares a symmetric key, $k(u)$, with the scheme administrator. This is a reasonable assumption, given that we must have *a priori* knowledge of all the users in the system in order for them to be assigned to security labels, and we would require some way of securely transmitting $\kappa(\lambda(u))$ to u for *any* KAS. In our scheme, we publish information tailored to allow each

user to derive their permitted encryption keys. We can do this in several ways, balancing key derivation time with the public storage requirements, analogous to the tradeoff between the IKE and DKE KASs.

3.1 User-Based KASs

In all our schemes the set of encryption keys $\{\kappa(x) : x \in L\}$ and the set of private user keys $\{k(u) : u \in U\}$ can be chosen at random from some appropriate key space. The public information requirements of each scheme are described in the following sections.

User-based Iterative Key Encrypting KAS (UIKE KAS). As the name implies, this scheme is analogous to an IKE KAS as only strictly necessary edges are included in the public information. The “graph” of the scheme includes an edge from each user to their security level and the edges in the covering relation on L .

- $Pub_{UI} = \{E_{k(u)}(\kappa(\lambda(u))) : u \in U\} \cup \{E_{\kappa(x)}(\kappa(y)) : y \leq x : x, y \in L\}$;
- The user first obtains $\kappa(\lambda(u))$ by decrypting $E_{k(u)}(\kappa(\lambda(u))) \in Pub_{UI}$ with $k(u)$. Then,
 - if $y \leq \lambda(u)$, u can use $\kappa(\lambda(u))$ to obtain $\kappa(y)$ by decrypting $E_{\kappa(\lambda(u))}(\kappa(y)) \in Pub_{UI}$;
 - if $y < \lambda(u)$, there exists a path $y = z_0 \leq z_1 \cdots \leq z_n = \lambda(u)$, $n \geq 1$, so that u can successively derive keys $\kappa(z_{n-1}), \dots, \kappa(z_0) = \kappa(y)$.

In other words, the user uses their secret key to derive the key to their particular security level and then traverses the graph, decrypting the key for each node in turn to obtain the desired key, as illustrated in Fig. 2(a).

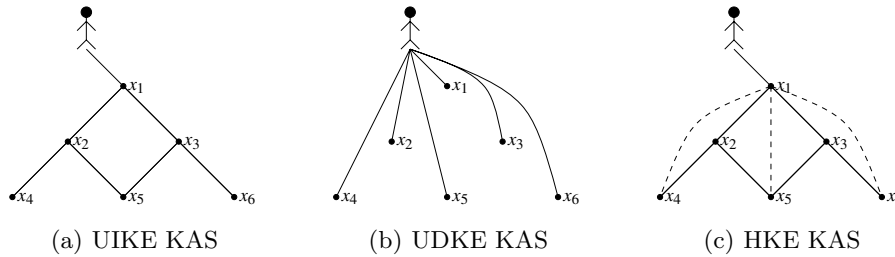


Fig. 2. The directed graphs used in our user-based key assignment schemes: the transitive relationships in the partially ordered set are denoted by broken lines; the user has security label x_1

User-based Direct Key Encrypting KAS (UDKE KAS). This scheme is analogous to a DKE KAS as the user can derive any (permitted) key in a single step. The “graph” of the scheme comprises an edge from each user to each of the security levels for which they are authorised.

- $Pub_{UD} = \{E_{k(u)}(\kappa(y)) : y \leq \lambda(u) : u \in U, y \in L\}$;
- $\kappa(y)$ is obtained by decrypting $E_{k(u)}(\kappa(y)) \in Pub_{UD}$ using $k(u)$.

This scheme allows the user to derive any (permitted) key in a single step, as demonstrated in Fig. 2(b). Although this is a desirable feature, the size of Pub_{UD} is potentially very large.

Hybrid Key Encrypting KAS (HKE KAS). This scheme is a compromise between the previous two, balancing key derivation steps with public storage requirements.

- $Pub_H = \{E_{k(u)}(\kappa(\lambda(u))) : u \in U\} \cup \{E_{\kappa(x)}(\kappa(y)) : y < x : x, y \in L\}$;
- The user first obtains $\kappa(\lambda(u))$ by decrypting $E_{k(u)}(\kappa(\lambda(u))) \in Pub_H$ using $k(u)$. Having derived $\kappa(\lambda(u))$, he can then obtain any $\kappa(y), y < \lambda(u)$, by decrypting $E_{\kappa(\lambda(u))}(\kappa(y)) \in Pub_H$.

Informally, it takes one derivation to “hop” onto the graph of (L, \leq) , from where any (authorised) node can be reached in one step, as shown in Fig. 2(c).

3.2 Performance Evaluation

Key Derivation Time. The UDKE KAS retains the advantage of the DKE KAS in that key derivation still requires a single computation. The UIKE KAS adds one extra step to the original IKE KAS key derivation. The HKE KAS generally takes two steps for any node (the exception being the key for the user’s own security level), making the key derivation time approximately equal to the derivation time for the UDKE KAS. When we compare like with like (that is, UIKE KAS with IKE KAS, etc.) there is no significant difference in key derivation time between the original and the new user-based schemes.

Public Storage Costs. There is no doubt that our schemes require more public storage than the original schemes. If we have n users and l security levels, and assume that $n \gg l$, then the UIKE KAS public storage is $\mathcal{O}(n + e) = \mathcal{O}(n + l^2) = \mathcal{O}(n)$, compared with the original IKE KAS storage of $\mathcal{O}(e)$. The UDKE KAS fares even worse, requiring $\mathcal{O}(nl)$ edges, as opposed to $\mathcal{O}(e^*) = \mathcal{O}(l^2)$ for the original DKE KAS, (a significant difference, given that $n \gg l$). This increase in the size of Pub_{UD} , was the main motivation for the introduction of the HKE KAS; the size of Pub_H is $\mathcal{O}(n + e^*) = \mathcal{O}(n)$.

If we consider an extreme example in which L is a total order containing 100 security levels with 1000 users assigned to each level, the number of edges required for the HKE KAS is proportional to $\frac{1}{2}99 \cdot 100 + 100000 \simeq 10^5$. If we use 256 bit keys/edges, this implies storage space requirements of ~ 3 MB. Whereas this could be significant in some systems, if we consider that the point of our entire scheme is to facilitate access to a remote file server, and would expect a file system of a size vastly exceeding this 3 MB cache to be stored, it becomes clear why we believe the cost of storing Pub_H is negligible in practical terms. The storage requirements for the UIKE KAS are proportional to $99 + 100000 \simeq 10^5$,

so there is almost no advantage to be gained by using the UIKE KAS. The public storage requirements for UDKE KAS are proportional to $1000(100 + \dots + 1) \simeq 5 \times 10^6$ edges, corresponding to storage requirements of ~ 150 MB. Even by modern standards, storing this amount of public information, even on a dedicated file server, may not be considered unimportant.

The increase in public storage for UDKE KAS may seem important when we consider the relative sizes of the public information for each scheme: however, as we have explained in Sect. 2.2, the relevance of this performance parameter is doubtful given the applications to which we expect such schemes to be applied. Nevertheless, although we believe that the size of Pub_{UD} is likely to be acceptable for most applications, we would generally recommend the HKE KAS, with its vastly reduced storage requirements achieved at the cost of merely an extra key derivation step.

Administrative Complexity. Whether through key compromise, user revocation or routine security measures, it will inevitably be necessary to update the keys used to encrypt the data on the file server. Of these, the most complex procedure from an administrative point of view is handling the revocation of a user u_R with a security label $\lambda(u_R) = x$, say. We write $\uparrow x$, to denote the set $\{y \in L : x \leq y\}$, we write $\downarrow x$, to denote the set $\{y \in L : x \geq y\}$, and we write $\kappa(\downarrow x)$ to denote $\{\kappa(y) : y \leq x\}$. Since we must assume that u_R has cached a copy of all the keys to which he had been previously entitled, we must update all $\kappa(\downarrow x)$ (and re-encrypt all affected objects).

In a standard IKE KAS or DKE KAS, this has the effect of changing all the edges incident to any of the affected nodes, meaning changes to the public information. Far more importantly, from a practical perspective, the private key for any $u \in U$ such that $\lambda(u) \leq x$ has to be updated, thereby requiring key redistribution.

If we now consider our schemes, the keys $\kappa(\downarrow x)$ still have to be updated, the data objects re-encrypted and the relevant edges updated (with precisely the same users affected) but, crucially, no key (re)distribution is required. The administrator simply doesn't update (or deletes) the entries $E_{k(u_R)}(\kappa(\downarrow x)) \in Pub_{UD}$ in a UDKE KAS or $E_{k(u_R)}(\kappa(x)) \in Pub_{UI}$ (respectively Pub_H) in a UIKE KAS (HKE KAS) meaning that u_R can no longer derive the new keys and the revocation is accomplished.

3.3 Discussion

In all our schemes, the burden of administration has shifted from the difficult task of redistributing new private keys to geographically distributed users in a secure, confidential manner, to the relatively simple task of updating encrypted information held on a public server. It is this crucial point that differentiates our schemes from all previous KASs in the literature. Although our schemes are based on existing generic constructions, and our ideas can probably be used to improve other schemes, we feel that our contribution is to change the way KASs are generally thought about by researchers and system designers; we anticipate that our ideas are likely to have the most impact on the implementation of KASs.

The only remaining concern is how to ensure that at least the authorised users have the necessary public information when they need it. We feel that a *pull* model, whereby users retrieve their data (edges) as and when they require, is particularly suitable, bearing in mind the system under consideration (users requesting objects), rather than a *push* model, where the central administrator seeks to actively broadcast any data changes to users who may not be online. In fact, the edges could be stored on the resource server itself, with the file system calls modified to return the object and the necessary edge(s), reducing overheads still further.

4 Is HKE KAS the Best Two-Step Scheme?

A number of attempts have been made to find a trade-off between the number of key derivation steps required and the amount of public storage, see [5] for example. The general approach is to construct a new graph $G = (L, R)$, where $R \subseteq L \times L$ and for all $x, y \in L$ such that $y \leq x$, there is a path no longer than p (for some fixed integer p) between x and y in G . Then the public information is defined to be $\{E_{\kappa(x)}(\kappa(y)) : (x, y) \in R\}$.

In this section we consider applying this kind of approach to our user-based KASs. In particular, we investigate whether there exists a two-step KAS with smaller public storage requirements than the HKE KAS. In the HKE KAS, the amount of public storage is proportional to the sum of the “internal” edges arising from the DKE KAS used for L and the “external” edges arising from the edges between each user and his security level. As we have already noted, the amount of public storage is therefore $\mathcal{O}(l^2 + n)$, where $n = |U|$.

What we now try to do is use analogous methods to reduce the number of internal edges at the expense of a small increase in the number of external edges. Suppose, then, that we can partition (L, \leq) into L_1, \dots, L_m such that each L_i has a unique maximal element (with respect to the original partial order). We call these maximal elements *anchor points*, and denote the set of anchor points by $A \subseteq L$.

We can define a DKE KAS for each sub-poset (L_i, \leq) with public information Pub_i . Finally, we can construct a user-centric KEKAS with the following properties:

- the public information contains $\bigcup_{i=1}^m Pub_i$;
- for each user u there is an element of public information for the pair $(u, \lambda(u))$ and for every pair (u, a) such that $a \in A$ and $a \leq \lambda(u)$.

Then a user can get to any anchor point $a_i \leq \lambda(u)$ in one hop and to any other node in L_i in a further hop. Hence, a user can derive the key for any $y \leq \lambda(u)$ in two steps. Writing l_i for $|L_i|$, the public storage required for each scheme is

$$\mathcal{O}\left(\sum_{i=1}^m l_i^2 + \sum_{u \in U} |\downarrow(\lambda(u)) \cap A|\right).$$

A formal comparison of the storage requirements for the standard HKE KAS and the modified scheme proposed above is beyond the scope of this paper. However, the following provides some justification for believing that in most practical instances, the modified scheme is unlikely to be better than the HKE KAS.

In the general case, we note that the reduction in the number of internal edges is fixed (assuming that L and L_1, \dots, L_m are fixed). In any “optimised” two-step solution, adding a single user must add more than one external edge (on average) in order to allow the user to reach every relevant anchor point. While the HKE KAS may well have more internal edges for a small number of users, adding a user only ever requires a single external edge to be added. It is clear then, that in a realistic application, where users will significantly outnumber security levels, so that the total number of edges in the scheme is dominated by the number of external edges, the HKE KAS will be an optimal solution. We include a simple example in Appendix A to demonstrate this.

5 Related work

5.1 Existing KASs

We refer the interested reader to a recent paper for a comprehensive survey of existing KASs in the literature [2], and note that none of these schemes eliminate private key redistribution. Atallah et al. touch upon the idea of eliminating key redistribution [5, Section 5], noting that this could be achieved in principle, but choose to focus their research entirely on other extensions such as trying to optimise the number of edges required (which we discuss in the next section).

5.2 Optimised KASs

In our evaluation of KASs (Sect. 2.2) we only considered the DKE KAS and IKE KAS. We can view these schemes as representing the extreme points on a continuum of possible key encrypting KASs (KEKASs). The IKE KAS represents the KEKAS with the minimum possible storage requirements, at the expense of maximising the average key derivation time. The DKE KAS represents the other end of the scale, with a single-step key derivation achieved at the expense of maximising the public storage requirements.

Recent work by Atallah et al. [7] has developed methods to construct optimal p -step KEKASs (as described briefly in Sect. 4). Whilst their work is intriguing from an abstract mathematical point of view, we believe it has only limited applicability in terms of finding an “optimal” KEKAS (OKE KAS).

In order to develop an OKE KAS, it is first necessary to find a meaningful way to compare the various possibilities. Unfortunately, it is not at all clear how to compare, say, a four-step solution with its particular storage requirements to, say, a three-step solution which has greater storage needs.

More importantly, as we have explained, the trade-off between key derivation time and public storage is not very important in practical terms. In any given

organisation, the actual gains made by using an OKE KAS over either a IKE KAS or a DKE KAS, will be negligible, since neither of these factors are likely to be of any practical significance.

Finally, in almost all situations, system designers are guided by the principle of trying to maximise the runtime performance. Slightly counter-intuitively, the greater the number of edges the organisation stores publicly, the smaller the number of edges each user has to retrieve. The overhead is shifted from a runtime performance hit to an off-line computation by the administrator. Thus, even if a suitable objective function could be devised and a cost-benefit analysis conducted, the DKE KAS is still overwhelmingly likely to be selected as the best overall option, as it requires the user to download the minimum possible number of edges and perform just a single computation at runtime in order to obtain any (permitted) key.

In the previous section, we introduced our new, user-based schemes and pointed out that we favoured the two-step HKE KAS over the single-step UDKE KAS, which would clearly be faster at runtime. The reason for this discrepancy is straightforward. The public storage requirements in the user-based schemes scale with $n = |U|$, as opposed to the original schemes which scale with $l = |L|$. Since we expect that in most organisations $n \gg l$, the size of Pub_{UD} can become sufficiently large for it to be considered non-trivial (as explained in Sect. 3.2). Once this happens, it is necessary to consider ways of reducing the requirements; the HKE KAS provides a natural way to construct a two-step solution which significantly reduces the public storage requirements.

6 Conclusion and Future Research

We have introduced three new KASs, the UIKE KAS, the UDKE KAS and the HKE KAS. In this work, we have developed a unique, pragmatic approach to evaluating KASs, ignoring the negligible public storage requirements and key derivation times, focusing instead on the issue of private key updates and the resulting requirement for key redistribution. We believe this single issue to be overwhelmingly the most important factor in any KAS analysis.

Our schemes address this point by eliminating all private key updates, transforming the problem of key redistribution following key updates into a simple data update process. Moreover, our ideas can be implemented on top of any existing provably secure KAS, thus automatically inheriting the same security guarantees.

There is an interesting parallel we can draw between our ideas and the concept of a CPFS. In all remote file systems it is necessary to protect sensitive data by encrypting it, at least for transmission purposes. The CPFS approach merges the cryptography layer into the file system itself. Nothing is fundamentally changed by moving the protection mechanism, except that our approach to security shifts completely into the area of KASs, making sure that only authorised users have the necessary keys. By using this approach, we need no longer worry about the consequences of the server being compromised or the security

of the communication channel between the client and the server, since the *trusted computing base* now comprises only the client machine.

In existing KASs, a secure, private key redistribution mechanism is assumed. This would almost certainly be in the form of a shared secret between each user and the scheme administrator, who would then send the new key to the user encrypted with the shared secret to maintain confidentiality.² Our contribution is to merge this key distribution mechanism into the KAS itself, by publishing the encrypted keys along with the rest of the public KAS information. Architecturally, little has fundamentally changed, but by using this approach, we reduce the overall complexity of the scheme, since all administrative tasks can be handled with simple updates to a public server.

In future work, we would like to implement our schemes, investigating whether our assumptions about performance characteristics are realistic. It is also worth investigating whether the HKE KAS is indeed an optimal two-step solution in realistic applications and the conditions under which it becomes optimal for arbitrary hierarchies and user distributions.

Another significant obstacle barring the widespread implementation of CPFSSs, is the problem of mass re-encryption of data following updates to the encryption keys. Certain approaches have been proposed, including delaying the re-encryption, known as the *lazy update* approach [8]. In future work we would like to consider ways to avoid this problem entirely.

References

1. Fu, K., Kamara, S., Kohno, Y.: Key regression: Enabling efficient key distribution for secure distributed storage. In: Proceedings of the Network and Distributed System Security (NDSS 2006) (2006)
2. Crampton, J., Martin, K., Wild, P.: On key assignment for hierarchical access control. In: Proceedings of 19th Computer Security Foundations Workshop, pp. 98–111 (2006)
3. Denning, D.: A lattice model of secure information flow. *Communications of the ACM* 19(5), 236–243 (1976)
4. Davey, B., Priestley, H.: *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, United Kingdom (1990)
5. Atallah, M.J., Frikken, K.B., Blanton, M.: Dynamic and efficient key management for access hierarchies. In: Proceedings of 12th ACM Conference on Computer and Communications Security, pp. 190–202. ACM Press, New York (2005)
6. US Department of Defense: Trusted computer system evaluation criteria. Technical Report 5200.28-STD, DoD (1985)
7. Atallah, M.J., Blanton, M., Frikken, K.B.: Key management for non-tree access hierarchies. In: Proceedings of the 11th ACM Symposium on Access Control Models and Technologies, pp. 11–18. ACM Press, New York (2006)

² Of course this could also be achieved in other ways such as using asymmetric cryptographic techniques. Our point is to demonstrate the advantages of eliminating the requirement to maintain a separate private key distribution mechanism, regardless of how this is implemented.

8. Backes, M., Cachin, C., Oprea, A.: Secure key-updating for lazy revocation. In: Proceedings of 11th European Symposium on Research in Computer Security, pp. 327–346 (2006)

Appendix A: A Simple Example

Let us now apply the analysis of Sect. 4 to a simple example, in which L is a total order and equal numbers of users are assigned to each security label. Then we can divide L into k sub-chains of length $m = l/k$. The DKE KAS for each sub-chain requires $\frac{1}{2}m(m-1)$ edges. Hence there are $\frac{k}{2}m(m-1) = \frac{1}{2}l(m-1)$ internal edges in the new scheme. The number of internal edges for the standard HKE KAS is $\frac{1}{2}l(l-1)$. Hence the number of internal edges is reduced by $\frac{1}{2}l(l-1 - (m-1)) = \frac{1}{2}l(l-m)$.

We now consider what extra external edges are needed in the new scheme. Users assigned to a label in the highest subchain have k edges (one for their security label and $k-1$ for each of the other anchor points). Similarly, users assigned to a label in the next highest subchain have $k-1$ edges. Hence, in total, we require n edges to connect each user to his respective security level and

$$\frac{n}{l} \sum_{i=1}^{k-1} i = \frac{n}{2l} k(k-1) = \frac{n}{2l} \frac{l}{m} \left(\frac{l}{m} - 1 \right) = \frac{n}{m^2} (l-m)$$

additional edges to connect users to relevant anchor points. The proposed scheme improves on the basic HKE KAS only if the number of additional external edges is less than the reduction in internal edges. Hence, we need to consider whether $\frac{n}{m^2}(l-m) < \frac{1}{2}l(l-m)$. We may assume that $l > m$ (otherwise $k = 1$ and the modified scheme is equivalent to the original HKE KAS). Hence, we have that the modified scheme offers an improvement if $lm^2 > n$. For most practical schemes, $n \gg l$ and this inequality is unlikely to hold. Now, the largest m can be is $l/2$, so there is only an improvement if $n < l^3/4$. If $l = 10$, for example, we only require 250 users for the optimal choice to be $k = 1$ (which represents the basic HKE KAS).