Cryptographic Access Control in the PESTO File System*

Margrete Allern Brose margrete@cs.uit.no
Ben Johnsen ben@math.uit.no
Tage Stabell-Kulø task@ifi.uit.no

Abstract

In the PESTO file system, the complete file update history is kept as a tree of file versions. It will often be desirable to give access to subsets of the file versions history without having to grant access to the entire tree. Keys on Trees is a scheme for assigning attributes to every tree node in such a way that knowledge of the attribute of one node makes it possible to calculate the attribute of every subnode, but impossible to calculate the attribute of any other nodes. This paper describes Keys on Trees and how it is intended applied in PESTO.

1 Introduction

Before we get into the details regarding cryptographic access control, we need to give an introduction to the context, namely the PESTO file system [1]. PESTO is a distributed, version based file system that provides its users with highly available, secure and sharable storage. One of the main goals of PESTO is to maximize progress at disconnection, and this is achieved through decentralization of administrative control.

Files in PESTO are stored as sets of file versions. Updates are non-destructive in that no information is lost by an update to a file; all its history is kept. The storage model for file versions is Write-Once Read-Many-Times. The set of file versions is partially ordered and structured as a tree, where each version has a link to its parent version (Figure 1 (a)). Each file version is encrypted with a separate key. A file is associated with a file key and each version with a version key (Figure 1 (b)). File versions are the units of storage, distribution and access, and access control is performed by limiting the access to the version keys.

* This work has been supported in part by the NFR funded project No. IKTSOS 158569/431 PENNE

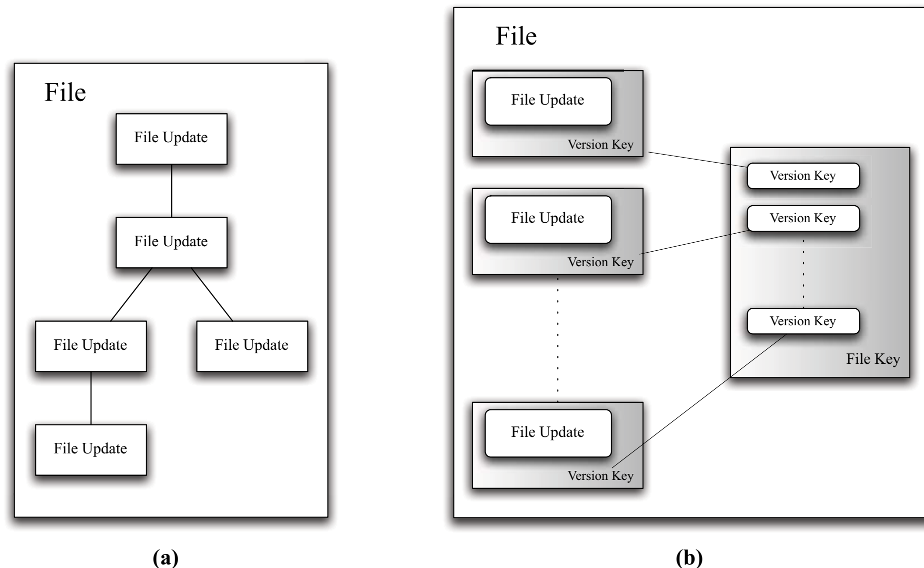

Figure 1: (a) Each file update has a link to its parent update. The initial empty version has no parent update. (b) Each file update is encrypted with a separate version key, and the version keys are in turn encrypted with the file key.

A user that wishes to make an update or to read a file needs to send a request for this to the owner of the file, which in turn may grant this access. In order to be able to read a file, the reader needs the version key for the relevant version. Authorized writes require a fresh version key, that is, a key that has not previously been used, and a copy of this key encrypted with the file key of that file. For more details, see [1].

In the current system a user has to make requests for the keys that he needs to make his intended updates, i.e. one key per update. Knowing how many in advance may be hard and, thus, the user may have to request more keys later. If connected, this is not a problem, but if he is going to be offline for a while a different solution to key management is desirable.

It may also be desirable to be able to grant access only to subsets of the version history tree. For example, a patient would like to give his doctor access to his medical record. If he moves, however, he would not like his old doctor to read his future record. This is hard to implement in PESTO.

Keys on Trees [2] supports resolving these problems, and provides a way of generating keys in a consistent manner, while at the same time conforming to the PESTO goal of maximum progress at disconnection.

The remainder of this paper is structured as follows. The next section will give a description of Keys on Trees. Section 3 will then provide some examples on how we intend to use this in PESTO. Some related work and discussion then follows.

2 Keys on Trees

2.1 Notation

Since any tree may be thought of as a subtree of the complete binary tree, T , we will only study the key structures for this in the following. T is associated with all finite binary strings $\{0,1\}^*$. The empty string is denoted $\varnothing$ and $a * b$ denotes the concatenation of the strings $a, b \in \{0,1\}^*$. This organizes the strings into a semigroup denoted *String* .

Every string $a \in \{0,1\}^*$ (thought of as an address) defines the subtree $a * T$ of all strings with prefix a . This gives the set $subT$ of subtrees $\{a * T\}_{a \in T}$. We identify T with the subtree $\varnothing * T$.

2.2 Finding a function

Figure 2 illustrates the complete binary tree, T ; $x \in X$ is the attribute/key assigned to the root.

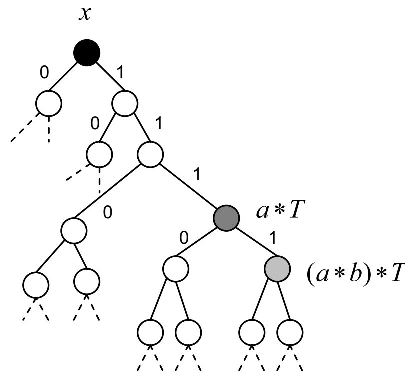

Figure 2: Binary tree

For calculating the attribute of a node we define a flow on the tree which assigns the value $V(y, a * T, b)$ to a descendant of the subtree $a * T$. This can be expressed by defining the functions

$$V(x, a * T, i) \text{ for } i \in \{0,1\}$$

i indicating which descendant of $a * T$; e.g. $i = 1$ for calculating the value for $(a * b) * T$, which is a descendant of $a * T$ (Figure 2).

Since we want to be able to calculate the flow values for all descendants but make it impossible to calculate values for any ascendants, the idea is to use one-way functions to express the flow functions. We first consider simple one-way functions obtained by exponentiations in a group X :

$$V(x, a*T, b) = x^{2^{rb}3^{s\underline{b}}}$$

where $b \in \{0,1\}$ and $\underline{b} = (1+b) \bmod 2$. The integers r and s are calculated by a weight function

$$w(x, a*T, b) = (r, s)$$

The flow values in the tree will then be repeated compositions of x^2 and x^3 .

Though formally any function

$$w : X \times subT \times \{0,1\} \rightarrow Z \times Z$$

can be used as a weight function, care must be taken when choosing the function. We will now provide an example to illustrate why having a one-way function is not enough to achieve what we want.

2.3 *An attack on the flow*

It is tempting to use an X - independent, additive function to obtain a nice formula for the flow function. Let $l(a)$ denote the length of a string a , $|a|$ the number of 0's in a and $(a)_2$ the number with binary digits a , the least significant digit to the left. A few example functions are:

$$w_1(a*T, b) = 2^{l(a)}(b)_2 \quad (1)$$
$$w_2(a*T, b) = |b| \quad (2)$$

Neither one of these functions is injective, but injectivity can be obtained through the combination:

$$w(a*T, b) = (w_1, w_2) = (2^{l(a)}(b)_2, |b|)$$

We thus get the flow

$$V(x, a*T, b) = x^{2^{w_1}3^{w_2}} = x^{2^{l(a)}(b)_2 3^{|b|}}$$

and see that

$$x^{2^{w_1(T,b)}3^{w_2(T,b)}} = x^{2^{w_1(T,a)}3^{w_2(T,a)}}$$

requires the exponents to be congruent modulo the order of x in X . The exponents are equal if and only if $a = b$, since $((a)_2, |a|)$ specifies a uniquely.

Example, sideway leakage:

Let $(y, a*T)$ be a given element of the flow just defined. The attack algorithm calculates $2^{(a)_2} \cdot 3^{|a|}$. Let

$$c = (a)_2 + s$$

and

$$d = (c \text{ to the base } 2)$$

where $s \geq 0$.

Let b be any string obtained from d padded right with enough 0's such that

$$|b| = |a| + t \geq \text{the number of 0's in } d$$

for $t \geq 0$. The least possible value of t is determined by $|d|, |a|$ and s. Let $z = y^{2^s 3^t} \bmod n$. The attack algorithm calculates $(z, b*T)$. If there exists an x such that:

$$flow(x, T, a) = (y, a*T) \qquad (1)$$

then

$$x^{2^{(2)_2 3^{|a|}}} = y$$

and

$$z = y^{2^s 3^t} = x^{2^{(b)_2 3^{|b|}}}$$

such that

$$flow(x, T, b) = (z, b*T) \qquad (2)$$

The calculated value $(z, b*T)$ is a descendant of $(y, a*T)$ if and only if $s = 2^{l(a)} \cdot s_1$. This would mean that a is a prefix to b. Other values for s break the system. Note that the attacker does not prove that the given value $(y, a*T)$ actually is in the flow, that is, that an x fulfilling (1) exists, only that the attack succeeds if this is the case.

To illustrate; the attacker obtains the element $(x^3, a = (0))$ of the flow on the tree shown in Figure 3:

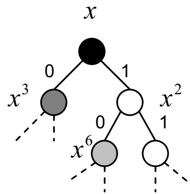

Figure 3

if a different flow value $(z, b*T)$ is to be calculated from the given value, then

$$x^{2^{(b)_2 3^{|b|}}} = x^{2^{(a)_2 3^{|a|} w}}$$

The attacker sets $w = 2^s 3^t$ such that

$$(b)_2 = (a)_2 + s = 0 + s = s$$

$$|b| = |a| + t = 1 + t$$

By choosing $s = 1$ and $t = 0$, the attacker has obtained the flow value $(x^6, b = (10))$, which is not a descendant of $(x^3, a = (0))$, without knowing x.

The success of the attack depends on the possibilities of calculating the weight $w(T, a)$ from the knowledge of a and then the logarithm of y to the unknown base x. The homomorphic properties of the logarithm makes it possible to manipulate the exponents. The main problem, though, is the possibility of finding exponents corresponding to solutions not being descendants of $(y, a * T)$ and then to find matching strings b to these exponents without knowing x.

2.4 Definition of attacks and security

The above example shows that a precise definition of an attack and security is needed. In addition to the requirement of a one-way function, a hypothesis regarding non-calculability restricting the attacking algorithm is needed:

Non-calculation hypothesis

Hypothesis 1: The flow value function is one-way

The flow function in one-way, that is, suppose r is random and that $(r, a * T)$ given to the attacker. If the attacker does not know an ascendant of $(r, a * T)$, he will not succeed in calculating one with more than negligible probability.

Hypothesis 2: The weight function is unsolvable

For a random x no efficient algorithm will find a solution $(c, (R, S))$ of the equation

$$w(x, T, c) = (R, S)$$

without knowing the value of $F(x, T, b)$, where b is a known, proper prefix to c.

The Definition of Attacks

An attack is an efficient, probabilistic algorithm which from a given value $(z, a * T)$calculates a value$(y, b * T)$. The attack is successful if:

- a is not a prefix to b
- If $F(x, T, a) = (z, a * T)$ then $F(x, T, b) = (y, b * T)$
- The equation $F(x, T, b) = (y, b * T)$ can be efficiently verified under the hypothesis $F(x, T, a) = (z, a * T)$

The success rate of the attack is measured by the probability that the calculation shows that $(y, b*T) = F(x,T,b)$ where a is not a prefix to b.

An X - flow is then cryptographically secure if there is no efficient attack algorithm running in acceptable time with an unacceptable success rate.

Theorem: *The probability that an attack restricted by hypothesis 1 and hypothesis 2 succeeds on the flow is negligible.*

Proof:

Let F be the flow with $(y, b*T)$, the value calculated by an attack algorithm A. Suppose $F(x,T,a) = (z, a*T)$ for a value x unknown to the attacker and that the attack is successful. Then for some integers (r,s) a priori unknown to the attacker

$$w(x,T,a) = (r,s) \text{ and } V(x,T,a) = z = x^{2^r 3^s}$$

Since $F(x,T,b) = (y, b*T)$,

$$w(x,T,b) = (R,S) \text{ and } V(x,T,b) = y = x^{2^R 3^S}$$

Since (y,b) is calculated from $(z, a*T)$,

$$y = \left(x^{2^r 3^s}\right)^M$$

for an integer M calculated by the attacker. The string b calculated by the attacker is then a solution of the equation

$$(R,S) = w(x,T,b)$$

for an x such that

$$z = V(x,T,a)$$

and for numbers R,S such that

$$\left(x^{2^R 3^S}\right) = \left(x^{2^r 3^s}\right)^M$$

This implies that an efficient and successful attacker either has found a solution x of the second equation or found a solution of the first without knowing x. According to hypotheses 1 and 2 this implies that when a is not a prefix of b, an event with very small probability has occurred. □

2.5 Unsolvable weight function

The weight function we chose in the example was X-independent, so it seems like a natural solution to make the function deeply dependent on X. We then need to consider

calculations in the group X. The basic requirement of the group is that the functions x^2 and x^3 are one-ways, so we want a group where these functions are one-ways under a reasonable assumption. The group Z_n^*, where $n = p * q$, and p and q are very large primes, seems to be a good choice, based on that the calculation of roots is considered difficult (Details regarding this can be found in [2]). The weight function could then be implemented by adapting collision free hash function with good randomizing properties.

2.6 *Applications*

If $p \equiv q \equiv 3 \bmod 4$, the flow values can be used as seeds to the cryptographically secure BBS-generator. In this way it is possible to assign a cryptographically strong bit sequence $BBS(V(x,T,a))$ as an attribute to the subtree $a * T$ for every string a. From each bit string $BBS(V(x,T,a))$ we can choose bit strings as AES-keys, RSA-parameters p,q,e, ElGamal parameters p,g,h etc. by deterministic algorithms.

3 Applications in PESTO

The system just described fits nicely with the way file versions are organized and handled in PESTO. We will now outline a few ways in which this may be applied in the file system.

3.1 Collaboration on files – Write Access

As previously described, a user that wishes to contribute updates to a certain file needs to request a fresh member key for each of these updates. With Keys on Trees, the creator of a file can generate two keys, x_1 and x_2, from the original/seed key x (Figure 4). He may then continue working on the part of the tree rooted at x_1 while the other key, x_2, can be used to delegate write authority to another user in the system. The other user may then continue making new file versions without having to send a new request each time he needs to make an update to the file, since he can generate additional keys from x_2.

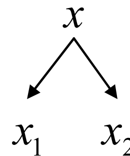

Figure 4: Generate two new keys from the original key

In this specific scenario, only the creator has access to the entire update tree. It may be useful to also hand over x, so that they both can read the other user's updates, but the creator still needs to generate two keys so they have different starting points. Upon merge, the other user only needs to send the encrypted updates to the creator, since the creator of the file will be able to calculate all the keys that are needed to decrypt the file updates. In the same manner, if another writer enters the system at a later stage, he may

be handed the key x_n and write updates to the subtree rooted at the node assigned this key, without access to the rest of the tree.

3.2 *Read Access*

While it may be advantageous to let users that have been given write authority the ability to generate new keys on their own, it is not necessarily the case with reads. We would still want to be able to give someone read access to single versions of the file. This is not a problem with Keys on Trees. Instead of handing out the actual key to someone, a hash of the key could be used to encrypt the file. This way the key need not be revealed to that user, which otherwise could use it to gain access to later versions of that file.

3.3 *Revocation*

How to revoke access in this system will depend on what access rights need to be revoked.

Revoking further update access

This would depend on where in the tree the user that needs to be denied access initially was granted access. If access to the whole tree previously was granted, the whole tree needs to be assigned new attributes. It would be a little different if access to only a smaller part of the tree needs to be revoked. If there are leaf nodes available, the subtree could be moved here and then only the attributes for these nodes need to be recalculated (Figure 5). If the subtree cannot be moved, then all attributes are recalculated. In all cases the newly encrypted versions are distributed, (like in the current system, without Keys on Trees).

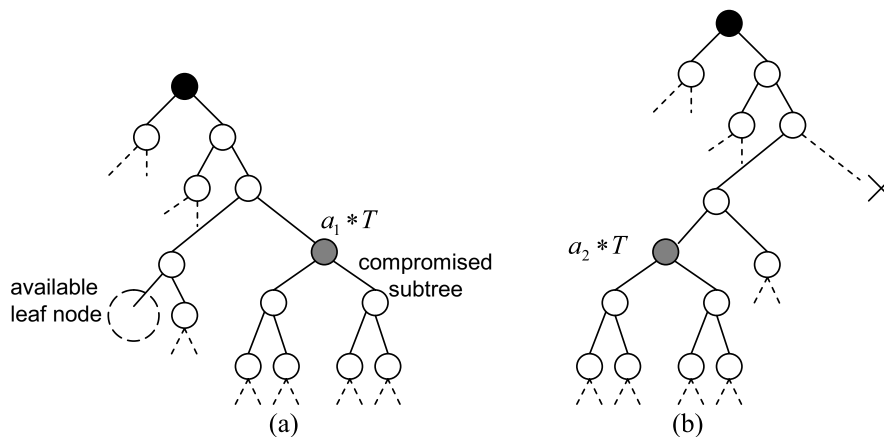

Figure 5: There is a free leaf node (a), so the subtree that is rooted at a_1 can be moved here, with the new address a_2 (b). a_1 is no longer a valid address in this tree.

Revoking read access

First of all, if someone has read something, the damage is already done. Furthermore, if the access was only for a specific version, and it is sufficient that this user does not have access to future versions of the file, then nothing needs to be done. Otherwise, the file version tree could have the keys recalculated in the same manner as when revoking write access.

3.4 Other

This grouping of related updates is useful in a couple of other ways, for instance, if the creator of a file decides that he wants to ignore the updates made by another user, for whatever reason, having these updates residing in the same subtree makes not using them easier to handle, since one can just leave out this part of the tree. Keys on Trees will also make it possible to partially reveal the events that have taken place for recovery purposes and the gathering of evidence.

4 Related Work and Discussion

Hierarchical access structures are found in many different contexts; for instance organizations and distributed applications such as multimedia applications and databases. Previous work has been done on access control in such contexts both on tree structures [4][7] and general partially ordered sets (posets) [3][5][6][8]. The setting in these cases is that users and resources are grouped into different security classes, so the hierarchy consists of these classes. Sandhu [4] studies users and information items that are grouped into security classes in a tree structure. The keys in the subtrees are generated by applying one-way functions iteratively. The different classes have different one-way functions. Which function to apply, is based on name of the child. The names and function family are all publicly known. General posets are dealt with in e.g. [5], where a few different access control schemes for distributed environments are described. Among others a one-way function-based keying scheme, which is a generalisation of Lin's scheme [6]. Each node selects its own key independently, but it is possible to deduce keys of descendants. Sideway leakages are dealt with through the use of so-called sibling intractable function families in [10].

An advantage of the Keys on Trees scheme is that adding nodes, i.e. making new versions, does not affect nodes higher up in the hierarchy. In the scheme suggested in [3], e.g., the keys of the ancestor nodes need to be recalculated every time a child node is added. One of the things that we need to evaluate, however, is the cost of calculating the keys in a tree when access is needed, since one potentially will need to traverse the whole path from the root to the node which contains the desired version. This will depend on the depth of such a tree. Zheng et al. [10] suggest one scheme for being able to calculate a descendant at a much lower level directly, which involves creating links between a node and each one of its ancestor nodes.

5 Conclusion/Future Work

In this paper we have introduced Keys on Trees, and given an overview of how this system can be used in the PESTO file system. We believe this approach may prove useful. In comparison to today's implementation, it is clear that the Keys on Trees solution provides improvement. When it comes to delegation of write authority, for instance, the scheme both reduces the number of messages exchanged, and makes it possible for a user to make as many updates as he may desire, also while not being connected, since each user can generate its own keys for the updates. We do, however, need to look into the details further. The next step now will be making an implementation that incorporates this scheme, and evaluate how this works in practice.

The PENNE project is a joint project between the Departments of Mathematics and Computer Science at the University of Tromsø. Two complementary efforts are combined: One in secure distributed systems research and one in cryptography. The overall goal of PENNE is to consolidate and enlarge the information security activity at the University of Tromsø. PENNE builds on previous work in the PASTA project, and later in the PESTO project at the Department of Computer Science. Keys on trees is one of the key issues currently being investigated using PESTO as a research vehicle.

References

[1] Dillema, Feike W. Disconnected Operation in the PESTO Storage System. 2004

[2] Johnsen, Ben. Keys on Trees

[3] Akl, Selim G and Taylor, Peter D. Cryptographic Solution to a Problem of Access Control in a Hierarchy. ACM Transactions on Computer Systems, Vol. 1, No. 3, August 1983, pp. 239-248

[4] Sandhu, Ravinderpal S. Cryptographic Implementation of a Tree Hierarchy for Access Control. Information Processing Letters 27 (1988) 95-98

[5] Birget, Jean-Camille et al. Hierarchy-Based Access Control in Distributed Environments. Proceedings of International Conference on Communication--ICC 2001, June 2001, Helsinki, Finland, pp. 229-233.

[6] Lin, Chu-Hsing. Dynamic Key Management Schemes for Access Control in a Hierarchy. Computer Communications 20 (1997) 1381-1385

[7] Sun, Yan and Liu, K. J. Ray. Scalable Hierarchical Access Control in Secure Group Communications. IEEE INFOCOM 2004

[8] De Santis, Alfredo et al. Cryptographic Key Assignment Schemes for any Access Control Policy. Information Processing Letters 92 (2004) 199-205

[9] Ray, Indrakshi et al. A Cryptographic Solution to Implement Access Control in a Hierarchy and More. SACMAT'02

[10] Zheng, Yuliang et al. New Solutions to the Problem of Access Control in a Hierarchy. 1993