

A Diffie-Hellman based Key Management Scheme for Hierarchical Access Control

Anna Zych, Jeroen Doumen, Pieter Hartel, and Willem Jonker
{Anna.Zych,Jeroen.Doumen,Pieter.Hartel,Willem.Jonker}@utwente.nl

University of Twente, Enschede

Abstract. All organizations share data in a carefully managed fashion by using access control mechanisms. We focus on enforcing access control by encrypting the data and managing the encryption keys. We make the realistic assumption that the structure of any organization is a hierarchy of security classes. Data from a certain security class can only be accessed by another security class, if it is higher or at the same level in the hierarchy. Otherwise access is denied. Our solution is based on the Diffie-Hellman key exchange protocol. We show, that the theoretical worst case performance of our solution is slightly better than that of all other existing solutions. We also show, that our performance in practical cases is linear in the size of the hierarchy, whereas the best results from the literature are quadratic.

Keywords. Access control, hierarchical key management.

1 Introduction

It has been estimated that the world currently produces five Exabytes (i.e. 5×10^{18} bytes) of digital content annually [13]. We may assume that (1) most of this information is stored on a large number of servers connected to the Internet, and (2) for business and privacy reasons, each item of content should be accessible only to a carefully controlled group of people. Classical access control mechanisms assume that the server can be trusted to enforce access control policies. While this is a realistic assumption for relatively small databases, with (1) the growing amounts of data being produced, and (2) the trend to outsource both data and processes, trusting the server becomes less realistic.

Fortunately, data can be stored securely on an untrusted server when the data is encrypted. Access control then boils down to managing the encryption keys. The server processes only encrypted data, and the user performs the en/decryption; the server never even sees the plaintext, or the keys. In most applications, access control is hierarchical; for example the CEO of a company has (at least in principle) access to all company information, the CFO only to all financial information etc. Therefore to provide access control over encrypted data we have to solve the problem of *hierarchical key management*. The naive solution would give the CEO the keys to each item of information; the CFO would get the subset of all keys that decrypt the financial records etc. Given the large numbers of keys involved, this is unworkable, so many sophisticated schemes have emerged, each basically calculating keys deeper in the hierarchy. There are

methods based on modular exponentiation top down [1, 9] or bottom up [8, 6]. Some of the approaches derive from the idea of sealed keys [12, 5, 3], others avoid public key cryptography by attaching public parameters to the edges of the hierarchy, instead of to the classes [7, 15]. Assuming that there are n nodes in the access control hierarchy, the spatial efficiency of an access control scheme is determined by its private and public parameters, which are de facto private and public keys assigned to (the nodes of) the hierarchy. To be more precise, the private/public space complexity of a key generation scheme, is the number of all private/public keys assigned to (the nodes of) the hierarchy. The computational complexity is the time needed to perform the key derivation step.

Our contribution is twofold. Firstly we develop a new hierarchical key management scheme for access control based on Diffie-Hellman key exchange, which has a worst case public space complexity of $O(\frac{n^2 \log \log n}{\log n})$, whereas all other proposals have at least $O(n^2)$. Secondly, our work is the first to also provide a practical evaluation of the proposed access control scheme. In particular, we study four realistic application scenarios, showing that in each case our scheme has a practical space and computational complexity of $O(n)$.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the scheme for a certain class of hierarchies. Section 4 generalizes the scheme to arbitrary hierarchies. Section 5 presents application scenarios and tests for our scheme. We summarize the contributions of our paper in Section 6.

2 Related work

In almost all schemes for access control in a hierarchy, there is a relationship between the key assigned to a node and the keys assigned to its children. The difference between proposed methods lies mostly in the different cryptographic solutions used for the key generation. We outline a few important approaches in this area below.

One of the early solutions to the hierarchical access control problem by Akl and Taylor [1] is based on modular exponentiation top-down. The authors choose the exponents in such a way, that the key of a child node can be derived from the key of its parent. MacKinnon et al. [9] optimize the scheme by choosing less space consuming exponents. However, even for optimized parameters, the public space required remains exponential in the number of nodes.

Hwang and Yang [6] present a scheme that improves the scheme of Akl and Taylor. This scheme is also based on modular exponentiation, but the novelty is the bottom-up approach. The space complexity remains exponential.

Lin, Hwang and Chang [8] also propose a scheme based on modular exponentiation, bottom-up. This scheme solves the problem of exponential public space requirement. The public space complexity becomes $O(n^2 \log n)$.

Sandhu [12] proposes a key generation scheme for a tree hierarchy. The solution is based on using different one-way functions to generate the key for each child node in the hierarchy. The one-way function is selected based on the name

of the child. When a new child is added, the keys for the ancestors do not have to be recomputed. Hwang [5] proposes a solution for the general poset. The advantage of this approach is its flexibility. The space required for public parameters is quadratic.

Ray et al. [11] present a scheme, for which the key derivation time is constant. This is achieved using modular exponentiation with the same power for all classes, but each class uses a different derivation modulus. The scheme is interesting from a theoretical point of view. However, the efficiency problem is pushed to private space, which becomes exponential. This means, that users themselves must store huge keys.

Das et al. [3] propose a scheme based on polynomial interpolation. A secret key derivation phase requires the user to interpolate a polynomial associated with a node. The space complexity is also quadratic.

Zhong [15] and Lin [7] propose schemes not based on public key cryptography. The public parameters are associated with edges of the graph instead of nodes. Therefore the public space required is quadratic. Avoiding public key cryptography results in computational efficiency of the key derivation phase.

In Table 1 we summarize the performance of existing approaches. Apart from the Ray et al. [11] scheme, which is unacceptable because of its private space complexity, all the solutions require at least $O(e) = O(n^2)$ public space, where e is the number of edges in the hierarchy.

Table 1. Comparison of some existing methods, $e = O(n^2)$ is the number of edges, and $h = O(n)$ is the height of the hierarchy

Scheme	Public Space	Private Space	Comput. Compl.
Akl, Taylor	$O((n \log n)^n \times n)$	$O(n)$	$O(n)$
Hwang, Yang	$O((n \log n)^n \times n)$	$O(n)$	$O(n)$
Lin, Hwang, Chang	$O(n^2 \log n)$	$O(n)$	$O(n)$
Sandhu, Hwang	$O(e) = O(n^2)$	$O(n)$	$O(h)$
Ray	0	$O((n \log n)^n \times n)$	$O(1)$
Das	$O(e) = O(n^2)$	$O(n)$	$O(h)$
Lin, Zhong	$O(e) = O(n^2)$	$O(n)$	$O(h)$

3 Scheme Description

For simplicity we assume the existence of a Central Authority (CA). The responsibility of the CA is to generate and distribute keys. Initially, the CA assigns to each user class two keys: a private key, and the accompanying public key that will be available to all user classes. The system could function without a CA, requiring a distributed assignment of public keys and authenticated channels. The public keys are used to derive the secret keys of other user classes one is allowed to access. The secret key is also needed for this, as well as for allowing access to the encryption/decryption key for the documents.

We first present our scheme for a specific class of posets. Then we describe how to generalize the scheme to arbitrary posets. Appendix A summarizes basic definitions from order theory, used further in this section.

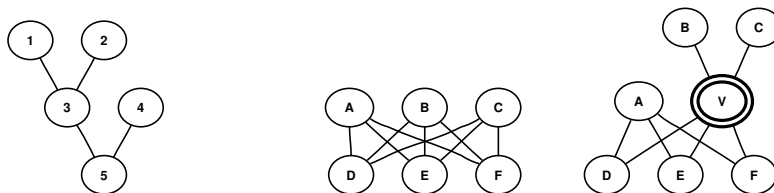


Fig. 1. Example of a cover various V-poset P used as example (left) and a non V-poset (middle) embedded into V-poset (right)

From now on we represent a class hierarchy as a poset, whose elements represent classes of users. All the algorithms presented further operate on a poset, that is the algorithms are given as input a set of elements, and the relations between the elements. We operate on an abstract structure *Poset* P representing poset $P = (H, <_H)$. Structure *Poset* P contains an abstract structure *Node* x for every element $x \in H$ of the poset P , and an abstract structure *Edge* (x, y) for every relation $x <_H y$ in P . We use the following standard operations on *Poset* P :

- *LeftParent*(*Poset* P , *Node* x) - returns the first node that covers x .
- *RightParent*(*Poset* P , *Node* x) - returns the last node that covers x .
- *InsertEdge*(*Poset* P , *Node* x , *Node* y) - adds a relation $x <_H y$ to P .
- *CreateNode*(*Poset* P) - creates a new node in P and returns it.

It is worth to note here, that while the algorithms operate on posets, for the sake of simplicity the illustrations show the Hasse diagrams of these posets.

The algorithms use an external *FIFOqueue* Q , with standard operations *Push*(*FIFOqueue* Q , *Node* x), *Pop*(*FIFOqueue* Q) and *Size*(*FIFOqueue* Q).

3.1 Scheme for cover various V-posets

We first present our scheme for cover various V-posets. A cover various V-poset is a poset in which each non-maximal node has exactly two parents, and where no two nodes have the same covering set. Later we will generalize it to arbitrary posets. The V-structure allows us to use the two party Diffie-Hellman key exchange protocol [4]. Each non-maximal node has exactly two parents, which can establish the child key using the DH protocol. This requires knowledge of the secret key of one of the parents, and the public key of the other parent. Therefore the security of our scheme is directly dependent on the security of DH. The fact that the input poset is cover various ensures that no two nodes obtain the same key.

In this section we present algorithms to assign and derive the keys.

Key Assignment. Algorithm *KeyAssignment*(*Poset* P) operates on a cover various V-poset $P = (H, <_H)$, representing the class hierarchy. It assigns private and public keys to the nodes of P . This algorithm is run once by the CA at the beginning of the systems lifetime. Therefore the computational complexity of

this step is not a significant factor. The space required for the generated keys is the crucial issue and should be minimized.

After completing this algorithm each *Node* x is labeled with two keys: S_x - its private key, and P_x - its public key.

The *KeyAssignment(Poset P)* algorithm

```

 $p := \text{LargePrime}; g := \text{random}([2..p - 1]);$ 
for each node  $x \in \text{Max}(P)$  do
   $S_x := \text{random}([2..p - 1]);$  //secret key
   $P_x := g^{S_x} \text{ mod } p;$  od; //public key
for each node  $x \in H \setminus \text{Max}(P)$  do
   $S_l := \text{SecretKey}(\text{LeftParent}(P, x));$  //only available to the CA
   $S_r := \text{SecretKey}(\text{RightParent}(P, x));$  //only available to the CA
   $S_x := g^{S_l S_r} \text{ mod } p;$ 
   $P_x := g^{S_x} \text{ mod } p;$  od;
```

Key Derivation. To derive a key of a descendant node, a node needs its own secret key, as well as the public keys of the “other” parents in the path to the target node. It will recursively look for a path to the target, and derive child keys by calculating $S_{\text{child}} = P_{\text{otherparent}}^{S_{\text{parent}}}$ recursively. This *KeyDerivation* algorithm is run by the users whenever they need to derive keys. Therefore the computational complexity of this procedure is an important factor of the scheme efficiency. To be precise, the complexity is $O(h)$, where h is the height of the input poset.

Example. For the hierarchy P presented in Figure 1 (left), the procedure *KeyAssignment(P)* generates the following keys.

Node	1	2	3	4	5
Secret key	random S_1	random S_2	$S_3 = g^{S_1 S_2}$	random S_4	$S_5 = g^{S_3 S_4}$
Public key	$P_1 = g^{S_1}$	$P_2 = g^{S_2}$	$P_3 = g^{S_3}$	$P_4 = g^{S_4}$	$P_5 = g^{S_5}$

The key derivation of key S_5 from key S_1 results in computing the child key $S_3 = P_2^{S_1}$. Then the key of the child of node 3 is computed as $S_5 = P_4^{S_3}$.

Complexity. The scheme presented above creates n private and public keys, therefore both public and private space complexity is $O(n)$. The computational complexity of the scheme is $O(h)$, where h stands for the height of the poset.

4 Adaptation for Arbitrary Posets

In reality, V-posets are rare, especially if they should also be cover various. Therefore in this section we generalize our scheme to arbitrary posets.

The general idea of the solution is to embed arbitrary posets into cover various V-posets, by inserting virtual nodes. Figure 1 shows in the middle a poset where all nodes at the bottom have three parents. The poset to the right has been extended with one virtual node v , so that each non-maximal node has exactly two parents. The relations between the original nodes, however, remain the same

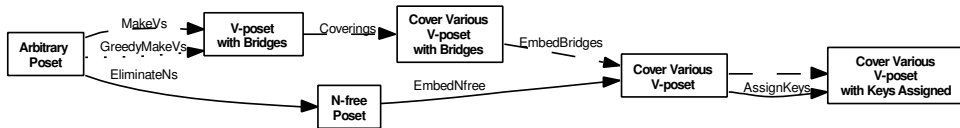


Fig. 2. Flow diagram of the embedding algorithms

after insertion. We call an embedding algorithm satisfying this invariant a proper embedding.

For virtual nodes we do not need to store the private keys, therefore the embedding step does not affect the private space complexity. However, the public space complexity is linear in the number of all nodes, original plus virtual. The complexity of the key derivation increases if the height of the embedded poset increases. The question arises how many virtual nodes must be inserted during the embedding phase, and how the height increases. Therefore with each given procedure we provide the corresponding complexity bounds.

Below we provide two proper embedding algorithms, that embed an arbitrary poset into a cover various V-poset, to which one can apply the scheme presented in the previous section. The first method, Phase Embedding, provides a theoretical improvement. The second one, N-free Embedding, refines the first method, and gives a significant practical improvement over all the other approaches.

Figure 2 shows the flow diagram of applying these embeddings to an arbitrary poset. The parts of Phase embedding are marked with dashed edges, and those of N-embedding with solid edges. In the two following sections we present and analyze Phase Embedding and N-embedding in more detail.

4.1 Phase Embedding

In general, we have to solve three sub-problems to achieve a proper embedding. First of all, we solve the problem of nodes having more than two parents. Secondly, we make sure that different nodes, that have the exact same parents get different keys. This step makes the poset cover various. Lastly, we solve the problem of nodes with only one parent. These three steps will embed an arbitrary poset into a cover various V-poset.

Too large covering sets. The main problem we face in arbitrary posets are nodes with more than two parents. To deal with more than two parents covering one node, we first present a greedy algorithm that embeds an arbitrary poset into a poset where each node has at most two parents. The *GreedyMakeVs(Poset P)* algorithm performs transformations as shown in Figure 3. The runtime of the *GreedyMakeVs* algorithm is at most quadratic in number of nodes, and the upper bound for the number of inserted virtual nodes is n^2 . This shows a greedy upper bound of n^2 .

The *GreedyMakeVs(Poset P)* algorithm
 for each node $x \in P$ with $|C(x)| > 2$ do

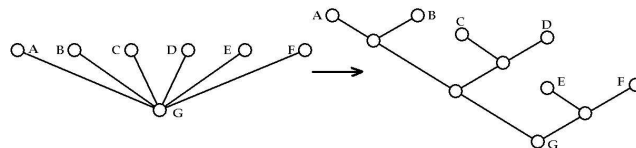


Fig. 3. GreedyMakeVs example: a structure as shown to the left is substituted by one as shown to the right

```

for each node  $y \in C(x)$  do  $Push(Q, y)$ ; od;
while( $Size(Q) > 2$ ) do
   $y := Pop(Q)$ ;  $z := Pop(Q)$ ;  $v := CreateNode(P)$ ;
   $InsertEdge(P, v, y)$ ;  $InsertEdge(P, v, z)$ ;
   $Push(Q, v)$ ; od;
 $y := Pop(Q)$ ;  $z := Pop(Q)$ ;
 $InsertEdge(P, x, y)$ ;  $InsertEdge(P, x, z)$ ; od;

```

Shared Covering sets. The $Coverings(Poset P)$ algorithm embeds a poset into a cover various poset. Virtual parents are created for nodes with the same covering sets. Every virtual parent thus created covers exactly one node, so there will be no nodes which share the same covering set anymore.

The $GreedyMakeVs(Poset P)$ algorithm makes sure that no node has more than two parents. Thus we can limit attention to two situations: covering sets of size 1 and 2.

```

The  $Coverings(Poset P)$  algorithm
for each node  $x \in P$  with  $C(x) \neq \emptyset$  do
  if ( $|C(x)| = 1$ ) then for each node ( $y \in P : y \neq x, C(y) = C(x)$ ) do
     $v := CreateNode(P)$ ;
     $InsertEdge(P, y, v)$ ; od;
  if ( $|C(x)| = 2$ ) then for each node ( $y \in P : y \neq x, C(y) = C(x)$ ) do
     $p = LeftParent(P, x)$ ;
     $v := CreateNode(P)$ ;  $w := CreateNode(P)$ ;
     $InsertEdge(P, y, v)$ ;  $InsertEdge(P, v, p)$ ;  $InsertEdge(P, v, w)$ ; od;
od;

```

The $GreedyMakeVs(Poset P)$ algorithm does not interfere with the algorithm $Coverings(Poset P)$, since virtual nodes added there will never have the same covering set as another node (whether original or virtual). Thus we can bound the number of inserted virtual nodes by $2n$, where n is the number of nodes in the original poset.

Bridges. The $EliminateBridges(Poset P)$ algorithm eliminates bridges. A single virtual node is created and becomes a parent for every bridge. External function $Bridge(Poset P, Node x)$ used in this algorithm, returns true if node x is a bridge in P , and false otherwise. The $EliminateBridges$ algorithm has a runtime linear in the number of nodes, and inserts one virtual node.

The *EliminateBridges*(Poset P) algorithm
 $v := \text{CreateNode}(P)$;
for each node x in P ; do
if ($\text{Bridge}(P, x)$) then $\text{InsertEdge}(P, x, v)$; od;

Theorem 1 *Applying the GreedyMakeVs, Coverings and EliminateBridges algorithms in order to an arbitrary poset P results in a proper embedding of P into a cover various V -poset.*

The proof of this theorem is given in Appendix C.

Main Theoretical Result. The crucial step in the embedding algorithm is how to deal with more than two parents. The greedy algorithm *GreedyMakeVs* shown above actually looks at edges, instead of nodes - hence the upper bound of n^2 . The two following theorems give bounds on the number of virtual nodes that need to be added in this step.

First we consider the theoretical minimum number of virtual nodes necessary to add in the worst case.

Theorem 2 *Consider an embedding \mathcal{E} that embeds arbitrary posets of n elements into posets where every node is covered by at most two other nodes. Then the number of virtual nodes added by \mathcal{E} is in the worst case at least $\frac{n^2}{16 \log n}$.*

This theorem shows that an arbitrary poset in theory requires quite a lot of virtual nodes to make a proper embedding. However, the proof (given in Appendix C) does not provide a concrete example where the bound is attained. Later we will see that practical cases require less virtual nodes.

An upper bound on the needed virtual complexity is given by the next theorem.

Theorem 3 *There exists an embedding \mathcal{E} that embeds arbitrary posets of n elements into a poset where every node is covered by at most two other nodes, adding $\frac{cn^2 \log \log n}{\log n}$ virtual nodes (where c is a constant).*

Unfortunately, the proof of this theorem is only partly constructive. Certain structures, namely complete bipartite subposets are guaranteed to be present as a sub-poset (if the poset is sufficiently dense), but they still have to be located. For more details we refer to Appendix C. However, the complexity of the embedding algorithm is not an issue here, so if necessary we can afford the CA to perform an exhaustive search for these structures during the setup phase. This means, that such a *MakeVs* algorithm adds at most $\frac{cn^2 \log \log n}{\log n}$ virtual nodes.

The number of virtual nodes added in the Phase Embedding algorithm is the sum of the numbers added during the three steps presented above. Therefore the number of virtual nodes inserted is at most $\frac{cn^2 \log \log n}{\log n} + 2n + 1$, which is dominated by the first term. The public space complexity of the scheme combined with Phase Embedding is therefore $O(\frac{n^2 \log \log n}{\log n})$. This improves the best known space complexity $O(n^2)$.

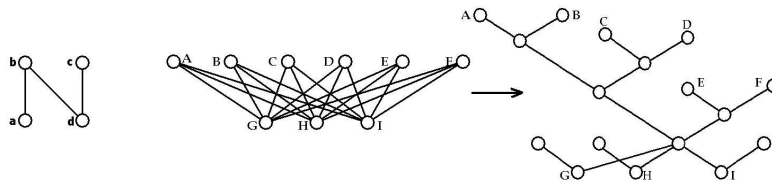


Fig. 4. N structure (left) and an example of QSP substitution (right)

4.2 N-embedding

As already briefly discussed in the previous section, being able to locate complete bipartite subposets is highly beneficial. For embedding these into cover various V-posets our Phase Embedding algorithm requires a linear number of virtual nodes. An efficient method for decomposing a poset into complete bipartite subposets exists in the literature (e.g. QSP decomposition [10]), but it only works on a specific class of posets, namely N-free posets. Thus, if we can efficiently make a poset N-free, we can apply the QSP decomposition. Then we can apply an optimized version of our Phase Embedding algorithm on the located complete bipartite subposets, as shown in Figure 4 (right).

In this section we present an efficient algorithm that embeds an N-free poset into a cover various V-poset increasing the size n by $\delta n = O(n)$, and increasing the height h by $\delta h = O(\min(h \log(n - h + 1), n))$. We end up with an algorithm that embeds any poset into an N-free poset, adding $\delta n = O(N)$ virtual nodes, where N stands for the number of N-spinning edges in the input poset, and increasing the height h by $\delta h = O(h)$. This gives an efficient solution for posets with a small number of N-spinning edges. In section 5 we show, that in practical hierarchies the number of N-spinning edges is indeed linear in n .

The N-poset is a poset on four elements a, b, c, d with $a \prec b, d \prec b, d \prec c$ and $a \parallel c, a \parallel d, c \parallel b$, as shown in Figure 4 (left). An N-spinning edge is the edge between b and d . An N-free poset does not contain any N in its Hasse diagram. For more definitions and notations corresponding to N-free posets, eg. the definition of a Quasi Series Parallel (QSP) poset and a QSP decomposition tree of a poset, we refer to Appendix B. The following theorem is the basis for the algorithm we present.

Theorem 4 *P is QSP if and only if P is N-free. [10]*

Embedding N-free posets. We have now all the necessary tools to introduce an embedding algorithm for N-free posets. The abstract structure we introduce here is *QSPtree* T , which is a representation of a QSP decomposition tree (see Appendix B) of a poset P we operate on. The nodes of the decomposition tree are represented with abstract structure *QSPtreeNode* w . Standard functions for binary trees, such as *Leaf(T, w)*, *LeftChild(T, w)* and *RightChild(T, w)* are used on *QSPtree* T .

The recursive algorithm *EmbedNfree(QSPtree T, QSPtreeNode w)* initially called with the root node of QSP tree as a parameter, traverses the QSP tree

representation T of an input poset P . When the node $w \in T$ currently being visited is a quasi series composition node $w = QS(M, N)$ (see Appendix B), the algorithm substitutes the complete bipartite subposet on sets M and N with a V -embedding of this subposet as shown in Figure 4 (right), just as the Phase Embedding would do in this case.

The $EmbedNfree(QSPtree\ T, QSPtreeNode\ w)$ algorithm

```

if( Leaf(T,w) ) then return w;
else  $P_1 := EmbedNfree(LeftChild(T, w))$ ;
    $P_2 := EmbedNfree(RightChild(T, w))$ ;
    $P := P_1 \oplus P_2$ ;
if( w is QSP(M,N)-composition ) then
  for each node  $y \in N$  do  $Push(Q, y)$ ; od;
  while( $Size(Q) > 1$ ) do
     $x := Pop(Q)$ ;  $y := Pop(Q)$ ;  $v := CreateNode(P)$ ;
     $InsertEdge(P, v, x)$ ;  $InsertEdge(P, v, y)$ ;
     $Push(Q, v)$ ; od;
   $x := Pop(Q)$ ;
  for each node  $z \in M$  do
     $v := CreateNode(P)$ ;
     $InsertEdge(P, z, v)$ ;  $InsertEdge(P, z, x)$ ; od;
return  $P$ ;

```

Result. The following theorem ensures the correctness of the $EmbedNfree$ algorithm. The proof may be found in Appendix C.

Theorem 5 *Each recursive call of $EmbedNfree(QSPtree\ T, QSPtreeNode\ w)$ returns a cover various V -poset for a poset corresponding to the QSP tree rooted in w .*

Number of Virtual Nodes and Height Increase The two theorems below provide the complexity properties of algorithm $EmbedNfree(QSPtree\ T, QSPtreeNode\ w)$. We assume, that the input poset $P(V, \leq_V)$ is N -free.

Theorem 6 *Let $n(P) = |V|$ and $\delta n(P)$ be the number of virtual nodes inserted by $EmbedNfree(QSPtree\ T, QSPtreeNode\ w)$. Then $\delta n(P) \leq 2n$.*

Theorem 7 *Let $h(P)$ be the height of poset P and $\delta h(P)$ be the increase of h while performing $EmbedNfree(QSPtree\ T, QSPtreeNode\ w)$. Then $\delta h(P) \leq \min(h(\log(n - h + 1) + 1), 2n)$.*

Runtime The runtime of the procedure $EmbedNfree(QSPtree\ T, QSPtreeNode\ w)$ is $O(t + n)$, where t stands for the size of QSP decomposition tree of a poset P .

Eliminating N-posets. We complete this section with an algorithm that is a proper embedding of an arbitrary poset into an N-free poset.

The algorithm *EliminateNs(Poset P)* inserts a virtual node on each N-spinning edge, therefore it eliminates all N-subposets from Hasse diagram of *P*. Verifying whether the edge is N-spinning is $O(n)$ in the size of input poset. Therefore *EliminateNs(Poset P)* runs in $O(n^3)$. It inserts $\delta n = O(N)$ virtual nodes, where *N* stands for a number of N-spinning edges in an input poset. The height is at most doubled, $\delta h(P) \leq h$. This is due to the fact, that new nodes are inserted at most one per an edge, therefore the number of nodes on any chain (also maximal) can be at most doubled.

The *EliminateNs(Poset P)* algorithm

```

for each pair  $(x, y) : x \prec y$  do // each edge in the Hasse diagram
  if( edge  $(x,y)$  is N-spinning ) then
     $v := CreateNode(P)$ ;
    InsertEdge $(x, v)$ ; InsertEdge $(v, y)$ ; od;

```

Performance parameters of N-embedding. To summarize this section, we have presented two algorithms, the *EliminateNs(Poset P)* algorithm and the *EmbedNfree(QSPtree T, QSPtreeNode w)* algorithm, that together constitute N-embedding, which embeds an arbitrary poset into a cover various V-poset. Table 2 shows the performance of these two algorithms.

We conclude, that the N-embedding adds in total $\delta n(P) = O(n + N)$ and increases the height by $\delta h(P) = O(\min(n + N, h \log(n + N - h + 1)))$.

Table 2. Performance of *EliminateNs* and *EmbedNfree*

	$\delta n(P)$	$\delta h(P)$
Input poset	0	0
After <i>EliminateNs</i>	<i>N</i>	<i>h</i>
After <i>EmbedNfree</i>	$3n$	$\min(2n + 2N, 2h(\log(n + N + h + 1) + 1))$

5 Applications and Tests

The previous section shows that, from a worst case complexity analysis point of view, the space complexity of the proposed scheme combined with Phase Embedding is slightly better than other efficient solutions by a factor $O(\frac{\log \log n}{\log n})$. In this section we present a practical average case analysis showing that, for the majority of cases the performance of our scheme combined with N-embedding is linear. We study examples from hierarchies occurring in different practical situations. The application scenarios for our test occur in industrial and research settings. In each case we identify a hierarchy in a group of people. The groups are large enough to analyze the asymptotic behavior of our scheme. We observe that, for all the test cases the space required for public parameters by the scheme combined with N-embedding is linear in the size of the input. To the best of our

knowledge no other proposal has linear public space complexity for average cases. The private space complexity is still linear for our scheme, like for other schemes.

We assume the existence of a database of documents. There are users, who have access to different subsets of the documents. The goal is to protect the data from being viewed by users, who are not allowed to access it. We assume that the storage for the database is essentially unlimited, and therefore data is never deleted. This is a realistic assumption, since many organizations need to have logs of all the transactions ever made (for instance banks), or all versions of developing projects (for instance software producers).

To challenge our scheme, we submit it to four different application scenarios: the Erdős Project Scenario, the DBLP Scenario, the Office Hierarchy Scenario and the PGP Scenario. We describe each in subsequent sections. The Erdős Project and Office Hierarchy correspond to centralized environments, the other two to decentralized environments. The Erdős project and PGP are based on authority as a social relation and the other two are based on a relationship of peers created online. Table 3 pictures this classification.

Table 3. Classification of Scenarios

	Centralized	Decentralized
Peers	Erdős	PGP
Authority	Office	DBLP

In the previous sections we showed that there are certain classes of posets for which our scheme performs better than for the general case. The upper bound for space complexity is $O(n)$ for N-free posets, where n is the number of nodes in the poset. We also provided an algorithm to embed an arbitrary poset into an N-free poset with $O(N)$ space complexity, where N stands for the number of N-spinning edges. Therefore we count the number of occurrences of N-spinning edges in each scenario as the most relevant performance parameter.

In each of the following sections we present a summarizing table. The tables show the number of N-spinning edges in a number of clusters, defined as the connected graphs in the considered hierarchies. Tests are applied to all the clusters, but in the tables only non trivial clusters are listed. The first row in the table is the cluster number, the second is the size in nodes, and the third row contains the number of N-spinning edges.

5.1 DBLP hierarchy

The DBLP (See <http://dblp.uni-trier.de/>) is a computer science bibliography stored in a big database of 675000 papers. Each paper determines a node, and each set of authors that is a subset of another set of authors determines an edge. An example is shown in Figure 9 in Appendix D. We apply our scheme to the hierarchy of articles, with the order given as mentioned above.

Each author receives his own keys, with which he is able to generate the keys for all his articles. The scheme creates a key for each article. The article

is encrypted with that key and stored in the database. When an author request access to an article, the server sends the encrypted version of requested article.

Table 4 shows the counts of N-spinning edges for non trivial clusters of the DBLP hierarchy. For example cluster number 31 presented in Figure 9 in Appendix D has 14 nodes, and contains only 2 N-spinning edges.

Table 4. the DBLP N-spinning edges counts

Cluster	1	2	4	11	13	22	23	29	31	33
Size	43	23	968	45	53	76	107	26	14	48
N-sp. e.	3	2	915	1	21	13	116	14	2	66

5.2 Erdős Collaboration Graph

Paul Erdős wrote hundreds of mathematical research papers, many in collaboration with others. His Erdős number is 0. Erdős' co-authors have Erdős number 1. People other than Erdős who have written a joint paper with someone with Erdős number 1 but not with Erdős himself, have an Erdős number 2, and so on. If there is no chain of co-authorships connecting someone with Erdős, then that persons Erdős number is said to be infinite. The data of Erdős' co-authors and their co-authors is gathered by the Erdős Number Project (See <http://www.oakland.edu/enp/>). A part of the Erdős hierarchy is shown in Figure 10 in Appendix D.

We build the following hierarchy. The nodes are single mathematicians. People with lower Erdős numbers are higher in hierarchy - the top member is Paul Erdős himself. There is an edge from A to B iff A has a bigger Erdős number than B and there is a chain of articles connecting A to B.

The Erdős hierarchy corresponds to the centralized model of an organization, where the leader has Erdős number 0, and for all the other staff the Erdős number indicates how long the line of command is to the top.

Table 5 summarizes the number of N-spinning edges in the Erdős hierarchy. The clusters are separated after removing Paul Erdős' node, which connects all the clusters from the top. An additional fourth row presents the number of all edges. In every cluster the number of N-spinning edges is less than the number of nodes, and significantly less than the number of edges.

Table 5. The Erdős Project N-spinning edges counts

Cluster	1	2	4	5	6	7
Size	18	7208	7	63	4	13
N-sp. e.	0	5478	0	8	0	1
Edges	17	10471	6	65	3	12

5.3 PGP trust chains

Following its intention as a 'cryptographic tool for the masses', PGP (See <http://www.rubin.ch/pgp/pgp.en.html>) breaks the traditional hierarchical trust architecture and adopts the 'web of trust' approach. There is no central authority

which everybody trusts, but instead individuals sign each others keys and weave a web of individual public keys interconnected by links formed by these signatures.

We define the trust relation as follows. A trusts B iff A signs B's key. The transitive closure of this directed relation gives the trust relation we analyze. An example is shown in Figure 11 to the left in Appendix D.

We assume, that all users want to keep their data (like name) secret, however they do not hide this data from entities they trust. Therefore the names and maybe some other data of the users, are encrypted with the keys, which only trusted users can generate.

Users lying on a cycle form a circle of mutual trust and are thus indistinguishable. Therefore, classes of users that lie on the same cycle in trust relation become a single node in the trust hierarchy we build. The poset on the resulting acyclic graph is given based on the trust relations between the cycles. There is an edge from class A to class B iff users in class B trust users from class A. Figure 11 shows the original PGP trust net to the left, and how we interpret the network as a hierarchy to the right. In the hierarchy, original cycles (classes) are merged into single nodes and original edges are transitively reduced. In the hierarchy of Figure 11 (right) in Appendix D there is one N-spinning edge.

The user keys and trust relations between them are called keyrings. These keyrings can be downloaded from the Internet. We separate the connected subgraphs (clusters) in every keyring, and analyze all non trivial clusters. The results are presented in Table 6. The additional fourth row presents the keyring, from which we took the analyzed cluster. Again, the numbers of N-spinning edges is always less than the number of nodes.

Table 6. PGP N-spinning edges counts

Cluster	1	1	1	1	3
Size	101	4444	676	42480	10466
N-sp. e.	21	1000	96	29690	1222
Keyring	UniZH key signing party	debian	philadelphia	current keys.pgp.net	2002-06-17 keys.pgp.net

5.4 Office Scenario

The last practical case concerns the organization of our own faculty (<http://www.eemcs.utwente.nl>), which has 223 academic staff and PostDocs, 260 PhD students, and 176 support staff, totalling 659 people. The faculty has 27 research groups in Electrical Engineering, Mathematics and Computer Science. Each of the 27 heads of group has two bosses: the dean and the director of one of the five research institutes. The academic staff and PostDocs have one boss: the head of the group. Most PhD students have two bosses: a full professor and a member of the academic staff. This information allows us to calculate the maximum number of N s in the organization: at most one N per PhD student and exactly one per group, i.e. $27 + 260 = 287$, which is less than half the total number of people in the faculty.

6 Conclusions

We propose a scheme for generating cryptographic keys in a hierarchy. The scheme is intended to be used for enforcing access control in encrypted databases. We provide a worst case complexity analysis showing that the scheme improves on all other published schemes. Our work is also the first to provide a detailed practical performance analysis, which shows that our scheme has a linear average performance for four practical case studies.

The scheme itself works for a narrow class of posets, but we present two methods to generalize the scheme to work for arbitrary posets. Both methods use virtual nodes to embed an arbitrary poset into a poset suitable for our scheme. The Phase Embedding method, based on theoretical results from order theory, results in an improvement of the worst case space complexity over all other published solutions. The best related approach has a private space complexity that is linear in the size n of the hierarchy, and a public space complexity linear in the number of edges e of the hierarchy (which in the worst case is quadratic in n). The space complexity of our scheme combined with Phase Embedding is $O(n)$ for private and $O(n^2 \frac{\log \log n}{\log n})$ for public space, hence the improvement is by a (small) factor $O(\frac{\log \log n}{\log n})$. Also, the computational worst case cannot be higher, and will increase by $O(h \log(n - h + 1))$ on average.

The N-embedding method gives a significant practical improvement compared to other solutions. The space complexity is bounded from above by $O(n)$ private and $O(n + N)$ public space, where N is the number of N -spinning edges (see section 4.2). Since $n < e$ and $N < e$, as in N we calculate only a subset of all edges, $O(e)$ is a guaranteed upper bound for the public space complexity. The upper bound for the computational complexity is in this case $O(\min(n + N, h \log(N + n - h + 1)))$, whereas the best solutions proposed by other authors yield $O(h)$ in the height h of an input poset.

To show that in practice N is significantly smaller than the number of edges e , we analyze four real hierarchies. Regardless of the nature of the hierarchy (i.e. centralized versus decentralized, or peer to peer versus authority based) we find that N is almost always smaller than the size of hierarchy n , and in only 2 out of 1343 cases N is of the same order of magnitude as n . Therefore the average complexity parameters for our scheme are $O(n)$ for both space complexities and $O(\min(n, h \log(n - h + 1)))$ for the computational complexity. We present the obtained complexities in Table 7, where we use the estimate $h \approx \log n$.

Table 7. Performance complexities

	Phase Embedding	N-embedding
Private Space Worst Case	$O(n)$	$O(n)$
Private Space Average	$O(n)$	$O(n)$
Public Space Worst Case	$O(n^2 \frac{\log \log n}{\log n})$	$O(e) = O(n^2)$
Public Space Average	?	$O(n)$
Computational Worst Case	$O(n^2 \frac{\log \log n}{\log n})$	$O(e) = O(n^2)$
Computational Average	$O(\log^2 n)$	$O(\log^2 n)$

Bibliography

- [1] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, 1983.
- [2] A. Blokhuis, J. M. Doumen, Z. Füredi, and H. A. Wilbrink. Manuscript in preparation.
- [3] Manik Lal Das, Ashutosh Saxena, Ved P. Gulati, and Deepak B. Phatak. Hierarchical key management scheme using polynomial interpolation. *SIGOPS Oper. Syst. Rev.*, 39(1):40–47, 2005.
- [4] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on information theory*, IT-22(6):644–654, Nov 1976.
- [5] Min-Shiang Hwang. A new dynamic key generation scheme for access control in a hierarchy. *Nordic J. of Computing*, 6(4):363–371, 1999.
- [6] Min-Shiang Hwang and Wei-Pang Yang. Controlling access in large partially ordered hierarchies using cryptographic keys. *J. Syst. Softw.*, 67(2):99–107, 2003.
- [7] Chu-Hsing Lin. Hierarchical key assignment without public-key cryptography. *Computers and Security*, 20(7):612–619, 2001.
- [8] Iuon-Chang Lin, Min-Shiang Hwang, and Chin-Chen Chang. A new key assignment scheme for enforcing complicated access control policies in hierarchy. *Future Gener. Comput. Syst.*, 19(4):457–462, 2003.
- [9] Stephen J. MacKinnon, Peter D. Taylor, Henk Meijer, and Selim G. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Trans. Comput.*, 34(9):797–802, 1985.
- [10] Rolf H. Möhring. Computationally tractable classes of ordered sets. *Algorithms and Order (I. Rival, ed.)*, pages 105–183, 1989.
- [11] Indrakshi Ray, Indrajit Ray, and Natu Narasimhamurthi. A cryptographic solution to implement access control in a hierarchy and more. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 65–73. ACM Press, 2002.
- [12] R. S. Sandhu. On some cryptographic solutions for access control in a tree hierarchy. In *ACM '87: Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*, pages 405–410. IEEE Computer Society Press, 1987.
- [13] M. Smith. Eternal bits – how can we preserve digital files and save our collective memory? *IEEE Spectrum*, 42(7):16–21, Jul 2005.
- [14] Maciej M. Syslo. A labeling algorithm to recognize a line digraph and output its root graph. *Inf. Process. Lett.*, 15(1):28–30, 1982.
- [15] Sheng Zhong. A practical key management scheme for access control in a user hierarchy. *Computers & Security*, 21(8):750–759, 2002.

A Basic Order Theory definitions

Partial Order is a binary relation over a set V which is reflexive, antisymmetric, and transitive. We denote this relation as \leq_V .

Partially Ordered Set (POSet) is a set V with a partial order \leq_V given on it, denoted $P = (V, \leq_V)$.

$y \in V$ **covers** $x \in V$ iff $x <_V y$, and there is no z such that $x <_V z <_V y$. We denote the covering relation as $x \prec_V y$. The set of nodes covering x is denoted $C(x) = \{y \in V : x \prec_V y\}$.

$y \in V$ **is incomparable with** $x \in V$ iff there is neither $x <_V y$ nor $x \prec_V y$ relation in P . We denote the incomparability relation as $x \parallel y$. This relation is reflexive, but not transitive.

Hasse Diagram is a graphical representation of a finite poset P , where each member of P is displayed as a node, and an edge connects y to x if $x \prec y$. Given that the nodes are labeled, a Hasse diagram uniquely determines a partial order. For example, Figure 5 shows the Hasse diagram of the powerset of the set $S = \{a, b, c, d\}$ ordered via subset inclusion.

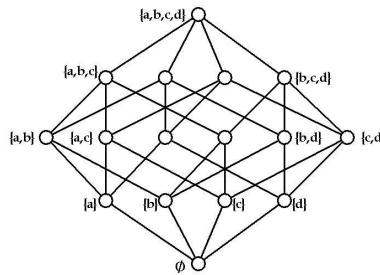


Fig. 5. Hasse diagram for $P(P(S), \subseteq_P)$

Maximal Element of a poset $P = (V, <_V)$ is any node $x \in V$, that is not covered by other nodes. We denote the set of all maximal elements $Max(P)$.

Minimal Element of a poset $P = (V, <_V)$ is any node $x \in V$, that does not cover other nodes. We denote the set of all minimal elements $Min(P)$.

V-poset is a poset $P = (V, <_V)$, where every node $x \in V$ is either maximal or covered by exactly two nodes. An example of non V-poset and V-poset is presented in Figure 1

Cover various poset is a poset $P = (V, <_V)$, where for each pair $x, y \in V$ $C(x) \neq C(y)$.

Bridge is a node, which is covered with exactly one other node. An example is shown in Figure 6.

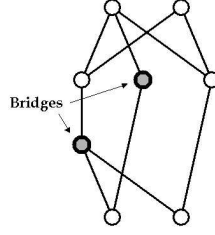


Fig. 6. Example of bridges

B Order Theory definitions corresponding to N-free property

Let $P_1 = (V_1, <_1)$ and $P_2 = (V_2, <_2)$ be partial orders with disjoint ground sets V_1 and V_2 . Let $V = V_1 \cup V_2$. Let $M \subseteq Max(P_1)$ and $N \subseteq Min(P_2)$.

Parallel composition is an order $P_p = (V, <_p) = P_1 \oplus P_2$, with $<_p$ defined as

$$x <_p y \iff \begin{cases} x, y \in V_1 \text{ and } x <_1 y & \text{or} \\ x, y \in V_2 \text{ and } x <_2 y \end{cases}$$

An example of a parallel composition is shown in Figure 7.

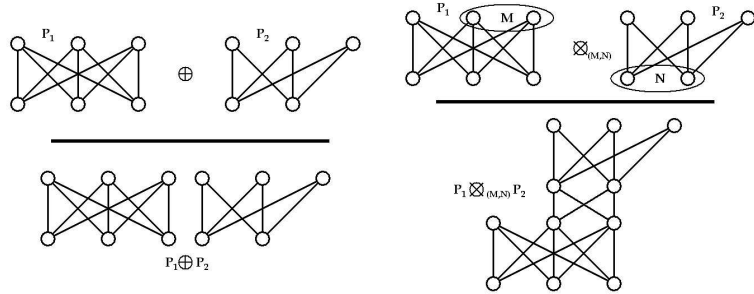


Fig. 7. Example of a parallel (left) and a quasi-series (right) composition

Quasi-series composition is a poset $P_{qs} = (V, <_{qs}) = P_1 \otimes P_2$ relative to M, N , with $<_{qs}$ defined as

$$x <_{qs} y \iff \begin{cases} x, y \in V_i \text{ and } x <_i y (i = 1, 2) & \text{or} \\ \exists a \in M, b \in N \ x \leq_1 a \text{ and } b \leq_2 y \end{cases}$$

An example of a quasi-series composition is shown in Figure 7.

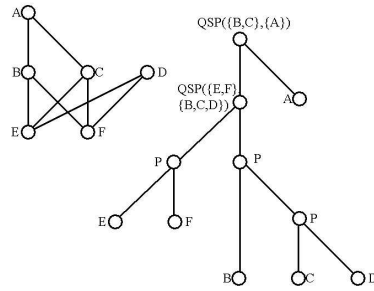


Fig. 8. Example of a QSP decomposition tree

QSP poset $P = (W, <)$ is QSP \iff

$$\left\{ \begin{array}{l} |W| = 1 \text{ i.e. } P \text{ is a one-element poset, or} \\ P \text{ is obtained by parallel composition or quasi-series} \\ \text{composition of two smaller QSP posets.} \end{array} \right.$$

QSP decomposition tree The recursive definition of a QSP poset yields a tree representation of QSP partial orders by a binary tree, the QSP-decomposition tree. An example is shown in Figure 8. The leaves of this tree represent one-element partial orders, and each interior node represents a parallel composition (label P) or a quasi-series-composition (label QS and sets M, N) of its left and right child. Order theory provides us with fast algorithms for constructing a decomposition tree [14]. Their running time is $O(n + m)$, where m stands for the number of edges of a Hasse diagram of an input poset.

An order is N-free if and only if it does not contain a subposet on four elements a, b, c, d with $a < b, c < b, c < d$ and $a||c, a||d, b||d$. Such a four element poset is called N-poset.

C Proofs of Theorems

Theorem 1. The result of the Phase Embedding is both a cover various V-poset as desired, and a proper embedding.

Proof. The *GreedyMakeVs(Poset P)* algorithm produces a poset, where each node has at most two parents, and that is a proper embedding of the original poset. After completing the *Coverings(Poset P)* algorithm, we obtain a cover various poset. This algorithm is designed such that the number of parents of an existing node remains unchanged, and such that any inserted virtual node is either maximal or has exactly two parents. Thus there is no interference with the *GreedyMakeVs(Poset P)* algorithm. Finally, the *EliminateBridges(Poset P)* algorithm leaves no bridges. The poset becomes a V-poset, since all bridges get a new parent, and the covering sets of other nodes do not change. For the same reason it remains cover various.

Theorem 2. Consider an embedding \mathcal{E} that embeds arbitrary posets of n elements into posets where every node is covered by at most two other nodes. Then the virtual complexity of \mathcal{E} is at least $\frac{n^2}{16 \log n}$.

Proof. Let V_n be the set of arbitrary posets with n elements, and let W_m be the set of posets with m elements, where every node is covered by at most two other nodes. A rough estimate on the sizes of both sets gives us $|V_n| \geq 2^{\frac{n^2}{4}}$, since there already are this many bipartite posets (with $\frac{n}{2}$ nodes on each layer), and $|W_m| \leq m^{2m}$, since every element can have at most two parents.

Thus, if all posets from V_n are to be embedded in some W_m , m has to satisfy the inequality

$$m^{2m} \geq 2^{\frac{n^2}{4}}.$$

Expanding this gives the bound in the theorem.

Theorem 3. There exists an embedding \mathcal{E} that embeds arbitrary posets of n elements into posets where every node is covered by at most two other nodes, which has virtual complexity $\frac{cn^2 \log \log n}{\log n}$ (where c is a constant).

Sketch of proof. We will only give a sketch of the proof of this theorem. The basic idea is to follow the greedy algorithm presented whenever this is sufficient. This algorithm will not be sufficient in all cases.

However, when this algorithm is not sufficient, we are dealing with a very dense poset. In this case, we will first search for complete bipartite sub-posets $K_{a,b}$ in the original poset, and replace them by a conforming structure, adding only $a+b$ virtual nodes. A recent extremal graph theory result [2] guarantees the presence of a complete bipartite sub-poset, when the poset is “dense” enough. Replacing these until the resulting poset isn’t sufficiently “dense” any more, and completing with the greedy algorithm, gives the indicated bound.

Theorem 5. Each recursive call $EmbedNfree(QSPtree\ T, QSPtreeNode\ w)$ returns a cover various V-order for a poset corresponding to the QSP tree rooted in w .

Proof.

- If w is a leaf, $EmbedNfree(T, w)$ returns a one element poset, which is a cover various V-order.
- If w is a parallel composition, $EmbedNfree(T, w)$ returns a parallel composition of cover various V-orders P_1 and P_2 generated for the children of w . A parallel composition of two cover various V-orders is a cover various V-order.
- In case w is a $QSP(M, N)$ -composition node, the returned poset P must preserve $x \prec y, x \in M, y \in N$. For that reason, first $|N| - 1$ virtual nodes are added, to obtain a node x , satisfying $x < y$ for any $y \in N$. All added virtual nodes have exactly two covering elements. Then x becomes a parent of all elements in M . To avoid the same covering sets, every node in M gets a second parent (virtual), which is different for any two elements of M . Figure 4 shows the effect of $EmbedNfree(T, w)$ in this case.

Theorem 6. Let $n(P) = |V|$ and $\delta n(P)$ be the number of virtual nodes inserted by $\text{EmbedNfree}(\text{QSPtree } T, \text{QSPtreeNode } w)$. Then

$$\delta n(P) \leq 2n$$

Proof. Let QS and Par be the sets of quasi series and parallel composition nodes in the QSP tree respectively.

Procedure EmbedNfree creates virtual nodes only when called for quasi series composition node of QSP tree. For a node $w = QS(M_w, N_w), M_w, N_w \subseteq V$, the algorithm creates $|M_w| + |N_w| - 1$ nodes. Therefore the total node increase

$$\delta n = \sum_{w \in QS} (|M_w| + |N_w| - 1) = \sum_{w \in QS} |M_w| + \sum_{w \in QS} |N_w| - |QS|.$$

A node $v \in V$ can be used as a member of N_w (or M_w) in only one quasi series composition, so

$$N_w \cap N_y = \emptyset \text{ and } M_w \cap M_y = \emptyset \text{ if } w \neq y.$$

Therefore

$$\sum_{w \in QS} |M_w| \leq \left| \bigcup_{w \in QS} M_w \right| \leq |V| = n$$

and

$$\sum_{w \in QS} |N_w| \leq \left| \bigcup_{w \in QS} N_w \right| \leq |V| = n.$$

This gives $\delta n \leq 2n - |QS| \leq 2n$.

Theorem 7 Let $h(P)$ be the height of poset P and $\delta h(P)$ be the increase of h while performing $\text{EmbedNfree}(\text{QSPtree } T, \text{QSPtreeNode } w)$. Then

$$\delta h(P) \leq \min(h(\log(n - h + 1) + 1), 2n).$$

Proof. Let QS and Par be the sets of quasi series and parallel composition nodes in the QSP tree respectively.

To give the upper bound on increase of the height $\delta h(P)$, we consider a modified QSP tree. We observe, that

$$h(P_1 \oplus P_2) = \max(h(P_1), h(P_2))$$

and

$$\delta h(P_1 \oplus P_2) = \delta h(P_i), \text{ for } i \in \{1, 2\} \text{ satisfying } h(P_i) = \max(h(P_1), h(P_2))$$

In other words for $w \in Par$

$$\delta h(\text{EmbedNfree}(w)) = \max \begin{cases} \delta h(\text{EmbedNfree}(\text{LeftChild}(w))) \\ \delta h(\text{EmbedNfree}(\text{RightChild}(w))) \end{cases}$$

We define a QSP* tree as a tree where each parallel composition node has exactly one child. This tree is obtained from QSP decomposition tree, by removing subtrees rooted in one of the children of each $w \in Par$. Let $y = LeftChild(w)$ and $z = RightChild(w)$. If $\delta h(EmbedNfree(w)) = \delta h(EmbedNfree(y))$, then subtree rooted in z is removed, otherwise we remove subtree rooted in y . Therefore all the subtrees, for which heights of corresponding posets are not propagated by parallel compositions, are removed from the QSP tree. In QSP* tree every quasi-series node adds one to the height: $|QS^*| = h(P)$. For $w \in QS^*$ algorithm *EmbedNfree* adds $\lceil \log_2 |N_w| \rceil$ to $\delta h(P)$. Therefore

$$\begin{aligned} \delta h(P) &\leq \sum_{w \in QS^*} \lceil \log |N_w| \rceil \leq |QS^*| \lceil \log |N_{max}| \rceil \leq h \lceil \log(n - h + 1) \rceil \\ &\leq h(\log(n - h + 1) + 1). \end{aligned}$$

As we add only $2n$ nodes in total, $\delta h(P) \leq 2n$ also holds. This gives the upper bound

$$\delta h(P) \leq \min(h(\log(n - h + 1) + 1), 2n).$$

D Illustrations from Application Scenarios

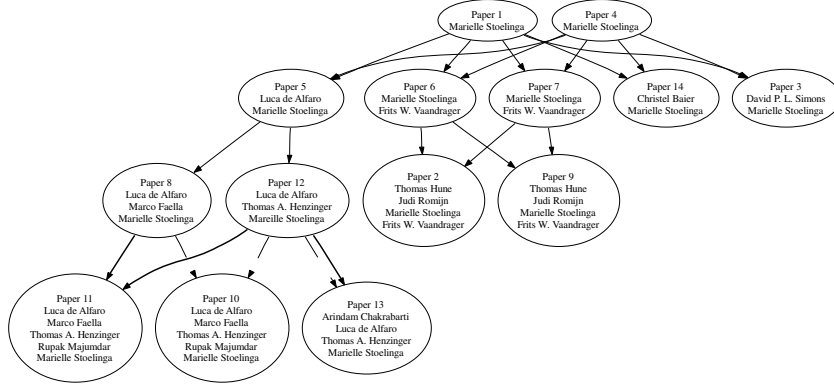


Fig. 9. Example of the DBLP hierarchy

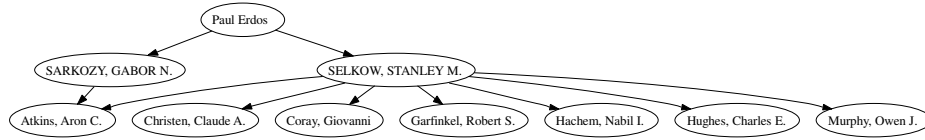


Fig. 10. Part of the Erdős hierarchy

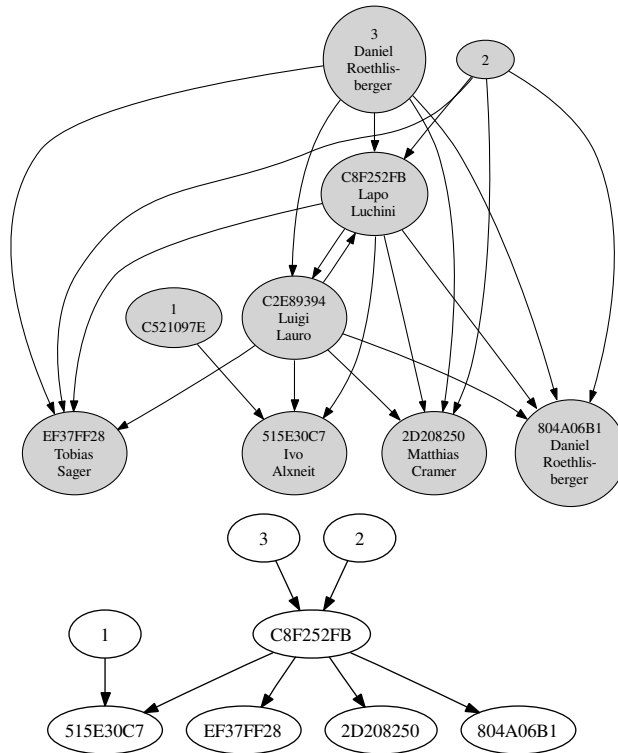


Fig. 11. Example of the PGP key ring not reduced (left) and reduced (right)