Storing Classified Files

Shai Halevi* Erez Petrank†

December 6, 1995

Abstract

We initiate a study of a natural problem in secure systems, namely - storing classified files on-line. A system of classified files contains a set of files (or documents), a set of users, and an authorization structure which defines the subset of files each user is authorized to see. We want a user in the system to be able to see every file in her "authorized list" but not any other file.

We present definitions of secure classified file systems and discuss various aspects of such systems, such as static versus dynamic models, resilience, the role of a central trusted entity, etc. We also present a highly space-efficient implementation for a secure system which is based on RSA.

*Laboratory for Computer Science, MIT, 545 Technology Square, Cambridge, MA 02139, USA. Email: shaih@theory.lcs.mit.edu

†Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4. Email: erez@cs.toronto.edu

1 Introduction

In this paper we put forward the problem of storing classified files on-line. A natural problem in a system where users have different classifications levels is how to allow each user to see all the files she has authorization for, but not any other file. The necessity of such a system is evident in organizations in which there is a sharing of information on a "need to know" basis only. In such a system we have a list for every user describing the set of files that this user is allowed to see, and our goal is to make sure that this user can see all and only the files in that list.

For systems in which the files are not too sensitive, one may apply an access-control based solution in which files are written on some shared disk, and the system checks for authorizations of each access-request for a file (e.g., UNIX). Although this solution suits many scenarios were access-control is required, it is not well suited for cases where the secrecy of the files is a real concern. Just to mention the more important reasons, note that the files are still kept in cleartext on the disk, so anyone who can gain access to the disk or the communication lines between the disk and the users can read the files. This is especially harmful when the system is distributed (in a popular client-server configuration) and the disk is accessed remotely (e.g., via NFS). Moreover, if the files are to be kept secret then the security of backups also becomes a real concern. Finally, it may be the case that even the system administrator (e.g. user '`root`') should not be allowed to see all the files.

Therefore when the files contain sensitive information they need to be stored *encrypted* in the system. Once encryption strategy is set, the problem of key distribution arises. That is, we need to encrypt the files using some keys and then somehow distribute these keys to the users so that each user will be able to see only the files in her "authorized list".

In this paper, we assume some secure encryption with which the files are encrypted, and discuss secure distribution of keys to the users. We discuss some of the security issues which should be addressed when designing a secure classified file system, and present definitions for secure distribution of keys in such systems. Last, we present a space-efficient implementation which meets these security definitions.

1.1 Naive solutions

The problem of key distribution for a set of classified files have two "obvious" solutions. The first solution is to encrypt each file once, and give the file-key to each user who is allowed to see the file. The main drawback of this implementation is that users are flooded with keys (one for each file). This property renders this solution impractical in cases where users have small (or expensive) memories (e.g., if keys are stored on smart-cards). Even if the user memories are not that small, it may be insecure to let each user store so many keys.

Another naive implementation is to give each user just one key, and encrypt the files many times. One encryption for each user who is authorized to see the file[1]. Here users only need to remember one key, but the cost in the public disk space is huge.

An obvious improvement on this is to encrypt each file F_j once with an independently randomly chosen key FK_j, and then to store encryptions of the key FK_j instead of encryptions of the full file F_j in the public space. In this manner, we have many encryptions of a smaller

[1] Namely, if a user i is authorized to see file j then we store an encryption of file j using the key of user i on the disk.

entity (we encrypt a key many times instead of encrypting a file many times). When a user wants to access a file F_j, she uses her key to decrypt the file-key FK_j, and then she uses FK_j to access the file F_j. Although this solution is clearly better that the previous one, it still has a significant cost in terms of public space. Also, we consider it a good property of a system if the public space is not changed unless a new file is entered. Note that in this case, every modification of the authorization list must be accompanied with a corresponding change in the information stored in the public space.

1.2 Our solution

We present a fairly simple scheme in which the files are encrypted only once in the public space, and each user needs to remember only one key. In addition, with each file there is one "number" related to this file which is also saved in the public space. At this cheap cost, we implement a secure classified file system whose security is provably equivalent to the RSA conjecture

Furthermore, our system has many other desirable features. The system is dynamic, in the sense that one can add a file, add a user, or add an authorization of a user to read a file. Among these modification, only adding a file requires changing the data stored in the public space (namely, addition of the encrypted file and the "number" corresponding to this file). Also, the system is fully resilient, in the sense that no collection of users is able to see a file unless one of the users is authorized to access this file. This holds even when these users do not comply with the regular system protocols and try to share their knowledge in order to try and gain extra authorization. Last, our scheme is simple and so is the intuition as to why it is secure. So although the proof may be technically involved, the implementation of the system is easy.

1.3 Related work

Although there has been work on introducing security into file-systems (e.g. [Ir95]), this work is mostly concerned with access control and not with secrecy of files. Other works include encryption of files and concentrate on facilitating access of users to their files. These works usually assumes that each user should only be able to read her own files (e.g., [Bl93]). We are not aware of any work which considered the problem of arbitrary authorization structures in a file system.

Another related work was done by Fiat and Naor ([FN93]) in the context of encrypting broadcast of movies. The problem there is how to broadcast encrypted movies to a set of users, such that each movie can only be seen by a privileged subset of these users. Since their setting involves a broadcasting environment, there has to be a preliminary stage in which each user gets some key, by which she should later be able to understand the broadcasted movie. This key has to allow a determination of any set of users as elligble to seeing the next film.

Although the problem statement and the various tradeoffs are quite similar to ours, there are some fundamental differences between these problems which make the solutions in [FN93] useless in our problem (and vise-versa). Moreover, the setting of [FN93] forces them to give each user $O(n \log^2 n)$ keys and to store in the public space additional $O(n^2 \log^3 n)$ keys with each encrypted movie[2]. Recall that in our suggestion of a secure file system, it is enough to

[2]This assumes full resilience. It is shown in [FN93] how to save space when the required resilience is smaller.

give each user only one key and it is enough to store each file once, with only one additional related number in the public space.

1.4 Contributions of this paper

A conceptual contribution of this paper is in initiating the study of secure classified file systems. We present the definitions as well as various properties which we believe should be discussed when designing such a system. Our technical contribution is in presenting an implementation for such a classified file system which is space-efficient, dynamic, and highly secure.

1.5 Organization

This paper is organized as follows: In §2 we give basic definitions and tools which we will use throughout the paper. In §3 we discuss variants of classified file systems, and give formal definitions for these variants. In §4 we describe RSA-based implementations of secure classified file systems.

2 Preliminaries

Negligible functions: We say that a non-negative function $f(n)$ is *negligible* if as n gets larger, $f(n)$ goes to zero faster than any polynomial in $1/n$. That is, for any constant $c > 0$ there is an integer n_c so that for all $n > n_c$, $f(n) < 1/n^c$.

Let f be a multi-variate function $f(n_1, n_2, \cdots, n_m)$. We use the notation $f(n_1, n_2, \cdots, n_m) = \text{negligible}(n_1)$ if for any choice of the other variables $n_2, \cdots, n_m$ the induced function of n_1 is negligible.

Efficient computations: We use the term *efficient algorithm* to say that the (possibly probabilistic) algorithm runs in polynomial time.

The RSA conjecture: In an RSA system there is a public modulus N which is a product of two large primes, and a public exponent $e \in Z_N$. The RSA conjecture asserts that given N, e it is infeasible to compute the function $f(y) = y^{1/e} \ mod \ N$. More precisely

Conjecture 1 *([RSA78]) For any probabilistic polynomial time algorithm $\mathcal{A}$*

$$\Pr_{N,e,x}[z \leftarrow \mathcal{A}(N, e, x^e \ mod \ N); \ z^e = x^e \ mod \ N] = \text{negligible}(|N|)$$

where the probability is taken over the "appropriate" choices of the composite N and the two elements $e, x \in Z_N$.

An "appropriate" choice of N usually means picking two primes p, q of the same size and computing $N = pq$. Sometimes the number N is required to have additional properties (e.g. be a Blum integer). An "appropriate" choice of e usually means either picking a random odd element in Z_N, or picking a random element in Z_N which is relatively prime to $\phi(N)$, or even using a fixed number (e.g. $e = 3$). Finally, the element x is usually just picked at random from Z_N. In this paper we use the following two variant of the RSA conjecture:

1. N is a product of two large primes and e is a random odd element in Z_N.

2. N is a product of two large primes so that $\phi(N)$ does not have any small factors other than 2, and e is relatively prime with $\phi(N)$.
The second variant is more restrictive than the first. In the sequel, we will show that the static scheme we suggest is equivalent to the first variant above, while the security of our dynamic scheme is equivalent to the second variant.

Computing gcd in the exponent: In our efficient implementation of a secure file system (see §4), which is based on the RSA conjecture, we use the following observation due to Shamir ([Sha82]).

Observation 2.1 *There is an efficient algorithm which on input* $\langle N, g^x, g^y, x, y\rangle$ *(everything mod* N*) computes* $g^{gcd(x,y)}$ *mod* N*. This is done by first computing* α*,* β *so that* $\alpha x + \beta y = gcd(x, y)$ *(over the integers) and then computing* $(g^x)^\alpha (g^y)^\beta$ *mod* N*.*

3 Secure Classified File Systems: Definitions

We consider a system of n users $u_1, \ldots, u_n$ and m files (or documents) $F_1, \ldots, F_m$. For each user there is a list of files which she may access. We call this list the *authorization list* of that user. The set of lists of all the users is the *authorization structure* of the system. A convenient representation of the authorization structure is a bipartite graph with the users on one side, the files on the other side, and edges connecting each user to all the files in the user's authorization list.

In order to keep users from accessing "the wrong files" in the system, all the files are stored encrypted and the key for each file is only given to authorized users. This brings up the problem of key distribution. In the systems we consider in this paper there is a "central authority" who is responsible for key distribution. We call this central authority the *distributor*. The distributor initiates the system by generating two types of keys.

- For each file in the system there is a *file-key*. This key is used to encrypt the file.
- For each user in the system there is a *user-key*. This key should allow the user to access all files in her authorization list.

The system may also contain additional public information which is used by the users for accessing their files.

The parameters of the file system include n (the number of users), m (the number of files), and k (the security parameter). The *space efficiency* of the system is measured by the size of the key that each user receives and by the size of the space needed in the public domain for the additional public information, i.e., the public information which the users must have in order to access files. In terms of time efficiency we require that the users and the distributor be efficient, i.e., probabilistic polynomial time in m, n, and k. Let us now present the definitions for the various interesting settings.

Remark: Notice that we may have more than one user with a specific authorization list. We choose to represent all these users by one "virtual" user in the authorization structure. Clearly, all these users may use the same user-key. Doing the same for files is a little more problematic: If we use the same key for two files, and later we add a user who is allowed to see only one of

these files, then we have to encrypt (at least) one of the files again and redistribute user-keys for all the users who are authorized to see this file.

3.1 The static model

The simplest setting is the static model, in which the system is set once by the distributor and the authorization structure is never changed afterwards.

Definition 3.1 *A* key distribution scheme for a static classified file system *is a pair of two (probabilistic polynomial-time) algorithms: The distributor algorithm and the user algorithm.*

- *The* distributor algorithm *D gets as input a security parameter 1^k and an authorization structure AS. The output of D contains a file-key FK_i for each file F_i and a user-key UK_j for every user U_j in the authorization structure AS. In addition, D may also output some other public information PI which can be used by the users to compute the file-keys. We denote this algorithm by*

$$D(1^k, AS) \rightarrow \langle \{FK_i\}_i, \{UK_j\}_j, PI \rangle$$

- *The* user algorithm *U gets in its input the security parameter 1^k, an authorization list of some user u_j (which we denote AL_j), the user-key UK_j for this user, the public information PI, and the name f_i of the file for which access is needed. If the user u_j is authorized to access the file f_i by the access structure with which the distributor was invoked, then this algorithm outputs the file key FK_i for f_i. We denote this by*

$$\forall f_i \in AL_j \qquad U(1^k, AL_j, UK_j, PI, f_i) \rightarrow FK_i$$

We proceed now to discuss the security of such a scheme. One obvious requirement is that given a user-key UK_j and the public information, it is hard to compute FK_i for any file f_i which is not in the authorization list of u_j (i.e., $i \notin AL_j$). However, this security requirement may not be enough. In particular, we require that even when several users collude, they will not be able to compute any file-key for a file not in the union of their authorization lists. If a system can withstand collusions of t users, we say that it is t-resilient.

Definition 3.2 *We say that a key distribution scheme for a static classified file system is* t-resilient *if for any authorization structure AS, any set of t users $u_{j_1}, \cdots, u_{j_t}$, any unauthorized file f_i (i.e., which is not in any of the authorization lists of these users), and any probabilistic polynomial time algorithm $\mathcal{A}$, it holds that*

$$\Pr\left[\mathcal{A}(1^k, AS, UK_{j_1}, \cdots, UK_{j_t}, PI) = FK_i\right] = negligible(k)$$

where the probability is taken over the random choices of the distributor (who picks the user keys and the public information) as well as the random choices of $\mathcal{A}$.

We say that a static scheme is fully resilient *if it is n-resilient (where n is the number of users in the system).*

Remark: The above definitions assume an oblivious adversary who chooses the set of users before the system is set-up. We discuss an adaptive adversary (who chooses the users in an adaptive fashion) in the context of the dynamic model hereafter.

(A very important) Discussion: The definition above only offers security in the sense that the adversary *cannot find keys* of files. In fact, what we need is a much stronger notion of

security: We need to prevent the adversary from *knowing anything about these keys*. That is, we want the keys to be *polynomially indistinguishable* from random strings given the relevant inputs (See [GM82]).

Luckily, there is a common simple heuristic for transforming a scheme which is resilient in the weak sense above, into one which is secure in the sense of indistinguishability: The idea is to introduce an intermediate step of secure hashing between the key distribution and the actual encryption of the files. Namely, rather than using FK_i to encrypt the file f_i, we instead use $H(FK_i)$ as the encryption key, where H is an appropriate hash function. It is widely believed that such "secure hashing" guarantees indistinguishability. Indeed, if we model H as a random oracle (See [BR93] for a complete discussion and many examples) then it is possible to prove that the resulting scheme is secure in the sense of indistinguishability.

3.2 The dynamic model

A more interesting model to consider is the dynamic model, in which the authorization structure may change with time, and the distributor has to maintain it on-line. Changes to the structure may include addition of users, files, or addition of files to authorization-lists of existing users (allowing these users to access more files).

It does not seem useful to consider also deletion of users, files, or authorizations from the authorization structure, since once a user is given authorization to access a file, she can copy it to her own space, and therefore it would be unrealistic to later revoke her authorization to access this file.

A dynamic system is initiated by executing the distributor algorithm on some (perhaps empty) authorization structure, as in the static case. The distributor distributes a user key to each user and outputs some public information as before. In addition, it may also keep some private values which will enable it to process future modifications.

After this initial step, requests for adding users, files and authorizations may be invoked. As a consequence, the distributor may output a new user-key (which allows a user to access more files) or new file-keys (usually for new files), and can possibly also modify the public information and his own private values.[3]

Although it is not necessary to keep the current authorization structure explicitly written in the system, the history of the system always determines uniquely the current authorization structure of the system.

Definition 3.3 *A* key distribution scheme for a dynamic classified file system *consists of five (probabilistic polynomial-time) algorithms: Distributor, User, Add_User, Add_File and Add_Authorization.*

- *The distributor algorithm is defined like in the static model, except that in addition to the file-keys user-keys and public information, it can also output some secret value SV. We denote it by*

$$D(1^k, AS) \rightarrow \langle \{FK_i\}_i, \{UK_j\}_j, PI, SV \rangle$$

- *The user algorithm is defined exactly as in the static case, where all the input parameters in this case are the "current" parameters when this algorithm is invoked. We denote this algorithm by*

[3] In §3.3 we discuss the possibility to reduce the dependency on the distributor in this process.

$$U(1^k, AL_j, UK_j, PI, f_i) \rightarrow FK_i$$

- *The Add_User algorithm gets as input the security parameter, the current authorization structure and an authorization list for the new user. In addition it also gets the public information and the secret values of the distributor. It adds a new user with the specified authorization list to the authorization structure and outputs a user-key UK for the new user, the (possibly modified) public information PI' and the (possibly modified) secret values SV'. It is denoted*

$$AddU(1^k, AS, AL, PI, SV) \rightarrow \langle AL, UK, PI', SV' \rangle$$

- *The Add_File algorithm gets as inputs the security parameter and the current authorization structure as well as the current public information and secret values. It adds a new file (which no user is authorized to access yet) to the authorization structure and outputs a file-key FK for the new file[4], the (possibly modified) public information PI' and the (possibly modified) secret values SV'. We denote it by*

$$AddF(1^k, AS, PI, SV) \rightarrow \langle AS, FK, PI', SV' \rangle$$

- *The Add_Authorization algorithm gets as inputs the security parameter, the current authorization structure, a user and a file, and the public information and secret values. It adds f to u's authorization list and outputs a new user-key UK' for the given user, the (possibly modified) public information PI' and the (possibly modified) secret values SV'. We denote it by*

$$AddE(1^k, AS, u, f, PI, SV) \rightarrow \langle UK', PI', SV' \rangle$$

The requirement from the system is the same as in the static case. That is, we require that for any user u_j and any file f_i in AL_j we get $U(1^k, AL_j, UK_j, PI, f_i) = FK_i$.

In order to define the security of a dynamic scheme we first have to describe the adversary model. We consider an adversary who may initiate a system and then interact with it. We allow the adversary to select the authorization structure, and use whatever order of updates to the system he wishes. The adversary may look at the public information, and he may also choose to get the keys of selected users. His goal at the end of this interaction is to output the key of a "new" file. Namely, a file that none of the users, whose keys were revealed is authorized to access.

The adversary is modeled as a probabilistic polynomial time machine that gets as an input the security parameter 1^k, and has access to a black box (representing the system) to which he can make the following queries.

Distribute(1^k, AS) which runs the algorithm D above. This query returns only the public information (PI) part of the output of D.

Reveal-user(u_j) which returns the current user-key UK_j of user u_j.

Add-user(AL) which runs the *Add_User* algorithm to add a new user with authorization list AL to the current authorization structure. This query returns only the PI' part of the algorithm's output.

[4]The natural use of this key is to encrypt the new file using some standard encryption scheme, and then store the encrypted file in the public domain. We do not discuss the encryption scheme here.

Add-file() which runs the *Add_File* algorithm to add a new file to the current authorization structure. This query returns only the PI' part of the algorithm's output.

Add-authorization(u,f) which runs the *Add_Authorization* algorithm to add file f to the authorization list of user u. This query returns only the PI' part of the algorithm's output.

The black box virtually contains all the system inside it in the following sense. The answers of the black box to the queries of the adversary are distributed by exactly the same distribution as the answers of the original algorithms in a real system are distributed when the actions requested by the adversary are executed by the distributor.

When the adversary is run with a security parameter 1^k, it may run for time which is polynomial in k and interact with the black box in the ways specified above. (Like invoking an oracle, we count the running time of the black box as one unit of time.) At the end of this execution, the adversary should output a pair (f_i, v) where f_i is a file and v is a value (which should be the key of file f_i).

For a given execution of adversary $\mathcal{A}$, we say that the file f_i in the adversary's output is *fresh* if the adversary did not issue any "Reveal-user" query to a user which had f_i in its authorization list at the time of the query. That is, the key to f_i was never revealed to the adversary by trivial means.

Definition 3.4 *We say that a key distribution scheme for a dynamic classified file system is fully resilient if for any probabilistic polynomial time adversary* $\mathcal{A}$

$$\Pr\left[\mathcal{A}(1^k) = \langle f_i, FK_i\rangle \text{ and } f_i \text{ is fresh }\right] = negligible(k)$$

where the probability is taken over the random choices of the black box and of $\mathcal{A}$.

The scheme is t*-resilient if the above holds when* $\mathcal{A}$ *is allowed to make "Reveal-user" queries to at most* t *different users throughout the interaction. Namely, the above probability is conditioned on the event that* $\mathcal{A}$ *issues at most* t *"Reveal-user" queries.*

Remark: In the above definition we assumed that the adversary uses the security parameter 1^k throughout his calls to the black box. This models the "real" world in which the adversary is required to deal with an on-going system. Theoretically, we could allow the adversary to run the black box with other parameters. (We assume that the black box doesn't allow the adversary to change the security parameter in his various calls.) Allowing a very low security parameter, would make it easy for the adversary to break the system but this success would not have any interesting implications. On the other hand, we could let the adversary invoke the black box with a very big security parameter but this doesn't seem to be right since the system uses operations which are too expensive (in time) for the adversary to access. We choose to allow the adversary use only his given security parameter. Formally, it can be viewed as letting the black box share the security parameter of the adversary.

3.3 Reducing the role of the central authority

In the above, we assumed the existence of some central trusted party (the distributor) which controls all the modifications of the authorization structure. However, it is sometimes desirable to reduce the amount of trust that has to be put on the central authority. Indeed, it is sometimes possible to execute some of the modification algorithms without the distributor. In particular,

we would like a user u to be able to delegate authorization for a file f to some other user u' without revealing any other secrets (and also without affecting the space efficiency of the system).

3.4 Keeping the authorization structure hidden

In some cases it may be desirable to keep the authorization structure hidden from the users (that is, we want that a user will not know the authorization lists of other users). Clearly, this structure contains information about the abilities of the users in the system, and there is no reason to keep it public. In the schemes we present in this paper, it is enough that a user knows its own authorization list.

4 An Efficient Implementation Based on RSA

In this section we describe an implementation of a fully resilient key-distribution scheme in which each user only needs to store one secret value. In addition the implementation also requires that we keep $m+1$ public values, where m is the number of files in the system (this last requirement can be avoided using some simple heuristics which we describe in Appendix 4.2). The security of this implementation is provably equivalent to the RSA conjecture.

4.1 The Static System

We start with a key-distribution scheme for a static system and prove its security. In §4.3 we show how this scheme can be extended to handle the dynamic case. This scheme is based on RSA, and resembles Shamir's pseudo-random number generator ([Sha82]). Like in RSA we use a public modulus N which is a product of two large primes, where the primes themselves are kept secret. In addition, we use a bunch of public exponents (one for each file) $e_1, \cdots, e_m$, where the e_i's are relatively primes.

To generate the keys we first select a (secret) random value $g \in Z_N^*$, and compute $E = \prod_i e_i$. The file-keys are set to $FK_i = g^{E/e_i} \ mod \ N$. Intuitively, one should note that there is no apparent way of getting one file key given another file-key. On the other hand, a knowledge of g discloses all the file-keys.

Another way to describe the file keys is as RSA-decryptions of a single random value v with respect to the various exponents $e_1, \cdots, e_m$. That is, there is a single value v (which is in fact g^E), and the file-key FK_i is $v^{1/e_i} \ mod \ N$ for all i, $1 \leq i \leq m$.

To generate a user-key for user u_j who is *not allowed* to see the files $F_{i_1}, \ldots, F_{i_l}$, we multiply $e_{i_1} \cdots e_{i_l}$ and set this user key to be $UK_j = g^{(e_{i_1} \cdots e_{i_l})} \ mod \ N$. Using this key the user is able to compute any file-key other than $FK_{i_1}, \cdots, FK_{i_l}$, but should not be able to get any of the non-authorized file-keys.[5]

The static scheme - formal description: Given a security parameter 1^k and an authorization structure AS with m files and n users, the distributor algorithm is as follows:

[5] An interesting point is that the distributor does not need to know the prime factors of N in order to generate the keys. It is enough that he knows g.

1. Generate a random RSA modulus N and pick uniformly a random element $g \in Z_N$.
2. Pick uniformly m odd elements $e_1, \cdots, e_m \in Z_N^*$ so that the e_i's are relatively primes.
3. Set the file-key for file f_i to be $FK_i \stackrel{\text{def}}{=} g^{E/e_i} \ mod \ N$ (where $E \stackrel{\text{def}}{=} \prod_{i=1}^{m} e_i$).
4. For each user u_j with authorization list $AL_j \subset \{1, \cdots, m\}$ we compute the user key hspacelin $UK_j \stackrel{\text{def}}{=} g^{E_{\overline{AL_j}}}$ (where $E_{\overline{AL_j}} = \prod_{i \notin AL_j} e_i$).
5. The public values in the system are $N, e_1, \cdots, e_m$.

A user u_j with authorization list AL_j, who knows UK_j (and also knows N and all the e_i's) can compute the file-key FK_i for any $i \in AL_j$ in the following way:

$$FK_i \leftarrow (UK_j)^{E_{AL_j}/e_i} \ mod \ N \quad (\text{where } E_{AL_j} = \prod_{i \in AL_j} e_i)$$

The user is able to compute the exponent since he knows all e_i's. The resulting number is indeed FK_i since

$$FK_i = g^{E/e_i} = \left(g^{\prod_{i \notin AL_j} e_i}\right)^{(\prod_{i \in AL_j} e_i)/e_i} = (UK_j)^{E_{AL_j}/e_i}$$

We next show that this construction indeed satisfy our security requirements.
Theorem 1 *If the RSA conjecture holds, then the RSA-based static scheme is fully resilient.*
Proof: *see Appendix A.*

4.2 Properties of the static scheme

Eliminating the public-values: In a system which contains many files, it may be inconvenient to store that many public values. One way around it is to have the center output a single value e, and then to set $e_i =$ the i'th prime larger than e. The security of this solution can be shown to be equivalent to the RSA conjecture using essentially the same proof as in Appendix A.

A different way (which works as long as m is not too large) is to choose the e_i's as small numbers, to store each e_i with the the i'th file, and to give each user u_j the value $E_j = \prod_{i \in AL_j} e_i$ together with UK_j (where the product in E_j is done *over the integers* and not over Z_N). Notice that neither the E_j's nor the e_i's need to be kept secret. To compute FK_i, the user (who knows UK_j, E_j) simply computes $(UK_j)^{E_j/e_i} \ mod \ N$.

The main disadvantage of the later solution is that the size of E_j is (at least) linear in the size of AL_j. However, this still offers two advantages over the naive solution. The first is that this E_j does not have to be kept secret, and the other is that we can choose the e_i's as small numbers, so we only add a few more bits for every file (as oppose to a full size key in the naive solution).

Manipulating authorization lists: The static scheme has some elegant properties with respect to authorization lists. In particular,

• Given a user-key UK_j which corresponds to authorization list AL_j it is easy to compute $UK_{j'}$ for any authorization list $AL_{j'} \subset AL_j$. This is since by definition we have

$$UK_{j'} = (UK_j)^{E_\Delta} \ mod \ N \quad (\text{where } E_\Delta = \prod_{i \in AL_j \setminus AL_{j'}} e_i)$$

• Given user-keys $UK_{j_1}, \cdots, UK_{j_t}$ which correspond to authorization lists $AL_{j_1}, \cdots, AL_{j_t}$, one can efficiently compute the user-key UK which correspond to authorization list $AL = AL_{j_1} \cup \cdots \cup AL_{j_t}$. For this we use Observation 2.1.

Recall that for all r we have $UK_{j_r} = g^{E_{\overline{AL_{j_r}}}} \ mod \ N$ (where $E_{\overline{AL_{j_r}}} = \prod_{i \notin AL_{j_r}}$). Moreover, since all the e_i's are relatively primes we have $gcd(E_{\overline{AL_{j_1}}}, \cdots, E_{\overline{AL_{j_t}}}) = E_{\overline{AL}}$. Therefore we can use the algorithm for computing gcd in the exponent to compute UK. This method can be made useful in the following manner:

Adding new users: Last, we would like to note a dynamic property in this static scheme. Although we described the above scheme as a static one, we never use our a-priory knowledge about the set of users. Indeed, we can use the same scheme even if we only know the number of files ahead of time, and learn the set of users and their corresponding authorization lists only afterwards. This makes this scheme "semi-dynamic" in the sense that we can easily add new users to the system. In Subsection 4.3 we show how this scheme can be made fully dynamic. Namely, how we can also add a new document or a new authorization to the system.

Reducing the role of the distributor: A nice feature of this scheme is that we can use the above property in order to add new users even without keeping any secret information in a trusted center, as long as the authorization-list of the new user is contained in the union of the authorization-lists of the other users.

To add a new user with authorization list AL, each user u_j (with authorization list AL_j) computes the user-key UK'_j corresponding to the authorization list $AL_j \cap AL$ and send this value to the new user. After receiving all the UK'_j values, the new user can compute UK which corresponds to AL since $AL = \bigcup_j (AL_j \cap AL)$.

4.3 A dynamic system

In this section we describe how the above scheme can be extended to support a dynamic system. Since the original scheme already supports addition of new users, all we need to do is to describe how it can also support addition of new files and addition of new authorizations.

In order to add new files, the distributor has to use the secret values g and $\phi(N)$. The initialization of the scheme and the file-keys and user-keys remain almost exactly the same as in the static case. The only difference (which is due to some technicality in the security proof) is that we pick N so that $\phi(N)$ does not have any small factors other than 2, and we pick all the e_i's to be relatively primes to $\phi(N)$.

When adding the $m+1$'st file to an authorization structure with m files, the distributor picks a new public value e_{m+1} which is relatively prime to all the previous e_i's and to $\phi(N)$,

computes $1/e_{m+1}$ *mod* $\phi(N)$ and sets

$$\begin{aligned} g' &\stackrel{\text{def}}{=} g^{1/e_{m+1}} \\ FK_{m+1} &\stackrel{\text{def}}{=} (g')^{E/e_{m+1}} \ mod \ N \quad (\text{where } E = \prod_{i=1}^{m+1} e_i) \end{aligned}$$

After setting these values, the center also replaces g with g' as its secret value. An important feature of this process is that the other file-keys remain unchanged. To see this, observe that for all $l \leq m$ we have

$$FK_l = g^{(\prod_{i=1}^{m} e_i)/e_l} = ((g')^{e_{m+1}})^{(\prod_{i=1}^{m} e_i)/e_l} = (g')^{(\prod_{i=1}^{m+1} e_i)/e_l}$$

To add an authorization, we just compute a new key for the user which allows him to access the new file as well as all the other files in his authorization list. The formal description of the dynamic scheme is omitted from this extended abstract.

Theorem 2 *If the RSA conjecture holds, then the RSA-based dynamic scheme above is fully resilient.*

The proof of this theorem is a generalization of the proof of Theorem 1 (with some added technicalities). The main idea of extending the static proof (See Appendix A) into the dynamic is the following. we assume some (polynomial) bound m on the number of files the adversary may add to the system. We (appropriately) select m e_i's. When the adversary adds a file to the system we randomly decide whether to associate this file to the unknown e or to the known e_i's. If the adversary asks to see the key of a user who is allowed to access the "bad" file (i.e., the file that is associated with e) then the experiment failed. But with probability at least $1/m$ the adversary never asks about such a key and it also outputs the file-key of the bad file. Note that m is polynomial and so if the success probability of the adversary is non-negligible then so is the success probability of the algorithm to invert the RSA.

References

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. Proceedings of the *First ACM Conference on Computer and Communications Security*, Fairfax, VA, 1993.

[Bl93] M. Blaze. A Cryptographic File System for Unix. Proceedings of the *First ACM Conference on Computer and Communications Security*, Pages 9–16, Fairfax, VA, 1993.
Also available on-line from ftp://ftp.research.att.com/dist/mab/cfs.ps

[FN93] A. Fiat and M. Naor. Broadcast Encryption. in Proceedings of *Advances in Cryptography CRYPTO'93*, Santa Barbara, CA USA, 1993. Springer-Verlag. pp. 480–491.

[GM82] S. Goldwasser and S. Micali, Probabilistic Encryption. in *JCSS* , Vol. 28, No. 2, 1984, pp. 270-299.

[Ir95] C.E. Irvine. A Multilevel File System for High Assurance. in proccedings of *IEEE Symp. on Security and Privacy*, CA, 1995, pp. 78–87.

[RSA78] R. Rivest, A. Shamir and L. Adelman. A Method for Obtaining Digital Signature and Public Key Cryptosystems. *Comm. of ACM*, 21 (1978), pp. 120–126

[Sha82] A. Shamir. On the Generation of Cryptographically Strong Pseudo-Random Number Sequences. *ACM Trans. Comput. Sys.*, 1 (1983), pp. 38–44

A Proof of Security

To prove Theorem 1 we assume that theRSA-based static scheme is not fully resilient, and deduce a counter-example to the RSA conjecture. By definition, if the scheme is not fully resilient then there is an authorization structure, a set of users, a file, and a probabilistic polynomial-time algorithm which witness this non-resilience (with non-negligible probability). The next claim asserts that in this case there is also a probabilistic polynomial-time algorithm which inverts RSA with the same probability.

Claim A.1 *Let AS be an authorization structure, $u_{j_1}, \cdots, u_{j_t}$ be users in AS, f_i be a file in AS which is not in any of the authorization lists $AL_{j_1}, \cdots, AL_{j_t}$, and $\mathcal{A}$ be a probabilistic algorithm, so that*

- *On input $\langle N, e_1, \cdots e_m, UK_{j_1}, \cdots, UK_{j_t}\rangle$ which is chosen with security parameter 1^k, $\mathcal{A}$ runs in time $T(k)$, and*
- $\Pr[\mathcal{A}(N, e_1, \cdots e_m, UK_{j_1}, \cdots, UK_{j_m}) = FK_i] \geq \epsilon(k)$

  where the probability is taken over the random choices of $N, e_1, \cdots e_m$, $UK_{j_1}, \cdots, UK_{j_t}$ with security parameter 1^k and over the random coin-tosses of $\mathcal{A}$ itself.

Then there exists another algorithm $\mathcal{B}$ which runs in time $T(k) + poly(k)$ on input $\langle N, e, y\rangle$ (where N is a random k-bit RSA modulus and $e, y \in Z_N^*$) and

$$\Pr_{N,x,e}[z \leftarrow \mathcal{B}(N, e, x^e);\ z^e = x^e \ mod\ N] \geq \epsilon(k)$$

where the probability here is taken over the random choices of $N, e, y = x^e$ and the random coin-tosses of $\mathcal{B}$.

Remark: Note that the parameters of AS (i.e., the number of users, files, and authorizations) do not appear in the running times of $\mathcal{A}, \mathcal{B}$. This is because our formulation fix those and only vary k. Therefore, $\mathcal{B}$ should be thought of as having the authorization structure AS (as well as the given users and file) "hard-wired" in its code. It will be obvious from the proof that the same construction works even if $\mathcal{B}$ gets those parameters as part of its input.

Proof: In the description of $\mathcal{B}$ we assume (w.l.o.g.) that $f_i = f_m$. That is, we assume that $\mathcal{A}$ gets several user-keys which do not reveal FK_m and outputs a guess for FK_m which is right with probability $\geq \epsilon(k)$.

The RSA-inverting algorithm: On input $\langle N, y, e\rangle$, algorithm $\mathcal{B}$ picks at random $e_1, \cdots, e_{m-1} \in Z_N^*$ which are relatively primes and are also relatively primes to e and sets $e_m \leftarrow e$. Then for each of the users $u_{j_1}, \cdots, u_{j_t}$ it computes

$$UK_j \leftarrow y^{E_{\overline{AL_j}}/e_m}\ mod\ N \quad (\text{where } E_{\overline{AL_j}} = \prod_{i \notin AL_j} e_i)$$

By our assumption, the file f_m is not in any of the authorization lists $AL_{j_1}, \cdots, AL_{j_t}$, i.e., we have $m \notin AL_{j_r}$ for all r, so $E_{\overline{AL_{j_r}}}$ contains e_m in the multiplication. Thus, computing the exponent involves only a multiplication of known numbers over the integers (and can be done without extracting modular roots). $\mathcal{B}$ now runs $\mathcal{A}(N, e_1, \cdots, e_m, UK_{j_1}, \cdots, UK_{j_t})$ until $\mathcal{A}$ halts and outputs a value w (which can be thought of as $\mathcal{A}$'s guess for the value of FK_m). Note that if x' is a number satisfying $x'^e = y$, then the system was set so that FK_m is

$$FK_m \;=\; x'^{(\prod_{i=1}^{m} e_i)/e_m} \;=\; y^{(\prod_{i=1}^{m-1} e_i)/e_m}$$

After getting $\mathcal{A}$'s guess for FK_m, $\mathcal{B}$ uses the extended gcd algorithm to compute α, β so that $\alpha e + \beta(\prod_{i=1}^{m-1} e_i) = 1$, and outputs $x = y^{\alpha} w^{\beta} \; mod \; N$. Notice that since the e_i's are relatively primes we have $gcd(e, \prod_{i=1}^{m-1} e_i) = 1$ so α, β exist.

Analysis of the inverting algorithm We start the analysis of $\mathcal{B}$ by asserting that the distribution which $\mathcal{A}$ "sees" while being invoked by $\mathcal{B}$ is the same as the distribution induced by the static scheme above:

By assumption, the input N of $\mathcal{B}$ is chosen according to the same distribution as in the scheme. Also, all the e_i's are chosen at random from Z_N subject to the constraint that they are odd and relatively primes, which is again identical to the distribution in the scheme. Thus the only thing left to show is that the UK_j's in the execution of $\mathcal{B}$ are the same as those in the scheme.

To see that, let x' be a value satisfying $(x')^e = y \; mod \; N$ (this is the value that $\mathcal{B}$ needs to compute). Then since $e = e_m$ we have for any u_{j_r}

$$\begin{aligned} UK_{j_r} &= y^{E_{\overline{AL_{j_r}}}/e_m} \qquad (\text{where } E_{\overline{AL_{j_r}}} = \prod_{i \notin AL_{j_r}} e_i) \\ &= \left((x')^{e_m}\right)^{E_{\overline{AL_{j_r}}}/e_m} \\ &= (x')^{E_{\overline{AL_{j_r}}}} \; mod \; N \end{aligned}$$

which is exactly the value of UK_{j_r} in our scheme for the same values of $N, e_1, \cdots, e_m$, assuming that g (the value chosen in the scheme) is equal to x' (the value which $\mathcal{B}$ needs to find). Since both these values are chosen from the same distribution (i.e., randomly from Z_N) we conclude that $\mathcal{A}$'s input distribution is indeed the same in both cases.

This implies in particular that $\mathcal{A}$ runs in $T(k)$ steps, so the running time of $\mathcal{B}$ is indeed $T(k) + poly(k)$ (recall that all the structure parameters are fixed). It also implies that the probability of $\mathcal{A}$ outputting a "correct" value of FK_m is at least $\epsilon(k)$. We now show that in this case $\mathcal{B}$ also outputs a correct answer.

Claim A.2 *If the output w of $\mathcal{A}$ in the execution of $\mathcal{B}$ is a correct value for FK_m, then the output x of $\mathcal{B}$ in this execution satisfies $x^e = y \; mod \; N$.*

Proof: Let us first see what it means for the output w of $\mathcal{A}$ to be a correct value for FK_m. Recall that all the file-keys in the scheme are RSA-decryptions of a single value $v(= g^E)$ w.r.t. the e_i's. Namely, FK_m is the value that - when raised to the e_m'th power - yields v. Note also that the value of v is uniquely determined given $N, e_1, \cdots, e_m$ and any one of the UK_j's, since $v = g^E = (UK_j)^{E_{AL_j}} \; mod \; N$ (where $E_{AL_j} = \prod_{i \in AL_j} e_i$).

We conclude that w is a correct value for FK_m if and only if it satisfies

$$w^{e_m} = (UK_{j_r})^{E_{AL_{j_r}}} \ mod \ N$$

for each of the UK_{j_r}'s in this execution of $\mathcal{B}$. This, in turn, implies that

$$w^e = w^{e_m} = (UK_{j_r})^{\prod_{i \in AL_{j_r}} e_i} = \left(y^{(\prod_{i \notin AL_{j_r}} e_i)/e_m}\right)^{\prod_{i \in AL_{j_r}} e_i} = y^{(\prod_{i=1}^{m-1} e_i)} \ mod \ N$$

Recall that the output of $\mathcal{B}$ is $x = y^\alpha w^\beta \ mod \ N$ where α, β satisfy $\alpha e + \beta(\prod_{i=1}^{m-1} e_i) = 1$. Thus we get

$$x^e = \left(y^\alpha w^\beta\right)^e = y^{\alpha e}(w^e)^\beta = y^{\alpha e}\left(y^{\prod_{i=1}^{m-1} e_i}\right)^\beta = y^{\alpha e + \beta(\prod_{i=1}^{m-1} e_i)} = y \ mod \ N$$

■

This concludes the proof of Claim A.1 and Theorem 1. ■