Distributed Encryption and Decryption Algorithms

André Postma, Willem de Boer, Arne Helme, Gerard Smit
University of Twente, Department of Computer Science
P.O.Box 217, NL 7500 AE Enschede, the Netherlands
Tel: +31-53-4894655 Fax: +31-53-4894590
e-mail: postma@cs.utwente.nl
WWW : http: // wwwspa.cs.utwente.nl / aid / fade / fade.html

Submission category:	Regular paper
Keywords:	dependable distributed systems, fault-tolerance, Byzantine failure model, security, cryptography
Approximate word count:	8490 words

Abstract

In this paper, we describe distributed encryption and decryption algorithms. These algorithms are required in order to make a dependable distributed data storage system in which data is stored in an encrypted form resilient to a number of arbitrarily faulty nodes in the system. By execution of these algorithms, a group of n nodes is able to encrypt or decrypt data with any secret cryptographic function in the system in the presence of up to t arbitrarily faulty nodes, whereas the following requirements are met:

- ❏ *loss of information of up to t nodes never leads to loss of any cryptographic key*
- ❏ *t or less nodes are unable to compute a secret cryptographic function and encrypt or decrypt data with it*
- ❏ *any group of n - t or more nodes may encrypt or decrypt data with any cryptographic function.*

1. Introduction

Nowadays, computer systems are used for a wide range of applications. For some applications, it is of crucial importance to minimize the probability that data stored in the system is lost or corrupted. Loss or corruption of data may be caused by humans, but also by hardware or software failures. In this paper, we will use the word ***node failure*** for either hardware or software failures in a node in the system. Node failures may be caused by transient or permanent errors. In order to make a system resilient to both types of errors, redundancy in the form of extra nodes can be employed. Provided that all data in the system is stored in a fault-tolerant way as a collection of data fragments on a number of nodes, arbitrary (or Byzantine) failures of one or more arbitrary nodes needs not lead to loss of data. Although as a result of a node failure, one or more data fragments may be lost, it may be possible to recover these fragments with the help of the other data fragments. After replacement of the failed node and recovery of all lost data fragments that were stored on it, the system is able to survive a subsequent node

failure. Thus, it is advantageous to recover lost data fragments as soon as possible.

In a multi-user environment, users may want to encrypt confidential data in order to protect their data against unauthorized reading and undetected mutilation by others.

Cryptography [1] is a commonly used technique to protect data against unauthorized reading and undetected mutilation of data. Roughly, two types of cryptosystems exist: symmetric and asymmetric (or public-key [2,3]) cryptosystems. Encryption and decryption of data is done much faster with a symmetric cryptosystem than with an asymmetric cryptosystem. Therefore, usually, a symmetric cryptosystem is used to encrypt or decrypt bulk data in a system. Encryption and decryption of data is done with a so-called ***cryptographic key***. Since, in a symmetric cryptosystem, the same cryptographic key is used for encryption and decryption, this key must be kept secret to prevent unauthorized reading and undetected mutilation of the data encrypted with it. For this purpose, the (symmetric) key is usually encrypted by means of an asymmetric (or public-key) cryptosystem.

Public key cryptography [2,3] is a commonly used technique to encrypt and decrypt data to protect it against unauthorized reading and undetected mutilation. In a public key cryptosystem, for every user u, two cryptographic functions E_u and D_u are defined, which are each other's inverses. User u keeps the details of D_u secret, whereas function E_u is made public to all users in the system.

In Section 2 we will show that for encryption as well as for decryption of data in order to protect this data against unauthorized reading and undetected mutilation, both the user's public and secret cryptographic function of the public-key cryptosystem are needed.

If we want to make a system with encrypted data resilient to one or more arbitrary node failures, it is essential that at a node failure the lost data fragments are recovered as soon as possible.

The recovery process consists of reconstruction of the lost data fragments with the help of the redundant information available in the remaining data fragments. However, if the remaining data fragments are in encrypted form, the user's secret cryptographic function of the asymmetric cryptosystem is needed to obtain the symmetric key with which the data has been encrypted, and be able to decrypt the data fragments (with the symmetric key) and perform the recovery process.

Since the *secret* cryptographic function is only known on the user node, it seems straightforward to let the user be responsible for the recovery process of his data fragments. However, this would imply that the *user node must function correctly to make recovery possible*.

The latter means that for the correctness of the system, we would have to rely on the correctness of a single node (viz. the user node). This is impossible, since the system should be resilient to a number of failures of *arbitrary* nodes.

Therefore, a *group* of nodes must be able to encrypt or decrypt data with the secret cryptographic function of any user. Notice that it is not possible to delegate this operation to a single

node, since unauthorized reading and updating is then possible as soon as this node fails (or is compromised).

In public key cryptography, encryption and decryption of data is done by executing a commonly known algorithm. The output of the algorithm depends on the input data, but also on the cryptographic key fed into the algorithm. Usually, for encryption as well as decryption, the same algorithm is used, only the cryptographic key fed into the algorithm, differs. Each user u makes the cryptographic key of the function E_u known to all other users, whereas it keeps the cryptographic key needed to perform function D_u secret.

In order to make a distributed multi-user system with encrypted data resilient to a number of up to t simultaneously failing nodes, encryption and decryption should be done by a group of n nodes ($n \geq 2t+1$) such that:

- ❏ loss of information of up to t nodes never leads to loss of any cryptographic key
- ❏ t or less nodes are unable to compute a secret cryptographic function and encrypt or decrypt data with it.
- ❏ any group of $n - t$ or more nodes may encrypt or decrypt data with any cryptographic function.

In this paper, we describe so-called ***distributed encryption and decryption algorithms***. By execution of such algorithms, a group of n nodes is able to encrypt or decrypt data with any secret cryptographic function in the system in the presence of up to t arbitrarily faulty nodes, whereas the above-mentioned requirements are met.

For this purpose, the information needed to generate a secret cryptographic function in the system (i.e. the secret cryptographic key) is split into a number of ***key fragments***. The key fragments are replicated and distributed among the nodes in such a way that:

R1. every key fragment is possessed by at least $t + 1$ nodes.

R2. t or less colluding nodes do not possess a sufficient number of key fragments to generate any secret cryptographic function and encrypt or decrypt data with it.

R3. any group of $n - t$ or more nodes possesses sufficient key fragments to encrypt or decrypt data with any secret cryptographic function in the system.

Let f be a secret cryptographic key that is split into j key fragments f_i (with $1 \leq i \leq j$), which together make up key f.

An important assumption we make is that the result of application of the encryption (decryption) algorithm with key f on a piece of data B is equal to repeated application of the encryption (decryption) algorithm, once with every key fragment f_i, starting with data B and at every iteration applied on the result of the previous iteration. Furthermore, we assume that application of the encryption (decryption) algorithm is commutative (i.e. the result of encryption (decryption) of data B with key f followed by encryption (resp. decryption) of the intermediate result with key g is equal to the result of encryption (decryption) of data B with key g followed by encryption (decryption) of the intermediate result with key f). Notice that both

assumptions are satisfied in the RSA-system [3].

A vast amount of literature has appeared on a similar, but different problem, viz. that of secret-sharing schemes or threshold schemes (e.g. in [1,4]). In most of these schemes, it is assumed that all nodes that possess cryptographic key fragments can be trusted. Namely, any node in the system is able to collect sufficient key fragments to generate any cryptographic function. In our algorithms, we do not want to rely on the trustworthiness of a single node, since the system should be resilient to a number of arbitrary failures of arbitrary nodes.

In [5], a distributed authentication service is described, in which the above three requirements R1 through R3 are met. This service is used to help the users generate a secret cryptographic key to set up secure message communication.

In our distributed encryption and decryption algorithms, the main focus is on a group of nodes being able to encrypt or decrypt data with any cryptographic function in the system, such that the requirements R1 through R3 are satisfied. Although this is also possible in the distributed authentication service in [5], the main focus in [5] is on a different problem.

The rest of this paper will be devoted to the distributed encryption and decryption algorithms. We describe how these algorithms can be constructed, and the number of nodes and key fragments needed for such an algorithm.

2. Encryption and decryption of data

In order to protect bulk data against unauthorized reading and undetected mutilation, a user u encrypts it with a secret cryptographic key of a symmetric cryptosystem. This symmetric key is needed to encrypt or decrypt data with it.

In order to keep this symmetric key secret, the user u encrypts this confidential symmetric key with the public cryptographic function E_u of an asymmetric cryptosystem. Then, the symmetric key (and thus also the data encrypted with it) can only be read after decryption with the user's secret cryptographic function D_u.

The encrypted symmetric key itself should also be stored in a fault-tolerant way in the system, such that arbitrary failure of one or more arbitrary nodes does not lead to loss of the symmetric key. This can be established by storing the symmetric key as a number of symmetric key fragments on a number of nodes in the system. If, due to a node failure, one or more symmetric key fragments are lost, these key fragments should be recovered as soon as possible.

Recovery of the lost symmetric key fragments can be done with the help of the remaining symmetric key fragments residing on other nodes. After the lost symmetric key fragments have been recovered, the public cryptographic function E_u is needed to encrypt the recovered symmetric key fragments.

However, since the public function E_u of any user u is known by all users in the system, any user v may generate random data B, encrypt the data B with E_u, and replace a symmetric key fragment encrypted with E_u by data B, which may not be automatically detected. There-

fore, in order to protect symmetric key fragments encrypted with E_u against undetected mutilation, user u calculates a hash value from the symmetric key fragment (or, more general, data), and encrypts this hash value with his secret function D_u. In this way, user u is able to detect all mutilations of the data (symmetric key fragment) or the hash value. Thus, in order to encrypt and decrypt symmetric key fragments (or, more general, data), both the user's public and secret cryptographic function are needed.

Usually, the order in which the various encryption steps are performed is as follows: first, a hash value of the data is calculated and encrypted with the secret function D_u. Then, the data together with the encrypted hash value is encrypted with the public function E_u.

In the rest of this paper, we will focus on the general problem of distributed encryption and decryption of (arbitrary) data with the secret cryptographic function of an asymmetric cryptosystem.

Notice that both encryption and decryption of data consists of execution of a cryptographic algorithm on both the data fragment and the hash value. The only difference is in the cryptographic key fed into the algorithm.

3.Distributed encryption and decryption algorithms

In this section we will introduce distributed encryption and decryption algorithms. These algorithms are executed by a group of n nodes, t of which may behave maliciously, in order to encrypt or decrypt data with any secret cryptographic function, in such a way that requirements R1 through R3 are satisfied.

We will first discuss the minimal number of nodes needed for these algorithms. Then, we discuss how the key fragments should be distributed among the nodes, and we investigate how the actual encryption and decryption takes place. Finally, we describe some ways to reduce the execution time of the algorithm and the required amount of data communication needed in the algorithm.

3.1. The minimum number of nodes in the distributed encryption and decryption algorithms

As discussed before, the cryptographic key used to perform a secret cryptographic function is split into a number of key fragments. Every node possesses a number of key fragments. It seems logical to give each node an equal number of key fragments, since *any* group of $n - t$ or more nodes should possess sufficient key fragments to be able to encrypt or decrypt data with the secret cryptographic function (requirement R3).

Let k be the number of key fragments given to each node. Because t or less colluding nodes may never possess a sufficient number of key fragments to generate any secret cryptographic function (requirement R2), the minimal number of key fragments $L(t, k)$ needed to generate a cryptographic function is given by:

$$L(t, k) \geq t \cdot k + 1$$

Furthermore, every key fragment should be possessed by at least t+1 nodes, so that loss of information on up to t nodes does not lead to loss of any key fragment (requirement R1). Then, a lower bound on the total number of key fragments $M(t, k)$ is given by:

$$M(t, k) \geq L(t, k) \cdot (t+1)$$

The $M(t, k)$ key fragments have to be distributed over the nodes. Since each node is given k key fragments, the number of nodes $N(t,k)$ can be expressed as a function of t and k:

$$N(t, k) = \min(M(t, k)) / k =$$
$$= (L(t, k) \cdot (t+1)) / k =$$
$$= ((t \cdot k + 1) \cdot (t+1)) / k =$$
$$= t \cdot (t+1) + (t+1) / k$$

We can see that $N(t, k)$ is minimal with respect to k if k goes to infinity, i.e.:

$$N(t, k) = t \cdot (t+1) \text{ if } k \rightarrow \infty$$

Since k can never be infinite in any practical situation, this is only a theoretical minimum of $N(t, k)$. The practical minimum of $N(t, k)$ becomes:

$$\min(N(t, k)) = t \cdot (t+1) + 1 \text{ if } k \geq t+1$$

The corresponding minimum number of key fragments k given to any node is given by:

$$\min(k) = t+1.$$

We will call a key fragment distribution ***optimal***, if the number of nodes in the system is equal to $\min(N(t, k))$ and the number of key fragments given to any node is equal to $\min(k)$.

3.2. The distribution of key fragments among the nodes

From the equations in the previous section one can see that in an optimal key fragment distribution the minimal number of key fragments $L(t, k)$ is equal to the minimal number of nodes $N(t, k)$ in the system:

$$\min(L(t, k)) = t \cdot \min(k) + 1 =$$
$$= t \cdot (t+1) + 1 =$$
$$= \min(N(t, k))$$

Let $\mathcal{N}$ be the set of nodes over which key fragments are distributed. Assume that the key fragment distribution is optimal. Then, the number of nodes n in set $\mathcal{N}$ is:

$$n = t \cdot (t+1)+1.$$

For any i, with $1 \leq i \leq n$, let N_i be a node in set $\mathcal{N}$. Then, $\mathcal{N} = \{N_1, N_2, \ldots N_n\}$. Furthermore, the number of key fragments k given to each node is given by:

$$k = t+1$$

For any j, with $1 \leq j \leq t \cdot (t+1)+1$, let f_j be a key fragment of cryptographic key f. With all key fragments f_j, with $1 \leq j \leq t \cdot (t+1)+1$, key f can be generated. Then, we propose the following key fragment distribution:

For any i with $1 \leq i \leq n$, for any cryptographic key f composed of n key fragments f_y (with $1 \leq y \leq n$), node $N_i \in \mathcal{N}$ possesses the k key fragments in the set $\{ f_j \mid j = 1 + ((i + n - m)$ ***mod*** *$n) \wedge$*

$1 \leq m \leq k\}$.

Example:

In this example, we illustrate the proposed key fragment distribution of a cryptographic key f for the case $t = 2$. It follows that $n = 7$ and $k = 3$. Furthermore, $\mathcal{N} = \{N_1, N_2, N_3, N_4, N_5, N_6, N_7\}$. In table 1, an overview is given of the key fragments possessed by each of the nodes N_1 through N_7.

node	possessed key fragments
N_1	f_1, f_7, f_6
N_2	f_2, f_1, f_7
N_3	f_3, f_2, f_1
N_4	f_4, f_3, f_2
N_5	f_5, f_4, f_3
N_6	f_6, f_5, f_4
N_7	f_7, f_6, f_5

Table 1: An optimal key fragment distribution for t = 2

Notice that the requirements R1 through R3 are satisfied. Viz., every key fragment is possessed by at least 3 nodes. Furthermore, there are no groups of less than 3 nodes that possess all key fragments. Finally, any group of 5 or more nodes possesses all key fragments. ❑

3.3. Distributed encryption and decryption

In this section we will describe how the actual encryption and decryption takes place. First, we state the required assumptions. It turns out that one of these assumptions can be easily satisfied if so-called key fragment pairs are generated. We will describe how these key fragment pairs can be generated using the RSA-cryptosystem. Finally, we describe the encryption and decryption algorithm.

3.3.1. Assumptions

Assume we have a system consisting of a set $\mathcal{N}$ of nodes, up to t of which may behave maliciously. For any i, with $1 \leq i \leq n$ (where $n = t \cdot (t+1)+1$) , let N_i be a node in set $\mathcal{N}$. For any j, with $1 \leq j \leq t \cdot (t+1)+1$, let f_j be a key fragment of cryptographic key f. With all key fragments f_j, with $1 \leq j \leq t \cdot (t+1)+1$, key f can be generated.

Then, we make the following assumptions for our distributed encryption and decryption algorithms:

A1. The key fragments f_j (with $1 \leq j \leq n$) of key f are distributed among the nodes in $\mathcal{N}$ as

described in Section 3.2. Furthermore, all correct nodes know how the key fragments have been distributed.

A2. Any correctly functioning node N_i of $\mathcal{N}$ is able to verify the correctness of every partially encrypted piece of data.

A3. All correct nodes in $\mathcal{N}$ agree upon the data that should be encrypted resp. decrypted. Let $B(N_i)$ be the data that should be encrypted resp. decrypted according to node N_i, then for any pair of correctly functioning nodes N_i and N_j, it holds that $B(N_i) = B(N_j)$.

A4. All correct nodes know (within known bounds) when the algorithm starts.

A5. At least 1 correct node starts the algorithm.

The assumptions can be satisfied as follows. Since the user knows all the key fragments f_j (with $1 \leq j \leq n$) of key f, it is easiest to let the user be responsible for the distribution of the key fragments of its secret cryptographic key f. The key distribution problem is far from trivial, however, it falls beyond the scope of this paper.

In Section 3.3.3. we show that assumption A2 can be easily satisfied if the RSA-system [2] is applied for encryption and decryption of data, by generating so-called ***key fragment pairs*** (described in Section 3.3.2.). In our algorithm, for simplicity reasons, we will assume that for encryption and decryption, RSA is used.

Assumption A3 can be satisfied by having the nodes in $\mathcal{N}$ perform Byzantine Agreement Protocols (see e.g. [6,7,8]).

Assumption A4 can be satisfied by executing the distributed encryption or decryption algorithm as soon as the Byzantine Agreement Protocol is terminated. Then, provided that only up to t nodes may function maliciously, assumption A5 is automatically satisfied.

3.3.2. Generating key fragment pairs with RSA

In RSA [2], two large different prime numbers are p and q are chosen. We choose r to be the product of p and q.

Now, a public cryptographic key e is chosen such that the greatest common divisor (or *GCD*) of e and $\varphi(r)$ is equal to 1. Thus:

$$GCD(e, \varphi(r)) = 1$$

Here, $\varphi(r)$ is Euler's totient function and is given by:

$$\varphi(r) = (p\text{-}1)\cdot (q\text{-}1)$$

Since e is relative prime to $\varphi(r)$, it has a multiplicative inverse in the ring of integers modulo $\varphi(r)$ (by Euclid's algorithm (see e.g. [9])). This multiplicative inverse of e is the corresponding secret cryptographic key d, which is defined by:

$$e \cdot d \equiv 1 \ (\textbf{mod}\ \varphi(r))$$

The public and secret cryptographic function (E resp. D) with key e resp. d on a piece of data B are defined by:

$$E(e, B) = B^{\,e} \ (\textbf{mod}\ r) \text{ and}$$

$$D(d, B) = B^{d} \ (\mathbf{mod}\ r)$$

In [2] it is proven that the cryptographic functions D and E are each other's inverses, i.e.

$$B \equiv D(d,E(e,B)) \equiv E(e,B)^{d} \equiv (B^{e})^{d} \equiv B^{e \cdot d} \ (\mathbf{mod}\ r)$$

In RSA, the numbers p,q, and d are kept secret, and the numbers e and r are made public.

For our distributed encryption and decryption algorithms to work, cryptographic keys must be split up into a number of key fragments. In RSA, this is easily done by splitting each of the cryptographic keys d and e into n key fragments:

$$d \equiv d_1 \cdot d_2 \cdot \ldots \cdot d_n \ (\mathbf{mod}\ \varphi(r))$$

$$e \equiv e_1 \cdot e_2 \cdot \ldots \cdot e_n \ (\mathbf{mod}\ \varphi(r))$$

The key fragments are selected in such a way that:

$$\forall\ i \in [1,n] : e_i \cdot d_i \equiv 1 \ (\mathbf{mod}\ \varphi(r))$$

For all i with $1 \le i \le n$, the pair $\{e_i,d_i\}$ is called a key fragment pair.

It can easily be seen that the public and secret cryptographic function with key e resp. d can now be written as:

$$E(e,B) = B^{e} \equiv$$

$$\equiv B^{e_1 \cdot e_2 \cdot e_3 \cdot \ldots \cdot e_n} \equiv$$

$$\equiv (\ldots ((B^{e_1})^{e_2}) \ldots)^{e_n} \ (\mathbf{mod}\ r)$$

resp.

$$D(d,B) = B^{d} \equiv$$

$$\equiv B^{d_1 \cdot d_2 \cdot d_3 \cdot \ldots \cdot d_n} \equiv$$

$$\equiv (\ldots ((B^{d_1})^{d_2}) \ldots)^{d_n} \ (\mathbf{mod}\ r)$$

Furthermore, for any key fragment pair $\{e_i,d_i\}$ (with $1 \le i \le n$), it holds that:

$$E(e_i,D(d_i,B)) \equiv$$

$$\equiv (B^{d_i})^{e_i} \equiv B \ (\mathbf{mod}\ r)$$

and

$$D(d_i,E(e_i,B)) \equiv$$

$$\equiv (B^{e_i})^{d_i} \equiv B \ (\mathbf{mod}\ r)$$

3.3.3. The algorithm

Assume we have a system consisting of a set $\mathcal{N}$ of nodes, up to t of which may behave maliciously. For any i, with $1 \le i \le n$ (with $n = t \cdot (t+1)+1$) , let N_i be a node in set $\mathcal{N}$. For any j, with $1 \le j \le t \cdot (t+1)+1$, let e_j (resp. d_j) be a key fragment of cryptographic key e (resp. d). With all key fragments e_j, (resp. d_j) with $1 \le j \le t \cdot (t+1)+1$, key e(resp. d) can be generated. We assume that, for all i with $1 \le i \le n$, the pair $\{e_i,d_i\}$ is a key fragment pair. We assume that the key fragments of key d are kept secret, whereas the key fragments of key e are made public to

all nodes. The algorithm is performed by a group of n nodes and consists of applying a cryptographic function D with the secret cryptographic key d on data fragment B. We assume that A1 through A5 from section 3.3.1. are satisfied.

By assumption A5, at least one correct node starts the algorithm. Assume that N_i (for certain i, with $1 \leq i \leq n$) is a correct node that starts the algorithm. We will refer to node N_i as the ***source node***. Now, $B(N_i)$ is the data fragment that should be encrypted (resp. decrypted) according to node N_i.

Let $KF(N_i)$ be the set of the k key fragments of d possessed by N_i (Here, by assumption A1, $k = t+1$). Then:

$$KF(N_i) = \{d_i, d_{i-1}, \ldots, d_{n-k+i-1}\}$$

Now, node N_i performs cryptographic function D on $B(N_i)$ with all the key fragments it possesses, resulting in a partially encrypted data fragment $P(N_i, B(N_i))$, which is defined by:

$$P(N_i, B(N_i)) = D(d_{n-k+i-1}, \ldots, D(d_i, B(N_i)) \ldots)$$

This partially encrypted data fragment should be relayed to one or more correctly functioning nodes that did not yet participate in the encryption process. In order to guarantee that at least one correctly functioning node receives the partially encrypted data fragment in the presence of up to t maliciously functioning nodes, the partially encrypted data fragment should be relayed to at least $t+1$ other nodes. Assume that N_j is a correctly functioning node that receives the partially encrypted data fragment from node N_i. Then, at receipt of the partially encrypted data fragment, N_j will first check if the data fragment was properly encrypted. For this purpose it performs the cryptographic function E with all key fragments e_y (with $1 \leq y \leq n$) that form a key fragment pair with the key fragments that are possessed by N_i. I.e. node N_j computes V, where:

$$V = E(e_i, \ldots, E(e_{n-k+i-1}, P(N_i,B(N_i))) \ldots)$$

Node N_j decides that the partially encrypted data fragment was properly encrypted iff

$$V = B(N_j)$$

since, by assumption A3, it holds that $B(N_i) = B(N_j)$.

To be able to perform this check, node N_j must know with which key fragments the data fragment has been encrypted. By assumption A1, N_j knows how the key fragments have been distributed among the different nodes, it is only required that N_j has some information about the path (i.e. the sequence of nodes) along which the data fragment has travelled from the source node to N_j. The required information is called the ***path information*** of the data fragment. The path information should reveal all *correct* nodes along which the data fragment has travelled from the source node to N_j. Notice that, if all nodes in the path indeed function correctly, the path information actually indicates the path along which the data fragment has travelled. For our algorithm, we simply add the following assumption:

A6. Every correct node knows the path information of any correctly partially encrypted data fragment that it receives.

A general way of providing the path information of a data fragment to a node that receives it, is to append the path information to the data fragment. Any valid message in the algorithm therefore should consist of a partially encrypted data fragment followed by the path information of that data fragment. In the next section we describe how the path information is constructed.

If node N_j decided that the partially encrypted data fragment was properly encrypted, then node N_j believes that node N_i has encrypted a data fragment of the correct data B, and thus, N_j cooperates in the encryption (resp. decryption) process. Node N_j now performs cryptographic function D on the partially encrypted data fragment $P(N_i, B(N_i))$ that it received with all the key fragments that it possesses, and that are not possessed by node N_i.

In other words, N_j produces a partially encrypted data fragment $P(N_j, P(N_i, B(N_i)))$, which is defined by:

$$P(N_j, P(N_i, B(N_i))) =$$
$$D(d_{v(m)}, \ldots, D(d_{v(1)}, P(N_i, B(N_i)) \ \ldots \ )$$

in which:

$$\{v(1), \ldots \ v(m)\} = KF(N_j) - (KF(N_i) \cap KF(N_j))$$

This partially encrypted data fragment is again forwarded to one or more correctly functioning nodes that did not yet participate in the encryption (resp. decryption) process, etc., until the secret cryptographic function D has been performed with all key fragments d_i ($1 \le i \le n$).

3.3.4. Constructing path information

In each phase of the distributed encryption and decryption algorithms described in the previous section, messages are *relayed* from one node to other nodes.

In this section, we describe the actions required when relaying a message. We start with some notations.

Notations

The following notations will be used:

1. For message m, we will denote a message digest of its data fragment mv as $md(mv)$.
2. For paths, we will use the following notations:
   - $()$ the empty path
   - (p) a path consisting of node p.
   - $(\underline{\boldsymbol{s}};p)$ a path consisting of path $(\underline{\boldsymbol{s}})$ concatenated with node p.
3. For any message with data fragment mv which has travelled from a source node s along any path $(\underline{\boldsymbol{p}})$ to a destination node a, the path information of that message is

denoted as *pinfo(mv,(s;**p**;a))*. (Notice that, because in every valid message *m*, a message digest *md(mv)* of the data fragment *mv* of *m* is integrated in the path information of *m*, the path information of *m* is a function of the path along which *m* has travelled, and its data fragment.)

4. For any path information *pinfo(mv,(s;**p**;a))* and any node *i*, by $\{pinfo(mv,(s;\mathbf{p};a))\}_i$ we will denote the path information *pinfo(mv,(s;**p**;a))* signed by *i*.
5. For any data fragment *mv* and any node *a*, by $\{mv\}_a$ we will denote the data fragment *mv* signed by *a*. Similarly, a data fragment *mv* signed by all nodes in path ***p*** is denoted by $\{mv\}_{\mathbf{p}}$.
6. For any data fragment *mv* and any nodes *s* and *a*, by *m(mv,(s;a))* we will denote the message with data fragment *mv* sent from *s* to *a*. Similarly, for any path ***(p)***, by *m(mv,(s;**p**;a;b))* we will denote the message *m(mv,(s;**p**;a))* relayed by *a* and sent to *b*.

Since the signatures on the path information must be unforgeable, every node itself must sign the messages it relays. Thus, the path information must be updated *during execution of the distributed encryption and decryption algorithm*. For signed messages we use the following definitions:

7. The path information of a message *m(mv,(s;a))* with data fragment *mv* sent from the source node *s* to a node *a* has been signed by *s* and is defined as:
$$\{pinfo(mv,(s;a))\}_s = \{md(mv);a\}_s.$$
8. The path information of a message *m(mv,(**p**;i;a;b))* sent from node *a* to node *b*, that was sent via path *(**p** ;i)* to *a*, and has been signed by *a* is defined as follows:
$$\{pinfo(mv,(\mathbf{p};i;a;b))\}_a = \{\{pinfo(mv,(\mathbf{p};i;a))\}_i;b\}_a$$
Notice that *i* has signed the identification of *a* (the receiver of the message). This is done in order to prevent *b* from undetectably removing *a* from the path (***p***;*i;a;b*).
9. A message *m(mv,(**p**;i;a))* that has travelled via path (***p***;*i*) from the source node to node *a* is defined as:
$$m(mv,(\mathbf{p};i;a)) = (\{mv\}_{(\mathbf{p};i)});(\{pinfo(mv,(\mathbf{p};i;a))\}_i)$$

Relaying message *m(mv,(**p**;i;a))* from node *a* to node *b*, results in message *m(mv,(**p**;i;a;b))* being sent to *b*, where:
$$m(mv,(\mathbf{p};i;a;b))=(\{\{mv\}_{(\mathbf{p};i)}\}_a);(\{\{pinfo(mv,(\mathbf{p};i;a))\}_i;b\}_a)$$

Thus, the message relay operation performed by *a* consists of signing $\{mv\}_{(\mathbf{p};i)}$ with *a*'s signature, adding node *b* to the signed path information $\{pinfo(mv,(\mathbf{p};i;a))\}_i$, signing the new path information $\{pinfo(mv,(\mathbf{p};i;a))\}_i;b$ with *a*'s signature, and sending the new message *m(mv,(**p**;i;a;b))* to *b*.

3.4. Efficiency considerations on the distributed encryption and decryption algorithms

The encryption (resp. decryption) algorithm described in 3.3.3. is started at a source node, and proceeds along several paths of nodes, each of which encrypts (resp. decrypts) the partially

encrypted data fragment with the new key fragments it possesses.The algorithm continues until the cryptographic function has been applied with all the $t^2 + t + 1$ key fragments of the cryptographic key. It is interesting to look at the length of the paths of nodes followed by the encryption (resp. decryption) algorithm. We will define a ***completed path*** as a path containing a set of nodes that together possess all key fragments of the cryptographic key. The algorithm thus consists of a number of completed paths.

Any completed path starts with a source node which possesses k different key fragments. In the worst case situation, every subsequent node in the path has only one new key fragment, and then, the length of the path (i.e. the number of nodes in the completed path) is $t^2 + 1$. This is an upper bound to the length of a completed path.

A lower bound to the length of a completed path is $t + 1$, since, from section 1, we know that at least $t + 1$ nodes are required to encrypt (resp. decrypt) data with any secret cryptographic key.

The algorithm should be designed in such a way that, in every situation of up to t arbitrarily faulty nodes, at least one completed path contains only correct nodes.

The duration of the algorithm depends on the length of the longest completed path followed by the algorithm. In section 3.4.1., we will show that it is possible to have the algorithm follow only completed paths of length t +1, and still guarantee that the correct encryption (resp. decryption) is performed in the presence of up to t maliciously functioning nodes.

Furthermore, the amount of data communication in the algorithm depends on the number of completed paths followed by the algorithm. In section 3.4.2., we will describe a simple method by means of which the number of completed paths followed by the algorithm remains small. However, as we will also show in section 3.4.2., this method does not always yield the minimal number of completed paths.

3.4.1. Distributed encryption and decryption algorithms with completed paths of minimal length only

Since the duration of the algorithm depends on the length of the longest completed path followed by the algorithm, it is advantageous to minimize the length of the longest completed path. However, the algorithm must always follow at least one completed path which contains correct nodes only.

We will show that, provided the number of faulty nodes does not exceed t, the set of all completed paths of length t +1 always contains a path with correct nodes only.

In the rest of this section, we use the following lemma:

Lemma 1:

Let S_1 , S_2, … S_{i+1} be disjunct sets. Let $\mathbf{S}$ be $\{S_1, \ldots , S_{i+1}\}$. Let M be the conjunction of S_1 through S_{i+1}, i.e. $M = S_1 \cup S_2 \cup \ldots \cup S_{i+1}$. Let X be a subset of M containing at most i elements. Then:

$$\exists S_j \in \mathbf{S}: j \in [1,i+1] \wedge S_j \cap X = \varnothing \qquad \text{❑}$$

The proof of this lemma is straightforward. The lemma will be used to prove the following theorem:

THEOREM 1:

Assume we have a system consisting of a set $\mathcal{N}$ of nodes, up to t of which may behave maliciously. For any i, with $1 \le i \le n$ (with $n = t \cdot (t+1)+1$) , let N_i be a node in set $\mathcal{N}$. For any j, with $1 \le j \le t \cdot (t+1)+1$, let f_j be a key fragment of cryptographic key f. With all key fragments f_j, with $1 \le j \le t \cdot (t+1)+1$, key f can be generated. We assume that A1 through A5 from section 3.3.1. are satisfied. We assume that our distributed encryption / decryption algorithm is performed by a group of n nodes and consists of applying a cryptographic function F with the secret cryptographic key f on data fragment B, by following only completed paths of minimal length $t+1$.

Then, at least one completed path of length $t+1$ contains correct nodes only. ❑

For simplicity, we first prove theorem 1 for the case t =2. After that, we generalize the proof for arbitrary t.

Proof for t =2:

For the case t =2, set $\mathcal{N}$= $\{N_1,N_2,N_3,N_4,N_5,N_6,N_7\}$. The key fragment distribution is given in table 1 in Section 3.2. The minimal length of a completed path is $t+1 = 3$. Now, we can distinguish three different disjunct sets S_1 , S_2, and S_3 of t (=2) nodes, each of which contains $t \cdot (t+1)$ (=6) key fragments. The sets and the corresponding key fragments possessed by them are given in table 2.

set	nodes in set	possessed key fragments
S_1	N_1 , N_4	f_1 , f_2 , f_3 , f_4 , f_6 , f_7
S_2	N_2 , N_5	f_1 , f_2 , f_3 , f_4 , f_5 , f_7
S_3	N_3 , N_6	f_1 , f_2 , f_3 , f_4 , f_5 , f_6

Table 2: Disjunct sets of t nodes, each with k-1 key fragments for the case t =2

For each of the sets S_i (with $1 \le i \le 3$), the key fragment not possessed by one of the nodes in the set S_i, is possessed by each of the nodes in the corresponding set U_i, where:

$$U_1 = \{N_5 , N_6 , N_7\}$$
$$U_2 = \{N_6 , N_7 , N_1\}$$
$$U_3 = \{N_7 , N_1 , N_2\}$$

E.g. key fragment f_5, which is not possessed by one of the nodes in S_1, is possessed by N_5, N_6, and N_7.

Since t =2, each of the sets U_1 through U_3 contains at least one correct node.

Now, we define X to be a subset of $\mathcal{N}$, containing all faulty nodes of $\mathcal{N}$. Since we assumed that the number of faulty nodes does not exceed t, set X contains at most t (=2) nodes. Then, by lemma 1, at least one of the sets S_1 through S_3 has an empty intersection with set X, i.e. at least one of the sets S_1 through S_3 contains correct nodes only.

So there is always a set S_j (with $1 \le j \le 3$) which does contain correct nodes only, and its corresponding set U_j always contains a correct node, say N_h (with $1 \le h \le 7$).

Now, any path following the nodes in S_j and node N_h is a completed path which has minimal length and contains correct nodes only. ❑

Proof for arbitrary *t*:

Now we will prove theorem 1 for arbitrary t. Set $\mathcal{N} = \{N_i \mid 1 \le i \le t \cdot (t+1)+1\}$. The key fragment distribution is given in Section 3.2. The minimal length of a completed path is t+1. Now, we can distinguish t+1 different disjunct sets S_i of t nodes, each of which contains $t \cdot (t+1)$ key fragments. Set S_i (with $1 \le i \le t+1$) is defined as follows:

$$S_i = \{N_j \mid j = i + h \cdot (t+1) \wedge 0 \le h \le t-1\}$$

Set S_i (with $1 \le i \le t$+1) contains all key fragments of key f, except the fragment f_{i+t^2}. This fragment is possessed by each of the t+1 nodes in the corresponding set U_i, where, for $1 \le i \le t$+1, U_i is defined by:

$$U_i = \{ N_j \mid j = (i + t^2 + h) \textbf{ mod } n \wedge 0 \le h \le t\}$$

Each of the sets U_i (with $1 \le i \le t$+1) contains at least one correct node.

Now, we define X to be a subset of $\mathcal{N}$, containing all faulty nodes of $\mathcal{N}$. Since we assumed that the number of faulty nodes does not exceed t, set X contains at most t nodes. Then, by lemma 1, at least one of the sets S_1 through S_{t+1} has an empty intersection with set X, i.e. at least one of the sets S_1 through S_{t+1} contains correct nodes only.

So there is always a set S_j (with $1 \le j \le t$+1) which does contain correct nodes only, and its corresponding set U_j always contains a correct node, say N_h (with $1 \le h \le t \cdot (t+1)+1$).

Now, any path following the nodes in S_j and node N_h is a completed path which has minimal length and contains correct nodes only. ❑

3.4.2. A method to reduce the number of completed paths in distributed encryption and decryption algorithms

In Theorem 1, we assumed that the distributed encryption / decryption algorithm may follow all completed paths of minimal length. In the algorithm, there are $(t+1)^2$ different sets of

t+1 nodes, for each of which the nodes can be followed in any order by the algorithm, resulting in a large number of completed paths with minimal length.

In this section, therefore, we investigate how the number of completed paths followed by the algorithm can be reduced. This reduces the amount of data communication needed by the algorithm. We present a simple method by means of which the number of paths followed by the algorithm remains small.

First of all, provided that all nodes in a certain path are functioning correctly, since we assumed that the encryption (resp decryption) function is commutative, the order in which the nodes in a path are followed by the algorithm, has no influence on the encryption (resp. decryption) result. Therefore, we reduce the number of paths followed by the algorithm, by *requiring that every path followed by the algorithm contains a unique set of nodes (i.e. different from the sets of nodes of other paths)*. This can easily be established e.g. by visiting the nodes in every path in a fixed order.

However, the number of paths followed by the algorithm can be reduced much further, as long as it is guaranteed that at least one path contains correct nodes only (in the presence of up to t faulty nodes).

We will use the following lemma:

Lemma 2:

Let $S_1, S_2, \ldots S_{i+1}$ be non-empty sets, which have exactly one element x in common, i.e.

$$\exists_1 x: (\forall\, j,k \in [1,i+1]: (j \neq k \Rightarrow S_j \cap S_k = x))$$

Let $M \supseteq S_1 \cup S_2 \cup \ldots \cup S_{i+1}$, and $\mathbf{S}$ be $\{S_1, \ldots, S_{i+1}\}$. Let Y be a subset of $M \setminus \{x\}$ containing at most i elements. Then:

$$\exists\, S_j \in \mathbf{S} : j \in [1,i+1] \wedge S_j \cap Y = \varnothing \qquad \text{❑}$$

The proof of lemma 2 is straightforward and follows directly from lemma 1.

We will now describe our method of reducing the number of completed paths followed by our distributed encryption / decryption algorithms. We introduce the method for the case t =2, and thereafter generalize it for arbitrary t.

Assume we have a system consisting of a set $\mathcal{N}$ of nodes, up to t of which may behave maliciously. For any i, with $1 \leq i \leq n$ (with $n = t \cdot (t+1)+1$) , let N_i be a node in set $\mathcal{N}$. For any j, with $1 \leq j \leq t \cdot (t+1)+1$, let f_j be a key fragment of cryptographic key f. With all key fragments f_j, with $1 \leq j \leq t \cdot (t+1)+1$, key f can be generated. We assume that A1 through A5 from section 3.3.1. are satisfied. We assume that our distributed encryption / decryption algorithm is performed by a group of n nodes and consists of applying a cryptographic function F with the secret cryptographic key f on data fragment B, by following only completed paths of minimal length t+1.

The reduction method for the case *t*=2

For *t*=2, all completed paths of minimal length, each one containing a unique set of nodes, are given in table 3.

Path	Description
P_1	$N_1 \rightarrow N_4 \rightarrow N_5$
P_2	$N_1 \rightarrow N_4 \rightarrow N_6$
P_3	$N_1 \rightarrow N_4 \rightarrow N_7$
P_4	$N_2 \rightarrow N_5 \rightarrow N_6$
P_5	$N_2 \rightarrow N_5 \rightarrow N_7$
P_6	$N_2 \rightarrow N_5 \rightarrow N_1$
P_7	$N_3 \rightarrow N_6 \rightarrow N_7$
P_8	$N_3 \rightarrow N_6 \rightarrow N_1$
P_9	$N_3 \rightarrow N_6 \rightarrow N_2$

Table 3: Description of all different completed paths of minimal length for t=2

We will show that with a selection of 6 completed paths from the set above, provided that $\mathcal{N}$ does not contain more than 2 faulty nodes, there is always at least one path with correct nodes only.

For any combination of up to 2 faulty nodes in $\mathcal{N}$, the algorithm must follow at least one path containing correct nodes only.

Initially, the paths P_3, P_5, and P_7 are included in the algorithm. For any i with $1 \leq i \leq 9$, we define set $S_{i,j}$ as the set containing the nodes of path P_i. Sets S_3, S_5, and S_7 have only one node (viz. N_7) in common. Let Y be the set of all faulty nodes in $\mathcal{N} \setminus \{N_7\}$. Since we assumed that t=2, Y does not contain more than two nodes. By lemma 2, at least one of the sets S_3, S_5 , and S_7 has an empty intersection with Y. So, provided that node N_7 functions correctly, at least one of these three paths contains correct nodes only.

We have to include other paths from table 3 in the algorithm in order to guarantee a path with only correct nodes in the case that N_7 is faulty. Notice that the algorithm already contains a path with only correct nodes, if node N_7 functions correctly. The paths we are going to include next need only be followed in the case that N_7 is faulty !

For this purpose, we select the paths P_2 and P_4. We assume that N_7 is faulty. Sets S_2 and S_4 (i.e. the sets containing the nodes in path P_2 resp. P_4) have only 1 node in common (viz. N_6). Let Y be the set of all faulty nodes in $\mathcal{N} \setminus \{N_6,N_7\}$. Since we assumed that t=2 and N_7 is faulty, Y does not contain more than 1 node. By lemma 2, at least one of the sets S_2 and S_4 has an empty intersection with Y. So, provided that node N_6 functions correctly (and that node N_7 is

faulty), at least one of these paths contains correct nodes only.

Now we have to include a final path from table 3 in the algorithm in order to guarantee the presence of a path with only correct nodes in the case that both N_6 and N_7 are faulty. For this purpose, path P_1 is selected.

Thus the final algorithm for $t = 2$ contains the paths P_1, P_2, P_3, P_4, P_5, and P_7.

It is worthwhile noticing that the described reduction method does **not** yield an optimal solution (i.e. a solution with a minimal number of completed paths of minimal length). For t =2 an optimal solution only requires 5 paths, e.g. a solution with paths P_1, P_3, P_5, P_7 and P_9 is also possible. The verification is left to the reader. It is, however, not straightforward to arrive at this solution.

The reduction method for arbitrary *t*

For arbitrary t, all $(t + 1)^2$ completed paths P_i (for $1 \le i \le (t + 1)^2$) of minimal length, each one containing a unique set of nodes, are defined as follows:

$\forall\ j,k \in [1,t+1]$:

$$P_{(j-1)\cdot(t+1)+k} \equiv N_j \rightarrow N_{j+(t+1)} \rightarrow N_{j+2\cdot(t+1)} \rightarrow \ldots \rightarrow N_{j+(t-1)\cdot(t+1)} \rightarrow N_{j+(t-1)\cdot(t+1)+k}$$

The selection of the paths for arbitrary t is done analoguously to the selection of paths for the case t=2. The selection of the paths is done in t+1 selection steps. In selection step i (for $1 \le i \le t+1$), t+2-i paths are selected.

In the first selection step, we select all t+1 paths in the set:

$$\{ P_{(j-1)\cdot(t+1)+k} \mid j+k = t+2 \wedge 1 \le j \le t+1 \}$$

By lemma 2, we can prove that, provided that node $N_{t\cdot(t+1)+1}$ functions correctly, at least one of these t+1 paths contains correct nodes only.

In selection step i (with $1 \le i \le t+1$), we select all t+2-i paths in the set:

$$\{ P_{(j-1)\cdot(t+1)+k} \mid j+k = t+3-i \wedge 1 \le j \le t+1 \wedge 1 \le k \le t+1 \}$$

The final algorithm for arbitrary t contains the paths in the set:

$$\{ P_{(j-1)\cdot(t+1)+k} \mid j+k = t+3-i \wedge 1 \le i \le t+1 \wedge 1 \le j \le t+1 \wedge 1 \le k \le t+1 \}$$

The number of paths followed by the algorithm is equal to:

$$\sum_{i=1}^{t+1} i$$

which is equal to $(t^2 + 3\cdot t + 2) / 2$.

3.4.3. Precomputation of composed key fragments

The algorithm described in 3.3.3. proceeds along several paths of nodes, each of which performs the encryption / decryption function on the partially encrypted data fragment with all

the key fragments it possesses. In RSA, sequential application of the cryptographic function F with x key fragments f_i through f_{i+x} on a data fragment B results in the calculation of:

$$(\ldots (((((B^{f_i})(\mathbf{mod}\ n))^{f_{i+1}})(\mathbf{mod}\ n))\ldots)^{f_{i+x}})\ (\mathbf{mod}\ n)$$

which is equal to

$$(B^{f_{res}})(\mathbf{mod}\ n)$$

where f_{res} is defined by:

$$f_{res} = (f_i \cdot f_{i+1} \cdot \ldots \cdot f_{i+x})(\ \mathbf{mod}\ \varphi(n))$$

If we assume that every correct node knows the paths along which the data fragments are sent, every correct node can precompute the resulting composed key fragments with which the data should be encrypted (resp. decrypted). In the implementation we made of the algorithm for the case $t = 2$, we obtained a reduction in the execution time of the algorithm of approximately 72%, by applying this method of precomputation of composed key fragments.

3.4.4. Implementation

As a part of the FADE project carried out on the University of Twente, the distributed encryption / decryption algorithm described in this paper has been implemented for the case $t = 2$ with 5 paths using precomputation of composed key fragments. The algorithm runs on a set of Sun workstations in a Unix environment.

4. Conclusion

In this paper, we have described distributed encryption and decryption algorithms. These algorithms are required in order to make a dependable distributed data storage system in which data is stored in an encrypted form resilient to up to t arbitrarily faulty nodes in the system.

The algorithms run on a system consisting of a number of nodes, each of which possesses a number of key fragments. Data fragments are encrypted / decrypted by relaying it along several paths of nodes, each of which performs the encryption / decryption function on the partially encrypted data fragment with all the new key fragments it possesses.

We have considered several optimizations of the algorithm. We have minimized the length of the paths that are followed by the algorithm. Furthermore, we presented a simple method in order to reduce the number of paths of minimal length that are followed by the algorithm. Finally, we have considered precomputation of composed key fragments, which may save considerable amounts of computation time. We have implemented the algorithm for the case t=2.

5. References

[1] Simmons, G.J., **Contemporary Cryptology, The Science of Information Integrity**, IEEE Press, New York, 1992.

[2] Diffie, W., and Hellman, M.E., New directions in cryptography, in: **IEEE Transactions on Information Theory**, IT 22(6), November 1976, pp. 644-650.

[3] Rivest, R., Shamir, A., and Adleman, L., A method for obtaining digital signatures and public-key

cryptosystems, in: **Communications of the ACM**, Vol.21, 1978, pp.120-126.

[4] Shamir, A., How to share a secret, in: **Communications of the ACM**, Vol.22, No.11, November 1979, pp.612-613.

[5] Gong, L., Increasing Availability and Security of an Authentication Service, in: **IEEE Journal on Selected Areas in Communications**, Vol. 11, No.5, June 1993, pp.657-662.

[6] Pease, M., Shostak, R., and Lamport, L., Reaching agreement in the presence of faults, in: **Journal of the ACM**, Vol.27, No.2, April 1980, pp.228-234.

[7] Lamport, L., Shostak, R. and Pease, M., The Byzantine Generals Problem, in: **ACM Transactions on Programming Languages and Systems**, Vol. 4, No. 3, July 1982, pp.382-401.

[8] Barborak, M., Malek, M., and Dahbura, A., The Consensus Problem in Fault-Tolerant Computing, in: **ACM Computing Surveys**, Vol. 25, No.2, June 1993.

[9] Niven, I., and Zuckerman, H.S., **An Introduction to the Theory of Numbers**, Wiley, New York, 1972.