IMPROVEMENT OF DATA PROCESSING SECURITY BY MEANS OF FAULT TOLERANCE

Gilles Trouessin* Yves Deswarte* Jean-Charles Fabre* Brian Randell**

* LAAS-CNRS & INRIA
7, Avenue du Colonel Roche
31077 Toulouse Cedex – France

** Computing Laboratory, The University
Newcastle upon Tyne
NE1 7RU – United Kingdom

Abstract

This paper discusses various different solutions to the problem of reliable processing of confidential information. One of the major difficulties of this problem comes from the fact that conventional techniques for achieving reliability, on the one hand, and security on the other, tend to be in opposition to each other. The different solutions presented here have been classified in three distinct types: two are related to classical security techniques (protection, and encryption) and the third is a new technique, the fragmentation-redundancy-scattering technique, which it is claimed demonstrates a potentially advantageous unified approach to the provision of reliability and security, based on fault tolerance. Finally, a qualitative comparison of these solutions is given, taking into account both dependability, openness and performance criteria.

Keywords: secure architectures, integrity, reliability/availability, protection, encryption, fragmentation.

1. Introduction

In this paper we concern ourselves with the provision of high reliability and availability, and the preservation of data confidentiality, in large scale distributed systems, such as ones based on workstations connected over one or more high speed LANs.

1.1. Problem statement

Dependability, a generic concept – defined as *the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers* – may be viewed w.r.t. different *properties* [8] and so enables the definition of a number of different dependability **attributes**, including: **availability** (w.r.t. *readiness for usage*), **reliability** (w.r.t. *continuity of service*), **safety** (w.r.t. *avoidance of catastrophic consequences on the environment*), **security** (w.r.t. *prevention of unauthorized access and/or handling of information, i.e., provision of data integrity and confidentiality*).

Some of these attributes (reliability/availability and security) are often considered separately because the techniques used to achieve them are usually perceived as being mutually antagonistic. Firstly, *reliability* and *availability* are generally achieved by incorporating mechanisms for tolerating any faults (especially accidental faults) that occur, or that remain despite attempts at fault prevention during the system design process. These techniques will of necessity involve space and/or time redundancy; they can easily take advantage of a distributed computing architecture by means of replicated computation using sets of untrusted[1] (or fallible) processors. Secondly, *security features* are generally achieved by means of fault prevention mechanisms (w.r.t. intentional faults, such as intrusions) whereby critical applications are implemented on physically and/or logically protected computers. Such protection is usually based on the *TCB* (Trusted Computing Base) or *NTCB* (Network Trusted Computing Base) concepts [17] [18].

From the above we see that what can be termed an antagonism between reliability/availability and security arises in at least the two following ways [7]: (i) *accidental-fault tolerance* (by means of replication) increases the number of potential access points to confidential information and thus can reduce the effectiveness of the protection techniques; (ii) *intrusion prevention* (by means of a local *TCB* or a *NTCB* partition) can suffer from the fact that one cannot justifiably rely either on a single *TCB* (which forms a classical "single point of failure"), or on the local *TCB*/*NTCB* partition of each computer inter-connected to the network.

To be adequately realistic, a solution dealing with this antagonism must, we believe, take into account the following two requirements: (i) *trusted area reduction,* by which we mean that the security provided by a potential

[1] Here we use the term *trusted* component to mean one that is assumed to be highly reliable and available, and impervious to intrusions (i.e., not to be a source of deliberate faults), in its intended environment.

solution should depend on as small as possible a *trusted area*, because it is impossible to place confidence on all of the processors in the network; (ii) *openness,* by which we mean that a potential solution must not be (excessively) software and/or hardware dependent, but instead allow implementation over a network of heterogeneous systems. The former requirement, regarding *trusted area reduction*, would also contribute to the latter one, regarding *openness*, since untrusted processors belonging to same critical application would thus not be security-dependent.

1.2. Reliable processing of confidential information

Let us consider the problem of *reliable processing of confidential information* as involving a combination of the three following features *a)*, *b)* and *c)*:

a) Simple processing (Fig. 1a): processing *(P)* is applied to a set of input data *(D)* in order to obtain a set of output results *(R)*. Both areas are shaded to denote the fact that neither the processor (the lower area) nor the environment containing the I/O (input/output) devices and which provides the inputs and accepts the outputs (the upper area) are trusted.

b) Reliable processing (Fig. 1b): the redundant execution of *P* (by means of processor replication) in order to provide data integrity for *D* and *R*, and reliable processing of *P*.

c) Confidential processing (Fig. 1c): in this, and indeed all cases where confidentiality is required, input is provided from, and output is delivered back to, a trusted area. Neither of the two regions of Fig. 1c is shaded; this is to indicate that *P* is executed, and its I/O prepared/received, securely (in similarly trusted areas), in order to preserve the confidentiality of *D*, *R* and (perhaps) *P*.

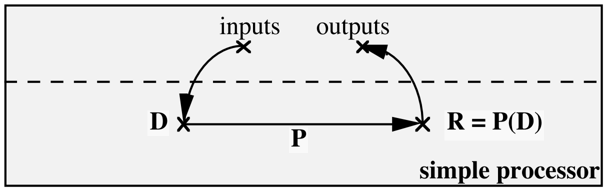

Figure 1a: Simple processing

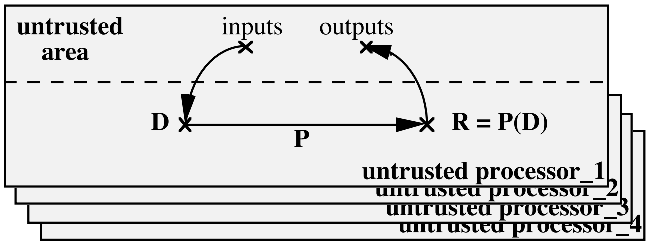

Figure 1b: Reliable processing

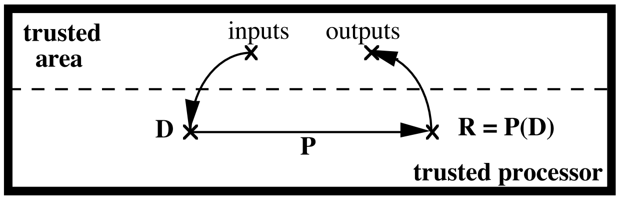

Figure 1c: Confidential processing

2. Achieving Combined Reliability and Security

2.1. Approach 1: Protection

This first approach is based on a classical security technique, *protection*, an intrusion-prevention technique which is based on forecasting and preventing, as far as possible, the different intrusions that could damage overall system security. This technique may be implemented by either of two different solutions: *1) Centralized protection* or *2) Local protection.* In each case, replication is also employed, in order to add both processing reliability and data integrity.

2.1.1. Solution 1.1: Centralized protection and replication

This first solution (Fig. 2) is in fact the logical combination of the features (reliability and confidentiality) represented in Figs. 1b and 1c. As in all cases where confidentiality is required, I/O operations, for each given user, are performed in a trusted area using a similarly trusted processor. However, the processing is replicated and executed by trusted processors that all belong to the same trusted area as that where the data is provided and the results received by the user. Solution 1.1 can be developed on the basis of a centralized *TCB* as recommended in the *Orange Book* [17], using a specific architecture, i.e., a fault-tolerant computer system, such as Tandem or Stratus systems.

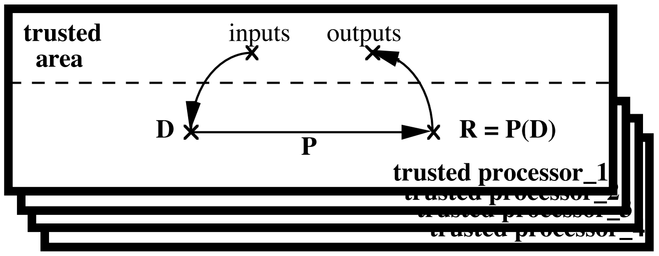

Figure 2: Solution 1.1 – Centralized protection and replication

There are two possibilities for preserving the confidentiality of data whilst it traverses the medium used for inter-processor communication, depending on whether or not the medium is considered as part of the trusted area. In the latter case, confidentiality preservation of the whole critical application is based on the encrypting of all communications between processors. In either case, the *trusted area reduction requirement*, and thus also the *openness requirement*, are not adequately met, since all the processors (together, one can be sure, with significant portions of their operating systems), and perhaps the communication medium, are considered as part of the trusted area.

Solution 1.1 is thus in practice perhaps well suited for very specific highly critical applications such as some types of military computation but does not fit well with general-purpose applications which may invoke remote processors and use several distinct networks.

2.1.2. Solution 1.2: Local protection and replication

This second solution (Fig. 3) is in fact a network generalization of the previous solution. I/O operations are performed in a trusted area located on a special trusted processor. Normal processing is still replicated but it is now accomplished in an untrusted area, on processors which are in general untrusted. Each of these processors is however protected by means of a local *TCB* and a *NTCB* partition as recommended in the *Red Book* [18].

An *Authentication—Access Control* scheme (*AAC* and *AAC'*, in Fig. 3) is needed between the special trusted processor and the other processors involved in the critical application, in order to ensure the overall security of the application.

There is only one possibility for the preservation of the confidentiality of the communication medium since processing is executed in the untrusted area: the medium must be considered as part of the untrusted area and all communications between the different processors must thus be encrypted.

Several hardware and/or software implementations of this solution have been developed, for example: the *Distributed Secure System* [2] [11], the *LOCK co-processor* [12] in connection with the *SDNS* project [15], *Secure Sun OS* [16].

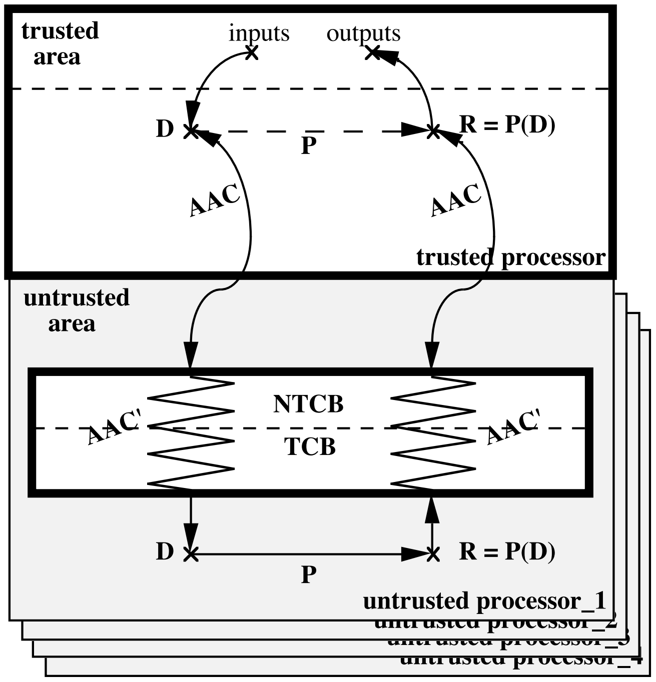

Figure 3: Solution 1.2 – Local protection and replication

With Solution 1.2, we can see that the *trusted area reduction requirement* is partially respected: (i) *respected*, since each processor involved in executing the critical application is now considered to be untrusted, and to be situated in an untrusted environment area; (ii) *partially*, because each of these untrusted processors must be protected by a local *TCB* and *NTCB* component, which are each in fact a local (albeit perhaps small) trusted software and/or hardware mechanism operating in an untrusted environment. However, this means that the mechanism should therefore be made tamper-proof, so that it cannot be opened without destroying its content. Such tamper-proof devices also need to possess a master key in order to communicate securely with other such devices in the untrusted area; and in practice, must be small and essentially maintenance-free.

The other requirement, openness, is not respected because each implementation of this solution requires the help of a *TCB*/*NTCB* partition to enforce security on the different processors of the network. This is the main drawback of Solution 1.2, particularly where the *TCB* or *NTCB* component is merely software running on the otherwise untrusted processor, because it is very difficult to protect the component from and by something as complex as an operating system, (e.g., Unix in the *LOCK*/*ix* project). However when the *NTCB* is in special-purpose hardware, monitoring all communications to and from the untrusted processor (as in the *DSS* project), its task, and that of making it tamper-proof, are more readily achievable. Anyway, in all cases, if any of the local *TCB*/*NTCB* components are corrupted or replaced by Trojan Horses, all the others are threatened so that the security of the whole network is compromised and the confidentiality of the critical application lost.

2.2. Approach 2: Encryption

This second approach is based on another classical security technique, *encryption*, which is a well established technique for preserving the confidentiality of communications and file archiving. It can be used for preserving the confidentiality of information processing in two different ways: *1)□omomorphic encryption* or *2)□lack-box encryption*. In each case, replication is also used, in order to provide both processing reliability and data integrity.

2.2.1. Solution 2.1: Homomorphic encryption and replication

With this solution (Fig. 4), a user's I/O operations are as always performed in a trusted area, and reliability features are obtained by means of processing replication, again in an untrusted area, but in a encrypted way. In the one trusted area, a special trusted processor transforms the data set *(D)* and the processing *(P)* by means of a specific kind of encryption technique *(C)* into an encrypted data set *(D')* and encrypted processing *(P')*.

Only certain types of encryption, called *privacy homomorphisms* [1] [10] [13], are suitable for such transformations. However, when *C* is of such a type, *P'* can be securely accomplished in the untrusted area, by untrusted processors. Encrypted results *(R')* obtained in the untrusted area can then be de-crypted *(C^{-1})* in the trusted area to obtain results in clear *(R)*:

- *D* thus has an image *D'* according to *C*: *D'=C(D)*;
- *P* also has its own "image" *P'* depending on both *C* and *P* features: *P' is a function of (C, P)*;
- *R'* is thus an image of *D'* with *P'*: *R'=P'(D')*.

With Solution 2.1 communication confidentiality is directly preserved by means of encryption and no additional techniques are required for this purpose.

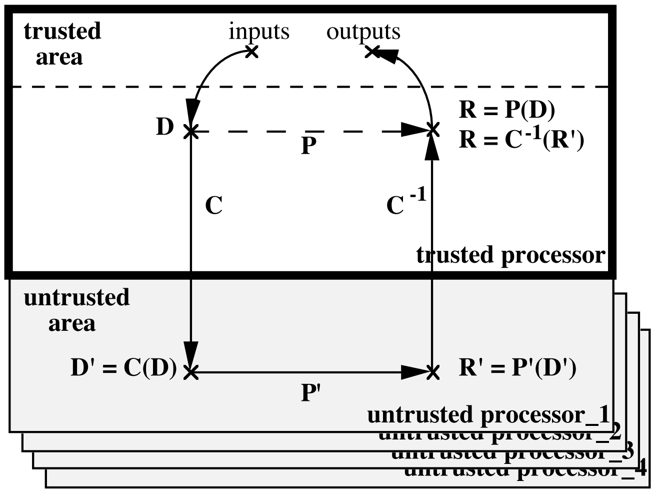

Figure 4: Solution 2.1 –
Homomorphic encryption and replication

But a restriction must be observed in implementing any scheme based on Solution 2.1. If an intruder can access the encrypted value of any arbitrary constant and if the comparison operator is available then usage of a privacy homomorphism is no longer secure. This is because the intruder can use a simple binary search strategy to discover the encrypted value of each data item of the whole data set *D* [10]. However in some particular cases (where there is no need for a comparison operator) Solution 2.1 is valid [1] [13]; but these cases are very limited (very specific banking transactions, for example) and thus this approach cannot be considered as providing a general solution.

Because of the above restriction, we can say that Solution 2.1 partially respects the *openness requirement* since processing is securely executed only in some particular cases (if *C* is a privacy homomorphism and if *P* does not provide the comparison operator). However, we can say that Solution 2.1 respects the *trusted area reduction requirement* perfectly, since processing is completely executed by untrusted processors, without any need for trusted devices in the untrusted area.

2.2.2. Solution 2.2: Black-box encryption and replication

This solution (Fig. 5) exhibits some common features with the previous solution: I/O operations are performed in the trusted area and processing in the untrusted area, reliability is obtained by replication and confidentiality by encryption. However, homomorphic encryption is replaced by *black-box encryption*. In fact, processing is apparently executed in encrypted form: *R'=P'(D')*, since only encrypted data *D'*, encrypted processing *P'* and encrypted results *R'* can be observed in the untrusted area.

In reality, processing is executed in clear inside a trusted "black box" associated with each untrusted processor. This solution involves three steps:

- encrypted input data *(D')* is received and de-crypted with *C^{-1}*: *$D=C^{-1}(D')$*;
- normal processing *P* is executed in order to obtain results *R*: *R=P(D)*;
- results *R* are encrypted with *C* in order to obtain *R'* that can be sent out of the black box securely: *R'=C(R)*.

The trusted black box thus contains a decrypting-processor, a small size memory, a processor and an encrypting-processor. To be really secure, it must be tamper-proof (as described in Section 2.1.2 above).

However, Solution 2.2 suffers from several major drawbacks [7], though the first three listed below are essentially similar to those possessed by Solution 1.2:

- *protection against a Trojan-horse black box:* in order to be qualified as *trusted*, it must not be possible to replace the black box by a *Trojan-horse black box* during its operational life (leave alone during initial installation);
- *management of encrypted addresses:* all data received by the trusted black box, such as addresses, are encrypted and are thus more difficult to decode and use;
- *management of communication keys:* one (or several) master cryptographic key(s) is(are) required in order to allow secure communications, which increases the management complexity of key distribution and use;
- *increase of local memory space:* for management of encrypted addresses or communication keys and local data storage, thus increasing the local memory space required for the black box whereas it ideally should, as mentioned previously, be small.

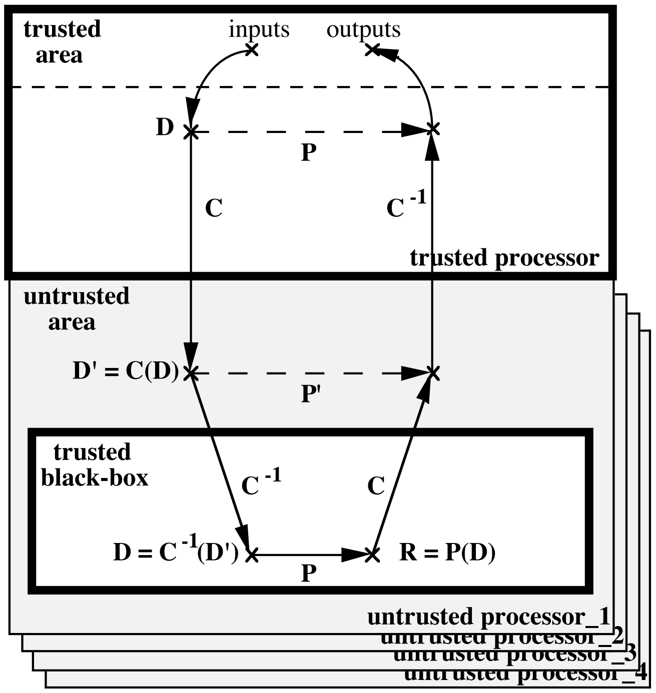

Figure 5: Solution 2.2 –
Black-box encryption and replication

Because of these drawbacks, we can say that though Solution 2.2 is feasible, like Solution 1.2 it does not meet the *openness requirement*, because the security in the untrusted area is really hardware- and software-dependent (i.e. black box dependent), and it does not meet the *trusted area reduction requirement* very well, since a trusted device (the trusted black box) must be installed essentially in each processor.

2.3. Approach 3: Fragmentation-Scattering

This third approach is based on what can be termed a "unified fault tolerance" technique, the *Fragmentation-Redundancy-Scattering (FRS)* technique, since it provides, in a single mechanism, means of tolerating both accidental and intentional faults, and hence of providing both reliability *and* confidentiality of data and its processing. Fragmentation involves defining fragments of information so as to ensure that, once isolated into physically separate processors, each fragment is of little value to a potential intruder due to the lack of significant information content in any one processor. (In principle such fragmentation can either be achieved at the programming language level, where it can take advantage of programmer-defined data structuring, or at the operating system level, where it is based on machine-level data types, such as bytes, words and/or pages. Particularly in the former case there is the possibility of requiring, and making use of, programmer-supplied constraints indicating which data items it would be especially undesirable for an intruder to be able to correlate.) Such fragments are then replicated, and the replicated fragments scattered across a (preferably large) number of processors.

FRS has been developed and successfully demonstrated in the context of a secure file archiving system [5] [6] and in the course of research into security management [3] [4]. In the processing context [7] [19] the approach relies on the correct execution of a majority of a set of copies of each of a number of program fragments with their corresponding data fragments, these fragments being widely distributed across a number of untrusted processors. Research to date on the application of *FRS* to processing[1] has resulted in the devising of two rather different implementation schemes: *1) fragmentation-scattering and replication* or *2) fragmentation-scattering and threshold.*

[1] The *FRS* technique applied to processing is called *Fragmented Data Processing (FDP)*. Some actual examples of this *FDP* technique are presented in the Appendix to this paper.

2.3.1. Solution 3.1: Fragmentation-scattering and replication

With this solution (Fig. 6), I/O operations are again performed in the trusted area, reliability features are again obtained by means of processing replication in the untrusted area, but in a fragmented fashion. In the trusted area, the trusted processor transforms the data set *(D)* by means of a set of projections, or data-fragmentation functions, $F = \{f_1, f_2, \ldots, f_n\}$, into a set $D = \{d_1, d_2, \ldots, d_n\}$ of data fragments. Similarly, processing *(P)* is transformed by means of a set of projections, or program-code fragmentation functions, $G = \{g_1, g_2, \ldots, g_n\}$, into a set $P = \{p_1, p_2, \ldots, p_n\}$ of program-code fragments. A critical application is thus split into n distinct program fragments, each of which consists of a data fragment d_i and a program-code fragment p_i, as follows:

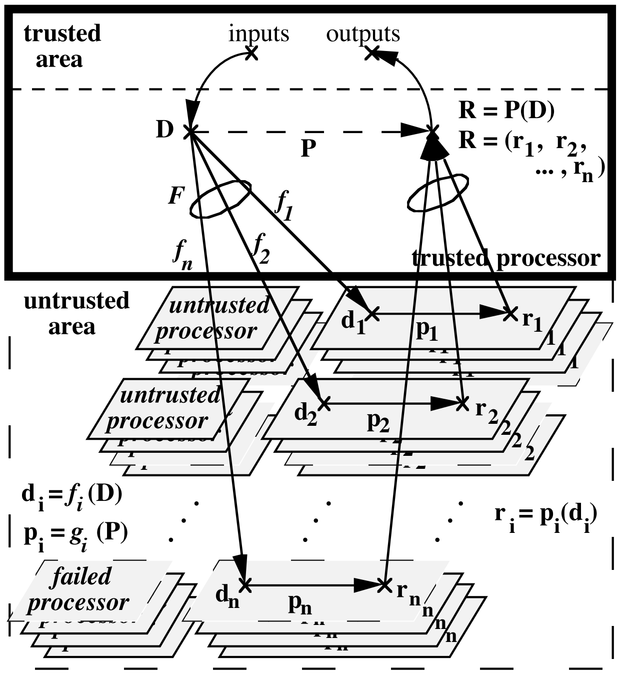

Figure 6: Solution 3.1 – Fragmentation-scattering and replication

- d_i is the image of D by projection f_i: $d_i = f_i(D)$;
- p_i is the image of P by projection g_i: $d_i = g_i(D)$;
- r_i is the image of d_i by processing p_i: $r_i = p_i(d_i)$.

Results R can only be reassembled on the trusted processor because each untrusted processor does not have enough information to permit such re-assembly: perhaps just a single program fragment (one data-fragment d_i, one program code-fragment p_i) and thus one result fragment r_i, for a given application. In practice, several program fragments could however be mapped on the same physical processor, provided that they do not in sum reveal any significant information.

Solution 3.1 possesses two main beneficial features:

- *different classes of fragmentation functions (f_i and g_i) can be defined:* security depends on the way the fragmentation functions (f_i and g_i) are chosen: for a given critical application different classes of fragmentation functions are possible (data and/or program-code driven) and different fragmentation strategies are also possible, thus allowing different security features to be obtained (data and/or program-code confidentiality preservation) [19] [20];
- *security does not depend on f_i or g_i confidentiality:* security does not rely on a potential intruder being ignorant of the semantics of the fragmentation functions (f_i or g_i).

A potentially major drawback of Solution 3.1 is that it might be expensive, in terms both of performance and program development effort. The major issues involved are as follows:

- *additional memory space overheads:* the memory space overheads due to replication are exactly the same as for any other solution using replication; those due to fragmentation come from the fact that there can be an overlapping of the data fragments d_i derived from D. In such a case and if these overheads are important, then another fragmentation strategy (based on a larger, but then admittedly perhaps less secure, fragmentation granularity) might be adopted;
- *increased number of processors:* this is unavoidable, but in the introduction of this paper it was indicated that we are assuming the basic global environment of the problem of *reliable processing of confidential information* in a distributed computing environment. In many such systems, for example university computing networks, a large number of processors can be idle very often [9]; in such cases the provision of reliability/availability by means of processing replication and of security by means of fragmentation-scattering can together take advantage of such unused processing power;
- *communication overheads:* since many more processors are required than with the two previous approaches, much more communication traffic may be induced; for example by data fragment exchanges between distinct program fragments. In such cases, and if communication overheads are significant, this again, might motivate the adoption of another fragmentation strategy, with larger fragmentation granularity;

- *development effort:* if significant development effort is needed to apply the technique to each separate application this would constitute the major cost of this approach, which would probably only be justifiable on very critical applications which were thereafter to receive extensive use. To date, investigations have been somewhat application-specific, but the prospect of application-independent methods of using the technique is now being considered.

From the above, and from experiments to date, we claim that solution 3.1 respects the *openness requirement*; we believe that most types of critical application can be fragmented, at least to a degree, largely because a wide range of fragmentation possibilities are offered by means of the different fragmentation classes and strategies. The *trusted area reduction requirement* is certainly respected since no trusted software and/or hardware components need be situated in untrusted areas. In particular, the fragmentation functions (f_i and g_i), though they are used in the trusted area and are the basis of security, are not confidential and could actually be held in untrusted areas.

2.3.2. Solution 3.2: Fragmentation-scattering and threshold schemes

This solution (Fig. 7) has some points in common with the previous one: I/O operations are performed in the trusted area, processing in the untrusted area, and confidentiality requirements obtained by means of fragmentation-scattering. But reliability/availability are now obtained by using so-called threshold schemes [14] applied to processing, instead of using processing replication. With the threshold technique, at least *s* (the threshold number) shadows of a given secret are needed to reconstitute the secret and less than *s* shadows do not give any information about this secret. Like error correcting codes, the threshold scheme technique thus imposes some redundancy in order to tolerate accidental but also intentional faults (especially intrusions w.r.t. data confidentiality and integrity), and during processing.

In the trusted area, the special trusted processor transforms the data set *(D)* by means of a specific set of projections, taking into account the threshold scheme, $F^s = \{f_1^s, f_2^s, \dots, f_m^s\}$, into a set of data fragments, $D^s = \{d_1^s, d_2^s, \dots, d_m^s\}$. Similarly, processing *(P)* is transformed by means of a specific set of projections, $G^s = \{g_1^s, g_2^s, \dots, g_m^s\}$, into a set of program-code fragments: $P^s = \{p_1^s, p_2^s, \dots, p_m^s\}$. A critical application is thus split into $m > n$ distinct program fragments (each consisting of a data fragment d_i^s and a program-code fragment p_i^s) as follows:

- d_i^s is the image of D by projection f_i^s: $d_i^s = f_i^s(D)$;
- p_i^s is the image of P by projection g_i^s: $d_i^s = g_i^s(D)$;
- r_i^s is thus the image of d_i^s by p_i^s: $r_i^s = p_i^s(d_i^s)$.

As with Solution 3.1, *R* can only be reconstituted on the trusted processor because each untrusted processor does not have enough information, possessing in principal just one data and one program-code fragment for a given application.

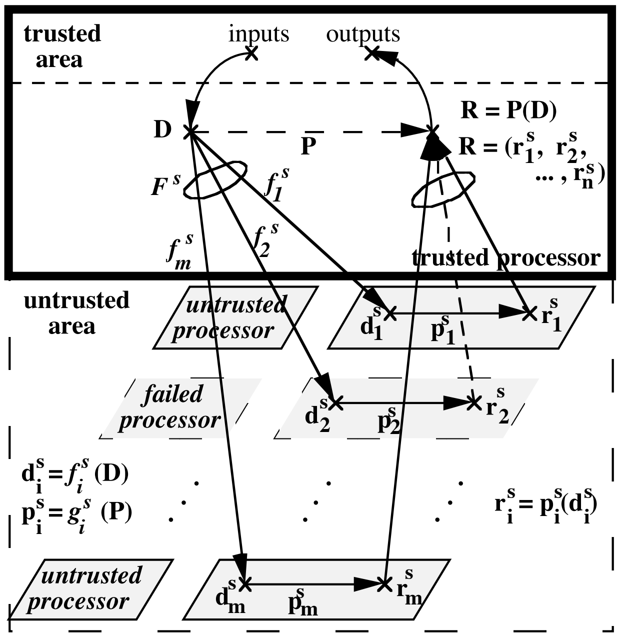

Figure 7: Solution 3.2 – Fragmentation-scattering and threshold schemes

Solution 3.2 possesses many of the advantages and disadvantages of Solution 3.1 - however, it allows reduction of the number of required processors to *m*, instead of the *n.r* needed by the previous solution (if *r* is the replication level) and thus reduces various types of overhead, including communication overheads. But its main drawback is that not all threshold schema are suitable. With Shamir's threshold schema based on polynomials [14] for example, a restricted set of polynomials must be chosen: addition is conserved with this kind of processing, but multiplication becomes quickly expensive (high degree polynomials are required in order to perform multiplications securely) and comparison can be used only if the restricted polynomial set offers the total order property (any shadow of any secret must be comparable and respect the total order property imposed by the secrets).

Thus we can see that this solution meets the *trusted area reduction requirement* fully and that just a restricted set of threshold schemes can be used (high degree and ordered polynomials). Two main problems must also be considered:

(i) the security of the fragmentation projections (F^S and G^S) must remain as strong as the (proved) security of the threshold scheme technique; (ii) the cost of these fragmentation projections, i.e., in terms of development effort and required execution time, must not be too important compared to the execution time of the program-code fragments (P^S).

3. Qualitative Comparison

Different comparison criteria must be considered, *dependability*, *openness* and *performance criteria*, to provide a qualitative comparison (Table 1) of the different solutions presented in the previous section.

3.1. Dependability, openness and performance criteria

As explained in the introduction of this paper, *reliable processing of confidential information* is assumed to be concerned with different dependability criteria *(dc)*, i.e., goals and requirements, and one openness criterion *(oc)*:

- *dc_1* – **reliability:** data processing reliability (and availability) and data integrity;
- *dc_2* – **confidentiality preservation (security):** preservation of data (and perhaps processing) confidentiality;
- *dc_3* – **trusted area reduction:** non excessive security-dependence on trusted areas;
- *oc_1* – **openness:** non excessive security-dependence on specific software or hardware.

Five performance criteria *(pc)* are considered, since valid solutions must not involve excessive performance overheads:

- *pc_1* – **number of processors:** the number of processors required, not counting those needed for redundancy purposes (see *pc_2*);
- *pc_2* – **redundancy overheads:** the number of processors required for redundancy purposes;
- *pc_3* – **number of messages:** the number of messages induced by the total set of processors;
- *pc_4* – **memory size:** the local (to each processor) memory space required for the solution implementation;
- *pc_5* – **system connectivity:** the number of inter-connection links required between the different processors.

As yet we do not have extensive experimental data concerning the performance and costs of *FRS*, when applied as here to processing, as distinct from file archiving and security management. Therefore Table 1 gives our subjective comparison of the six distinct solutions presented in this paper, taking into account the above dependability and performance criteria. Five values, corresponding respectively to the following qualitative scale, are used to characterize the extent to which the different criteria are satisfied:

- the five values are: −−, −, =, + and ++;
- the corresponding qualitative scale is: *very unfavourable*, *unfavourable*, *no influence*, *favourable* and *very favourable*.

Table 1:
Qualitative comparison of the different solutions with respect to dependability, openness and performance criteria

	dependability			open-ness	performance				
	rel.	security							
	dc_1	*dc_2*	*dc_3*	*oc_1*	*pc_1*	*pc_2*	*pc_3*	*pc_4*	*pc_5*
	reliability/avail-ability/integrity	confidentiality preservation	trusted area reduction	openness	number of processors	redundancy overheads	number of messages	memory size	system connectivity
Approach 1: **protection** *Sol. 1.1:* centralized protection + replication	=	−	−−	−−	++	=	=	=	=
Sol. 1.2: local protection + replication	=	−	=	−	++	=	=	=	=
Approach 2: **encryption** *Sol. 2.1:* homomorphic encryption + replication	=	−	++	=	+	=	=	=	=
Sol. 2.2: black-box encryption + replication	=	−	=	−	+	=	=	=	=

Approach 3: **fragmentation-scattering** *Sol. 3.1:* fragmentation-scattering + replication	++	++	++	++	−−	=	−−	+	−
Sol. 3.2: fragmentation-scattering + threshold schemes	+	+	++	++	=	+	=	+	=

3.2. Comparison results

Table 1 shows that in our opinion Approach 3 leads to better results w.r.t. the dependability (and openness) criteria: this is due to the fact that it is a unified concept providing both accidental- and intentional-fault tolerance. Approach 1 and Approach 2 globally present better results w.r.t. performance criteria *pc_1*, *pc_2*, and *pc_3*; this is due to the fact individual processing replicas (*D*, *P* and *R*) are not split over different processors, as is the case with Approach 3.

It should be noted that local (but not global) memory overheads are not expected to be significant with Approach 3; this is because more processors are required for almost the same global memory occupation and each processor then needs a smaller local memory space.

From this analysis and the judgements expressed in the Table, we conclude that all of the approaches described here are in principle capable of providing solutions, suitable for at least some situations, to the problem of *reliable processing of confidential information*. Each approach, however, has some weaknesses:

- *Approach 1:* poor reduction of the trusted area, and either poor security openness (Solution 1.1) or poor confidentiality preservation (Solution 1.2, when based on OS-enforced software partitioning);
- *Approach 2:* poor confidentiality preservation, and either poor security openness (Solution 2.1: operator restriction) or poor reduction of the trusted area (Solution 2.2);
- *Approach 3:* though adequate w.r.t. our dependability requirements this approach is somewhat poor w.r.t. performance, a situation which we believe can be ameliorated by exploiting the large range of possible fragmentation strategies already identified for this new technique.

4. Concluding Remarks

The originality and potential attractiveness of the fragmentation-scattering approach is that it is provides a unified means of achieving both reliability and security. At this stage, much remains to be discovered about its real advantages and disadvantages, w.r.t. the pre-existing combined techniques against which we have compared it, but we believe the analysis presented above shows that it has considerable promise as a new means of providing fault tolerance against both accidental faults and intentional faults such as intrusions.

Acknowledgements

The authors wish to thank very much Jean-Claude Laprie and David Powell for their earlier basic ideas about the *Fragmentation-Redundancy-Scattering* approach for ensuring high reliability/availability and security, Jean-Michel Fray for his contribution to the concept of *Fragmented Data Processing* for reliable processing of confidential information, and the reviewers of the submitted version of this paper for their constructive comments. This work was partially supported by DRET under convention no. 88.34.051.00.470.75.01 and the PDCS (Predictably Dependable Computing Systems) project no. 3092 of the European ESPRIT programme.

References

[1] N. Ahituv, Y. Lapid, S. Neumann, *Processing Encrypted Data*, Comm. of the ACM, Vol. 30(9), Sept. 1987, pp. 777-780.

[2] D.H. Barnes, R. MacDonald. *A Practical Distributed Secure System*, Proc. NEOS'85 (Networks and Electronic Office Systems), London – United Kingdom, IERE, 1985, pp. 83-91.

[3] L. Blain, Y. Deswarte, *An Intrusion-Tolerant Security Server for an Open Distributed Computing System*, Proc. European Symposium On Research In Computer Security, ESORICS'90, Toulouse – France, Oct.□4-26, 1990, Pub. AFCET, ISBN 2-9036778-9, pp.□7-104.

[4] Y. Deswarte, L. Blain, J.-C. Fabre, *Intrusion Tolerance in Distributed System*, Proc. 1991 IEEE Symposium on Research in Security and Privacy, Oakland – California, May□0-22, 1991, pp.□10-121.

[5] J.-M. Fray, Y. Deswarte, D. Powell, *Intrusion Tolerance Using Fine-Grain Fragmentation-Scattering*, Proc. 1986 IEEE Symposium on Security and Privacy, Oakland – California, April□-9, 1986, pp.□94-201.

[6] J.-M. Fray, Y. Deswarte, D. Powell, *A Secure Distributed File System based on Intrusion Tolerance*, Proc. 4th International Conference on Computer Security, IFIP/Sec'86, Monte Carlo, Dec. 2-4, 1986, pp. 407-416.

[7] J.-M. Fray, J.-C. Fabre, *Fragmented Data Processing: an Approach to Secure and Reliable Processing in Distributed Computing Systems*, Proc. 1st IFIP International Working Conference on Dependable Computing for Critical Applications, Santa Barbara - California, Aug. 23-25, 1989, pp. 131-137, published in *Dependable Computing for Critical Applications*, Ed. A. Aviz;ˇ ienis, J.-C. Laprie, Springer Verlag 1991, Vol. 4, ISBN 3-211-82249-6 & 0-387-82249-6, pp. 323-343.

[8] J.-C. Laprie, *A Unifying Concept for Reliable Computing and Fault-Tolerance*, in *Dependability of Resilient Computers*, Ed. T. Anderson, Blackwell Scientific Publications, 1989.

[9] D.A. Nichols, *Using Idle Nodes in a Shared Computing Environment*, Proc. 11th ACM Symposium on Operating Systems Principles, Austin – Texas, Nov. 1987, pp. 5-12.

[10] R.L. Rivest, L. Adleman, M. Dertouzos, *On Data Banks and Privacy Homomorphisms*, in *Foundations of Secure Computation*, Ed. R.A. Demillo, D.D. Dobkin, A.K. Jones, R.J. Lipton, Academic Press, 1978, pp. 169-179.

[11] J.M. Rushby, B. Randell, *A Distributed Secure System,* IEEE Computer, Vol. 16(7), 1983, pp. 55-67.

[12] S.O. Saydjari, J.M. Beckman, J.R. Leaman, *Locking Computers Securely*, Proc. 10th National Computer Security Conference, Sept. 1987, pp. 129-141.

[13] J. Seberry, J. Pieprzyk, *Cryptography: An Introduction to Computer Security*, Ed. R.P. Brent, Prentice Hall, 1989, 375 p.

[14] A. Shamir, *How to share a secret*, Communications of the ACM, Vol. 22(11), Nov. 1979, pp. 612-613.

[15] E.R. Sheehan, *Access Control within SDNS*, Proc. 10th National Computer Security Conf., Sept. 1987, pp. 165-171.

[16] B. Taylor, D. Goldberg, *Secure Networking in the Sun Environment*, USENIX Association Summer Conference Proc., Atlanta – Georgia, 1986, pp. 28-37.

[17] TCSEC, Department of Defense Standard, *Trusted Computer System Evaluation Criteria*, Department Of Defense, DOD 5200.28-STD, Dec. 1985, 121 p.

[18] TNI, National Computer Security Center, *Trusted Network Interpretation of TCSEC*, National Computer Security Center, NCSC.TG.005, July 1987, 278 p.

[19] G. Trouessin, J.-C. Fabre, Y. Deswarte, *Reliable Processing of Confidential Information*, Proc. 7th International Conference on Information Security, IFIP/Sec'91, Brighton – United Kingdom, May 15-17, 1991, pp. 210-221.

[20] G. Trouessin, *Quantitative Evaluation of Confidentiality by Entropy Calculation*, Proc. 4th Computer Security Foundations Workshop, Franconia – New Hampshire, June 18-20, 1991.

Appendix

Depending on the way that fragmentation is performed, it is possible to define different classes of fragmentation techniques, which are concisely described below and more detailed in [19]. In addition, fragmentation granularity is another parameter that can be considered in the choice of a given fragmentation class.

The first class of fragmentation technique relies on fine grain fragmentation (bit or small group of bits), and is in effect a sort of bit-slicing technique: each data item within the whole data structure is split into f fragments of b bits. Thus each of the individual processors has only a local view of the data structure, i.e., b bits out of $f.b$ bits. By this means, the global value of each data item and thus its semantics can be effectively hidden. The code must then be transformed into a set of code “fragments”, each of which has been modified so as to work appropriately with the corresponding fragmented data. Actually, this is not really code fragmentation since the code is simply slightly transformed, and the original program's semantics is unlikely to be effectively disguised from an intruder. The main interest of this class of fragmentation technique is that it allows a quantitative evaluation of confidentiality preservation by means of entropy calculation [20], but its main drawback, due to its restriction to fine granularity only, is that the *openness requirement* is not respected.

A second class can be defined when applied at the module level. Each program fragment (data- and code-fragment) corresponds to a single entity in the whole program structure (an instruction block, a program module or a library function) delimited by breaks in the code sequence. Each such fragment is then replicated in r distinct copies scattered over the network. Actually, this class of fragmentation technique is opposite to the previous one because the code is first fragmented in connection with its structure (this is particularly interesting in the case of modular programming languages or block-structured languages) and then the whole data structure is consistently fragmented (with respect to the first code fragmentation). This means that each code fragment will be associated with its own local variables and this implies also that global variables must be fragmented and shared by several distinct fragments.

From the above two classes of fragmentation technique we can derive a third one. As with the first class, this third class takes into account the data structure in order to apply a first step of fragmentation. This first step is well suited to the preservation of data confidentiality, since its aim is to cut some of the semantic links in the tree that could otherwise be built with the main semantic links of the data structure. And secondly, as with the second class, it is applied at the module level and thus similarly relies on the program structure, so is a technique which is well suited to preservation of code confidentiality. This third class thus is used at the programming language level, and is a compromise between fragmentation at the variable and program module levels. If fine grain fragmentation is required for efficient confidentiality preservation then data and code fragmentation can be applied in alternation to get an overall fragmentation which depends on both data and code structures.