DEPENDABLE SECURITY BY TWISTED SECRET SHARING

Semir Daskapan
Faculty of TBM, Delft University of Technology

Abstract: Large scale networked information systems are referred to as critical information infrastructures when they provide critical services to the critical physical infrastructures. Critical information infrastructures contain specific nodes that provide security services, like authentication servers; those are called security or trust centres. The goal of this research is to find an algorithm for enabling those centres to become sustainable by sharing their (superfluous) security resources and to resist Byzantine failures. The proposed secret sharing algorithm takes care of allocating in advance the secret content of the suffering centre to other healthy centres, so that only an arbitrary majority of them can reconstruct the content. This perfect t,n- threshold scheme is suitable in dynamic networks as it has an adaptive access structure. It is compared to existing schemes rather simple as it is purely based on permutations. It is efficient, i.e. favourable information rate, as all shares are much shorter than the secret itself. Secondarily, each secret share is even additionally protected (encrypted) against the holder as well against any outsider.

Key words: Trust, dependability, critical infrastructures, secret sharing, threshold cryptography

1. INTRODUCTION

An infrastructure consists of many dependent components, which enable together a service as a carrier. However, a dependable infrastructure contains usually one or more centralized nodes for certain control functions, like a centralized repository of metadata or an authentication server. Here the focus is on security distribution centers (SDCs). Any configuration that exhibits (multiple) single point of failures (SPF's) is avoided as much as possible. In

Y. Deswarte et al. (eds.), *Security and Protection in Information Processing Systems*

order to resist failures, that cause discontinuity, those centralized nodes are set up in redundancy. The traditional approach of investing in redundancy by dedicated hardware has its limitation in scalability and costs.

Complementary to these traditional approaches, where each weak node is enriched with a few identical nodes, a resource sharing approach between all the critical nodes is already introduced in [1]. The objective of that is autonomous collaboration of all the nodes to back up each other when failures occur by offering their resources as temporary hosts. Destitute nodes will send their valuable secret assets to the allied nodes (pool members) with healthy and free redundant resources. However, whether this resource-sharing concept will also work in critical information infrastructures (i.e. distributed information systems) depends on the additional requirements of forward and backward confidentiality, but most on the timing of releasing the secret assets and their keys to the allied pool members. By applying a secret sharing algorithm it should be possible to distribute the secret assets without revealing the content prematurely. The reason is that no member will receive the complete secret asset, but only a fragment. Even if he manages to get all the fragments he will not have all the keys to decrypt the fragments, because of the uncorresponding fragments and keys. It was therefore baptised as *twisted secret sharing*. Only by collaborating with a majority of pool members the secret asset can be reconstructed.

In the second section other works are discussed. In the third section a guiding case is presented for the problem, by which a pattern is sought for the algorithm. This pattern is then introduced and discussed in the fourth section. Finally, in section 5 the paper is finalized by a conclusion.

2. OTHER WORKS

Survivability is traditionally enhanced by means of redundancy [2,3,4]. Redundancy means that in parallel similar systems are ready to replace the main system. From a security point of view, redundancy seems to decrease the vulnerability as it avoids (multiple) single points of failure (MSPFs). With many of such replicas, the system can withstand (number of replicas - 1) failures. However, with the increasing number of redundant systems, the number of targets and thus the possibility of confiscation increases. A typical form of that is the Byzantine failure [5] that occurs due to corrupted processors. As more systems provide the same functionality the danger is that the redundant systems suffer from asynchrony and unequal information, due to technical implementations.

Survivability is regarded as one of the attributes of dependability next to other aspects like reliability, availability and safety [6]. Critical information infrastructures (CIS) are designed according to high availability and reliability requirements. Ideally, this means that critical infrastructures should be available, even when the supporting hardware of the CIS is failing. Survivability of the core functionality of a system is often the main focus of research and the survivability of other dependency attributes are not considered. Although those services appear to decrease the vulnerability of the system, they are themselves not survivable and could therefore jeopardize the whole system when they fail. Therefore additional measures are taken for those overall services. Usually this is realized by applying in parallel redundant hardware, which will catch up the breakdown of the primary CIS. Besides that it is not very cost-effective, as redundancy is in economical terms an indispensable overhead, it is also insufficient due to the limited dedicated hardware.

One of the earliest works on survivable SDC's is the work of Gong [7], although specific for authentication servers. Here fault tolerance is achieved by collaboration of many SDC's in a group. Also the RAMPART system [8] provides secure communication in the presence of actively malicious processes and Byzantine failures for SDC's (authentication services) specific. Here, with the assumption of an asynchronous network up to a third of the members in a Rampart group may behave in Byzantine manner yet the group would still provide reliable multicast facilities. The system uses secure channels between two members of the protocol to provide authenticity. Protocols depend greatly on the *consensus* of processors to reach agreement on the course of action. Other works in this specific field are [9,10,11], which all deploy static redundancy with replicated services (srrs).

MAFTIA [12], SINTRA [13], COCA [14] and Ensemble [15] provide approaches for tolerating both accidental (Byzantine) faults and malicious attacks in large-scale distributed systems like on the Internet. This enables them to remain operational during attack, without requiring time-consuming and potentially error-prone human intervention. They assume asynchronous networks and use many updated techniques like, randomization by coin-tossing scheme [16] to overcome the impossible existence of a deterministic protocol [17]. The resilience is guaranteed for $t > n/3$ dependable members [18].

Those works also apply somehow (combinatorial) secret sharing [19,20] or threshold cryptography [21,22]. In distributed systems secrets play a special role, as they can bind the scattered components. A *secret* means some piece of information, be it a password, the private key of a public key cryptosystem, a solution to some mathematical problem or a set of

credentials. With secret sharing, a single secret is divided into *w* parts and distributed across *n* nodes. To reconstruct the whole secret, one node must be able to collect or derive all the *w* pieces of the secret. The so-called t,n-threshold schemes have a specific access structure, i.e. the specification of participants which are authorized to reconstruct the secret. In that case at least t<n are able together to reconstruct the secret, where t<n is the threshold. Usually *t* is larger than *n/2* without gossip or larger than *2n/3* with gossip among the total nodes.

The twisted secret sharing algorithm that is proposed in this paper has a unique combination of characteristics, which have not been found in other works [19,23,24,25]. The *perfect* t,n- threshold scheme is suitable in dynamic networks as it has an *adaptive access structure*. It is compared to existing schemes rather *simple* as it is purely based on transpositions. It is *efficient* as all shares are much shorter than the secret itself and have thus a favorable information rate [26]. Each secret share is additionally *protected* (encrypted) against the holder as well against any outsider. Furthermore, whereas other works require numerical values this algorithm also applies easily for *alphanumerical values*. Due to proactive refreshing of keys and shares it is *unpredictable*.

3. TWISTED SECRET SHARING

In this section we will describe some scenarios to explain the problem and provide a corresponding solution to each scenario. This section describes as such the inductive phase to the general solution as is proposed in the next section 4.

3.1 The guiding case

Assume that in a pool of *p* leaders one leader suddenly collapses. Consider also the possibility that other arbitrary pool members (i.e. remaining minority) are also malicious. Each pool member has to decide on how to distribute his secret asset (token) and the belonging secret keys contrarily, such that his assets can be resurrected safely by the honest majority, despite the malicious minority in the pool.

The strategy of the survivability concept is to update replicas of the token frequently and to send them precautionary encrypted to other well functioning pool members without the decryption keys. After that the destitute SDC collapses one of those trust pool members receives the decryption keys and recovers the token so that he can continue the service.

The token has to be sent separately from the keys to uncorresponding locations before the collapse.

After the collapse the individual pool members sense the disability of the one particular SDC. They send subsequently their keys to exactly one predefined successor among them, so that he can recover the token. With the containing secrets he can prove to the clients to be the trusted leader. The clients will probably not notice the change of leadership because of this smooth masking. Those steps will be explained as follows.

Let T *be a set of* t *tokens, with* $T = \{t_1, t_2, ..., t_t\}$ *and*
K *a set of* k *keys, with* $K = \{k_1, k_2, ..., k_k\}$ *and*
L *a set of* l *pool members, with* $L = \{l_1, l_2, ..., l_l\}$
Let TL *denote the allocation of* T *to* L, *with*
$TL = \{(t_1, l_1), .., (t_t, l_x)\}$ *and*
KL *denote the allocation of* K *to* L, *with*
$KL = \{(k_1, l_1), .., (k_k, l_y)\}$ *and*
$TKL = TL \cup KL$ *and* $\{t,k,l,x,y\} \in N$ *with* $\{x,y\} \leq l$

Step1. The destitute SDC sends frequently the encrypted tokens and the belonging keys before the collapse to many pool SDCs, such that the destination of the tokens does not correspond with the one of the keys.

$TKL = \{.., (t_t, l_x; k_k, l_y)\}, \quad with\ x \neq y.$

Step 2. The receiving many pool SDCs send subsequently the tokens and the keys after the collapse to exactly one predefined healthy pool SDC.

$TKL = \{.., (t_t, l_x; k_k, l_y)_i\}, \quad with\ x = y = successor$

The result of this strategy is that each receiving pool member has a token, which he himself cannot open with the received key, so that no pool member can act malicious individually before the collapse. He has to find another pool member who has the key that belongs to his encrypted token. This third member should then also be corrupted to betray the destitute SDC. An additional requirement is therefore that up to a minority of the pool members should not be able to recover the token and is thus allowed to be corrupted.

Any majority is then able to recover the token and is thus assumed to be trusted. Why not to construct a strategy where exactly all the members are required to reconstruct the token. The reason for not requiring all the pool members (i.e. consensus) to cooperate in the recovery process is the increased power of the (corrupted) individual. If there is just one malicious member, he can block the recovery by not sending the token before and the key after the collapse. If that is the case the successor receives all the keys from the pool members, except from the malicious member with the

corresponding key. The resurrection will not be completed and the clients of the collapsed SDC will become orphans.

3.2 An analogy: the sick old father

The question is how to achieve an allocation of pieces among the many pool members that meets the majority/minority and the uncorrsponding secret/key requirement in a most efficient way. With efficiency a minimum in the amount of interactions between the members and the minimum load per sent package is aimed. In order to explain this confusion by diffusion secret sharing algorithms an analogous example is worked out. From this an allocation matrix is derived that contains the keys and the destinations of the loads (secrets). The depicted pattern of the matrices of different scenarios will help to derive a general approach by induction.

Assume a sick old father who trusts nobody but himself and some of his sons. He has a treasure to leave behind and 3 sons (p1,p2,p3) waiting for his death and their inheritance. He has buried previously the treasure somewhere in the woods and drawn a map of the location. As he has not seen his sons for a long time, his problem is that he does not know which son(s) to trust. He cannot believe that all his descendants turned out to be bad and as such he is sure that the majority of them can be trusted. If he entrusts the map to one of his sons, he might risk that this particular son appears to be unreliable. This son might thus run away with the treasure before the father dies or even after he dies, so that the other (good?) sons inherit nothing. The same could happen when he entrusts his greedy wife, which he already put out of his testament.

Therefore he decides to tear the map in 3 fragments (v1,v2, v3) and to put each piece in a locked case (c1,c2,c3). To confuse his sons, copies of some fragments (v1',v2') are also put in the locked cases. Now we have more pieces of the fragments (c1(v2), c2(v3), c3(v1), c1(v1'), c3(v2')). Each son is respectively given one or more cases and exactly one uncorresponding distinguished key ((k1,k2,..)), which cannot open the cases the particular son has been given. The old man has thought this out well, but now he is wondering how many pieces to cut, which piece to put in each case and how to distribute the cases among all the sons so that exactly the minimum majority of the sons (i.e. the trusted ones = 2) would be able to reconstruct the map when they share cases and keys after his death?

To phrase it more formally: What is the optimal relation (with minimal cases, fragments and copies) between v, k and p if a minimum minority will be able to recover the map? Lets workout some cases.

The case with p = 3

With p = 3 sons the minimum majority is (min(q >3/2)=) 2. When applying the splitting method the secret T should be split in v pieces and distributed among the three, such that at least any two sons are required to reconstruct the secret. To phrase it otherwise, no maximum minority (i.e. 1 son) should be able to recover the secret T.

To solve v an inductive approach with increased efforts can be used starting with v =1. If v =1, then T is not split. The question is now how to distribute the one secret, such that no single son but at least two of them can recover it? If the father makes three copies (i.e. for each son one) of the secret and puts each of them in three different cases (with three different keys), then each son receives exactly one case and one key that does not correspond with the case. So each son has one decision variable to disturb: either one secret piece (per case) or one key to exchange. In this situation exactly at least two cooperating sons only can reconstruct the secret, so that for p = 3 the right v =1.

The case with p = 4

With p = 4 sons the minimum majority is (i.e. min(q > 4/2)=) 3. When applying the splitting method the secret T should be split in v pieces and distributed among the 4 sons such that at least any 3 sons are required to reconstruct the secret. To phrase it other wisely, no maximum minority (i.e. ≤ 2 sons) should be able to recover the secret T.

If v =1, then T is not split. The question is again how to distribute the one secret, such that at least three of them can recover it? This is not possible, as has been shown already in previous case that two sons are already able to recover a secret if not split.

If v =2, then T is split in two pieces. The question is now how to distribute the two pieces, such that at least three sons are needed to recover the main secret? Assume that the father shares a distinguished symmetric key with each son. The father has 2 decision variables for each son now: the two pieces. A brute force approach, in which all combinations are subsequently tried, results in tables 1 to 4.

All the corresponding options of symmetric keys $\{K_{0,1} ..K_{0,4}\}$ between father L_0 and p sons $\{L_1 ..L_2\}$ and pieces (T1,T2) are first depicted in table 1. A random search has been conducted and in table 2 the options are given to allocate those 2 pieces (in locked cases) to the p uncorresponding sons. In table 3 a configuration is given in which a minimal number of cases is distributed to achieve the requirements. It is obvious from table 3 that by one or by cooperation of two sons the main secret cannot be recovered. Reconstruction by any three sons is however possible. Thus for this case p = 4 and v = 2 meets the requirements. A solution with p = 4 and v>2 is also

possible, but as then more pieces have to be distributed, it id considered less efficient and as such it is not preferred above v = 2. In table 4 only the distinct indexes are shown providing clearness. Testing whether the allocation meets the requirements is easy by hiding any minority of columns and then to check whether the rest will be able to reconstruct T.

Table 1

L_0	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$	$K_{0,4}$
T1	$K_{0,1}$,T1	$K_{0,2}$,T1	$K_{0,3}$,T1	$K_{0,4}$,T1
T2	$K_{0,1}$,T2	$K_{0,2}$,T2	$K_{0,3}$,T2	$K_{0,4}$,T2

Table 2

L_0	L_1; $K_{0,1}$	L_2; $K_{0,2}$	L_3; $K_{0,3}$	L_4; $K_{0,4}$
T1	$K_{0,2}$,T1	$K_{0,1}$,T1	$K_{0,4}$,T1	$K_{0,3}$,T1
T2	$K_{0,4}$,T2	$K_{0,3}$,T2	$K_{0,2}$,T2	$K_{0,1}$,T2

Table 3

L_0	L_1; $K_{0,1}$	L_2; $K_{0,2}$	L_3; $K_{0,3}$	L_4; $K_{0,4}$
T1	$K_{0,2}$,T1		$K_{0,4}$,T1	
T2		$K_{0,3}$,T2		$K_{0,1}$,T2

Table 4

L_0	L_1	L_2	L_3	L_4
T1	2		4	
T2		3		1

The case with p = 5

For p = 5 solutions with v < 3 are neglected, because those do not meet the majority requirement as in the previous sections is discussed. However, applying the above described approach leads to an exponential search algorithm, which is computationally not achievable (NP). Considering that the keys and secrets frequently have to be refreshed and that each refreshment of the pieces requires a reallocation of them, it would make the application practically infeasible. Therefore this approach is facilitated by identifying and adopting a pattern of sequences in v, k and p, by which a polynomial solution domain is achieved.

4. THE PROPOSAL

In the previous section it was shown how to achieve the right distribution of secret shares by random searching, which is computationally infeasible with large p. In this section a random search is avoided by adopting a distribution pattern for the shares, which depicts the relation between p,k and

v. A sequence of key numbers is assumed from the upper left corner to the under right corner of the table as is depicted in table 5.

4.1 The pattern

Consider again p =3 and v =1 in tabel 5a, with CE_0 as the suffering father. The sequence starts with (L1,T1) = 2 or any other number which is not equal to 1 (due to the uncorrespondingness criterium). For frequent refreshment, each time the keys have to be (re)allocated the row rotates so that 231 becomes 123 and then 312 etc. Note that 123 will not be suitable due to the uncorrespondingness criterium. Each son (L) now needs to cooperate with another son to get his key, which results in this case in a majority of 2 (see for one example the shadowed column).

With 4 sons the majority is 3. This number excludes the option of not splitting the token (v =1), as is shown that an unsplit token can be reconstructed by a minority of two sons. So, for this reason the token is split in two pieces T1 and T2 and two rows will fill in the table. The horizontal sequence of the first row (T1) again starts with (L1,T1) = 2 and increases further. The horizontal sequence of the second share T2 (row) starts exactly with the next number of the vertical sequence of column L1 (L1,T2) = 3 and increases further by one. This double sequence (horizontal and vertical) can be extended for any n. In table 5 this linear approach is depicted for p = 3,5,7. Refreshment is achieved by rotating the table (both rows) horizontally.

Table 5 v = p/2 –1/2

T/L	L_1	L_2	L_3
T1	2	3	1
T2			
T3			

a. p=3, v=1

L_1	L_2	L_3	L_4	L_5
2	3	4	5	1
3	4	5	1	2

b. p=5, v=2

L_1	L_2	L_3	L_4	L_5	L_6	L_7
2	3	4	5	6	7	1
3	4	5	6	7	1	2
4	5	6	7	1	2	3

c. p=7, v=3

This approach is computationally feasible, but is not optimal for even p numbers, as can be seen in the next tables 6 where for the cases p = 4,6,8 the same goal is achieved but with less pieces to distribute. For even numbers p a different pattern is adopted. The saving compared to the odd numbers is (p*v)/2 = (2v*v)/2 = v^2 so that for p = 4 and v = 2 not 8 pieces are distributed, like in table 4, but 2^2 = 4 pieces. This counts not only for p = 4,6,8, but for all even numbers. The explanation for this saving is that for even numbers a roundup by 1 is done to achieve majority (p/2+1), whereas a majority is already achieved by the first rational number larger than p/2. The

smuggle of +1 leads to redundancy in the number of distributed pieces. The odd numbers also smuggle, but up (+1/2) and down (-1/2), so that the net result is zero (honest).

Table 6 . v = p/2

T/L	L_1	L_2	L_3	L_4
T1	2		4	
T2		1		3
T3				
T4				

a. p= 4, v= 2

L_1	L_2	L_3	L_4	L_5	L_6
2		4		6	
	5		1		3
6		2		4	

b. p= 6, v= 3

L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8
2		4		6		8	
	5		7		1		3
6		8		2		4	
	1		3		5		7

c. p= 8, v= 4

4.2 The algorithm in pseudo code

The next list in pseudo code shows how to make such allocation tables. The pseudo code comprehends two main parts: for an even and for an odd number of pool members. They manage the key allocation according to the pattern of the previous tables 5 and 6. The even part resolves three contentious issues: 1) the hop from row to row of the sequence of keys, 2) the increase of sequential key number due to that hop and 3) continuously resetting the limited sequence. The odd part adds a fourth issue: alternation of blank spaces except for the first cell in a row.

```
ShiftO = 0                    ' Initialisation of the shifts
ShiftE = -1
Period = 600 sec              ' setting an example timer to refill the allocation tables
Do each Period
  Blank = false
  Count(p, mypool)            ' p = number of pool members (sons)
  If p = Odd then
    key = ShiftO              ' key = allocated key = content of cell in table
    v = n/2 –1/2                                  ' v = number of pieces
    AllocateKeyPiece = ARRAY[1..v,1..p]           '
    For a = 1 to v do                             ' fill in rows from left to right
      key = key + 2                    ' jump two numbers when a new row starts
      For b = 1 to p do
        If b=! 1 then key = key + 1    ' setting the content of a cell
        If key > p then key = key-p    ' reset key sequence
        AllocateKeyPiece[a,b] = key
      End For
```

```
    End For
    ShiftO = ShiftO +1                    'counter for rotating the rows each period

  Else
    key = ShiftE
    v = n/2
    AllocateKeyPiece = ARRAY [1..v,1..p]
    For a =1 to v do                      ' when a new row starts:
      key = key +3                        ' jump three numbers
      For b = 1 to p do
        If b=! 1 then key = key + 1       ' setting a key
        If key > p then key = key-p       ' reset key counter when all keys are set
        If Blank = false then             'alternating blank cells
          AllocateKeyPiece [a,b]  = key   'allocate key to cell= piece and location
          If b=! p then Blank = true      'block alternation when new row starts
        Else
          If b=! p then Blank = false
        Endif
      End For
    End For
    ShiftE = ShiftE +1

     End if
  End do
```

With the allocation table a leader can determine by which keys to encrypt the pieces and where to sent them. In order to commit the logistical operation efficiently the pieces with the same destination have to be clustered and sent together. The tables with indexes of key numbers are used to encrypt the shares first. This happens by sequentially reading the records from the table and replacing the indexes with the encrypted piece (eventually in a new table).

```
AllocateEncrPiece = ARRAY [1..v,1..p]
AllocateKeyPiece = ARRAY [1..v,1..p]
AllocateEncrPiece [ ] = 0
    For a =1 to v do
      For b =1 to p do
        If IsNotEmpty(AllocateKeyPiece [a,b]) then
          AllocateEncrPiece [a,b]) = E(k_{l,a}; t_{l,b})
      Else
          AllocateEncrPiece [a,b]) = Empty
```

```
      Endif
    End For
  End For
```

The logistical preparation procedure will be completed by packaging each column as one message to one destination (pool member).

```
Package = Array[id1 ..idp]
Package[] =0
For a =1 to p do
  Package[ida] = ""
  For b =1 to v do
    If IsNotEmpty(AllocateKeyPiece [a,b]) then
      Package[idb] = Package[idb] + ";"+AllocateEncrPiece[a,b])
    Endif
  End For
  ProcSend(Package[idb]; idb)
End For
```

The refreshment occurs usually by subsequently rotating the rows by one column as is shown in table 7 for n = 5. An exception occurs, when the keys in the first shift would correspond with the key owner. The rotation shifts then two columns at once.

Table 7. Refreshing by rotation

L_1	L_2	L_3	L_4	L_5	
2	3	4	5	1	Shift=1
3	4	5	1	2	

L_1	L_2	L_3	L_4	L_5	
3	4	5	1	2	Shift=2
4	5	1	2	3	

L_1	L_2	L_3	L_4	L_5	
4	5	1	2	3	Shift=3
5	1	2	3	4	

5. DISCUSSION

This paper proposes a scalable and efficient decentralized secret sharing algorithm (i.e. with no premature central facilitator) in order to avoid multiple single points of failures before and after a failure. It prevents premature exposition of the individual shares by the participants as they all host a decrypted share with a key of the others. The algorithm requires an

arbitrary majority of participants to reconstruct the total secret and prescribes the exact keys for encryption and distribution of shares. The proposed algorithm has been adopted in a framework called Medusa that is meant to enhance autonomous survivability of security distribution centers in large-scale distributed information systems, like grid systems. The Medusa protocol set is subsequently added to an arbitrary authentication server and tested in a simulated environment by a discrete event network simulator NS+ with C++/TCL on Linux. The test case was a basic failure test in which randomly one of the joint authentication servers deliberately was stopped to assess how the secret sharing algorithm performed. It was obvious that the larger the allied pool was the more survivable the authentication service was, due to many shares. The extra performance overhead due to increasing numbers of participants and thus the increased number of shares was acceptable in comparison to the survivability increase. The complexity of the secret sharing algorithm is $O(n^2)$, due to the double loops of 1 to v and 1 to p. As such they were computable in polynomial time. However, there is a trade off between survivability and processing time. With a growing number of participants in the pool survivability increases at the expense of the increasing processing time for secret sharing. As such medium sized pools are aimed and by connecting them via one or more pool members this trade off can be avoided. The secret sharing algorithm is performed for the limited number of pool members in each pool, whereas survivability is determined by the range of all the interconnected pools.

A second test is intended where a distributed denial of service attack (ping-of-death) in which random adversaries send en masse ping messages to the authentication server.

REFERENCES

[1] S. Daskapan, W.G. Vree, A. Ali Eldin, "Trust metrics for survivable security systems", *Proc. of the IEEE International Conference on Systems, Man & Cybernetics*, Washington, 2003.

[2] A.E. Barbour, A.S. Wojcik, "A General Constructive Approach to Fault-Tolerant Design Using Redundancy", *IEEE Transactions on Computers*, pp. 15-29,1989.

[3] Matti A. Hiltunen, Richard D. Schlichting, Carlos A. Ugarte, "Enhancing Survivability of Security Services Using Redundancy", *The International Conference on Dependable Systems and Networks*, Sweden, 2001.

[4] Bruno Dutertre, Valentin Crettaz, Victoria Stavridou, "Intrusion-Tolerant Enclaves", *IEEE Symposium on Security and Privacy*, California, 2002.

[5] L. Lamport, R. Shostak, M. Pease, "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems*, vol.4(3), pp.382-401, July 1982.

[6] Jean-Claude Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology," *15th International Symposium on Fault-Tolerant Computing*, pp. 2-11, 1985.

[7] L. Gong, "Increasing Availability and Security of an Authentication Service", *IEEE Journal on Selected Areas in Communications*, Vol 11(5), pp.657-662, 1993.

[8] M. Reiter, "Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart", *Proc. of 2nd ACM Conference on Computer and Communications Security*, pp. 68–80, 1994.

[9] Thomas Hardjono, Jennifer Seberry, "Replicating the Kuperee authentication server for increased security and reliability", *Australasian Conference on Information Security and Privacy*, pp.14-26, 1996.

[10] Matti A. Hiltunen, Richard D. Schlichting, Carlos A. Ugarte, "Building Survivable Services Using Redundancy and Adaptation", *IEEE Transactions on Computers*, Vol. 52(2), pp. 181-194, 2003.

[11] Vivek Pathak, Liviu Iftode, "Byzantine Fault Tolerant Authentication", Rutgers University, Department of Computer Science *Technical Report*, DCS-TR-492, June 2002.

[12] P. Veríssimo, NF Neves, M. Correia, "The middleware architecture of MAFTIA: A Blueprint", *Proc. of the IEEE Third Information Survivability Workshop*, Boston, pp. 24-26, 2000.

[13] C.Cachin, J. Poritz, "Secure Intrusion Tolerant Replication on the Internet", *Proc. of the 2002 International Conference on Dependable Systems and Networks*, Washington, pp. 167-176, 2002.

[14] Lidong Zhou, Fred B. Schneider, Robbert van Renesse, "COCA: A Secure Distributed On-line Certification Authority", *Technical Report TR2000-1828*, 2000.

[15] Ohad Rodeh, Kenneth P. Birman, Danny Dolev ,"The architecture and performance of security protocols in the ensemble group communication system: Using diamonds to guard the castle", *ACM Transactions on Information and System Security*, Vol. 4(3), pp.289 – 319, 2001.

[16] R. Canetti and T. Rabin, "Fast asynchronous Byzantine agreement with optimal resilience", *Proc. 25th Annual ACM Symposium on Theory of Computing*, pp. 42-51,1993.

[17] M. J. Fischer, N. A. Lynch, M. S. Paterson, "Impossibility of distributed consensus with one faulty process", *Journal of the ACM*, Vol. 32(2), pp.374-382, 1985.

[18] G. Bracha. "An asynchronous [(n -1)/3]-resilient consensus protocol", *Proc. of 3rd ACM Symposium on Principles of Distributed Computing*, pp. 154-162, 1984.

[19] A. Shamir, "How to Share a Secret", *Communications of the ACM*, Vol 22(11), pp.612-613, 1979.

[20] K. Kurosawa, K. Okada , H. Saido , D. Stinson, "New combinatorial bounds for authentication codes and key predistribution schemes", *Designs, Codes and Cryptography*, Vol 15(1), pp.87-100, 1998.

[21] Y. Desmedt, "Threshold cryptography", *European Transactions on Telecommunications*, Vol 5(4), pp. 449–457, 1994.

[22] J. Seberry, C. Charnes, J. Pieprzyk and R. Safavi-Naini, "Crypto Topics And Applications II", Algorithms *and Theory of Computation Handbook*, CRC Press, 1999.

[23] G. J. Simmons, "An Introduction to Shared Secret and/or Shared Control Schemes and their Application", *Contemporary Cryptology: The Science of Information Integrity*, G. J. Simmons (Ed.), IEEE Press, pp.441-497, 1991.

[24] G. R. Blakley, "Safeguarding cryptographic keys", *Proc. of American Federation of Information Processing Societies 1979 National Computer Conference*, Vol.48, pp. 313-317, 1979.

[25] C. Cachin, On-line secret sharing, in "Cryptography and Coding V", C. Boyd (ed.), *Lecture Notes in Computer Science*, Vol. 1025, pp. 190-198, 1996.

[26] E.F. Brickell, D.R. Stinson, "Some improved bounds on the information rate of perfect secret sharing schemes", *Journal of Cryptology*, Vol. 5(3), pp. 153-166, 1992.