Intrusion-Tolerant Protection for Critical Infrastructures

Alysson Neves Bessani
Paulo Sousa
Miguel Correia
Nuno Ferreira Neves
Paulo Verissimo

DI–FCUL TR–2007–8

April 2007

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Intrusion-Tolerant Protection for Critical Infrastructures

Alysson Neves Bessani Paulo Sousa Miguel Correia
Nuno Ferreira Neves Paulo Verissimo
LASIGE, Faculdade de Ciências da Universidade de Lisboa – Portugal[*]

Abstract

Today's critical infrastructures like the Power Grid are essentially physical processes controlled by computers connected by networks. They are usually as vulnerable as any other interconnected computer system, but their failure has a high socio-economic impact. The paper describes a new construct for the protection of these infrastructures, based on distributed algorithms and mechanisms implemented between a set of devices called CIS. CIS collectively ensure that incoming/outgoing traffic satisfies the security policy of an organization in the face of accidents and attacks. However, they are not simple firewalls but distributed protection devices based on a sophisticated access control model. Likewise, they seek perpetual unattended correct operation, so they are designed with intrusion-tolerant capabilities and hardened with proactive recovery. The paper discusses the rationale behind the use of CIS to improve the resilience of critical infrastructures and presents a design using logical replication based on virtual machines.

1 Introduction

Critical infrastructures like the power, water and gas distribution networks have a fundamental role in modern life. These infrastructures are essentially physical/mechanical processes controlled electronically. The control systems, usually called SCADA (Supervisory Control and Data Acquisition) or PCS (Process Control System), are composed by computers interconnected by computer networks [25, 40].

In recent years these systems evolved in several aspects that greatly increased their exposure to cyber-attacks coming from the Internet. Firstly, the computers, networks and protocols in those control systems are no longer proprietary but standard PCs and networks (e.g., wired and wireless Ethernet), and the protocols are often encapsulated on top of TCP/IP. Secondly, these networks are usually connected to the Internet indirectly through the corporate network or to other networks using modems and data links. Thirdly, several infrastructures are being interconnected creating a complexity that is hard to manage [38].

Therefore these infrastructures have a level of vulnerability similar to other systems connected to the Internet, but the socio-economic impact of their failure can be huge. This scenario, reinforced by several recent incidents [42, 8], is generating a great concern about the security of these infrastructures, especially at government level.

A reference architecture was recently proposed to protect critical infrastructures, in the context of the CRUTIAL[1] EU-IST project [40]. The whole infrastructure architecture is modeled as a WAN-of-LANs. This topology allows simple solutions to hard problems such as legacy control subnetworks, and interconnection of critical and non-critical traffic. Typically, a critical information infrastructure is formed by facilities, like power transformation substations or corporate offices, modeled as collections of LANs and interconnected by a wider-area network, modeled as a WAN, in the WAN-of-LANs model.

This architecture allows defining realms with different levels of trustworthiness. In this paper we are interested in the problem of protecting realms from one another, i.e., a LAN from another LAN or from the WAN. However, given the ease of defining LANs in today' IP architectures (e.g., through Virtual switched LANs), there is virtually no restriction to the level of granularity of protection domains, which can go down to a single host. In consequence, our model and architecture allow us to deal both with outsider threats (protecting a facility from the Internet) and insider threats (protecting a critical host from other hosts in the same physical facility, by locating them in different LANs).

Protection of LANs from the WAN or other LANs is

[*]This work was partially supported by the EC through project IST-2004-27513 (CRUTIAL) and NoE IST-4-026764-NOE (RESIST), and by the FCT through project POSI/EIA/60334/2004 (RITAS) and the Large-Scale Informatic Systems Laboratory (LaSIGE).

[1]Critical UTility InfrastructurAL Resilience: `http://crutial.cesiricerca.it`.

made by a device called *CRUTIAL Information Switch* (CIS). A CIS provides two basic services: the *Protection Service (PS)* and the *Communication Service (CS)*. The PS ensures that the incoming and outgoing traffic in/out of the LAN satisfies the security policy of the infrastructure. The CS supports secure communication between CIS and, ultimately, between LANs. The CS provides secure channels, reliable and atomic multicast primitives, and probabilistic gossip-based information diffusion between CIS. Although the CIS supports these two services, this paper presents only the protection service, not the communication service. Therefore, from now on “the CIS” means “the CIS *Protection Service*”. Figure 1 ilustrates the use of CIS protecting several LANs of a critical infrastructure.

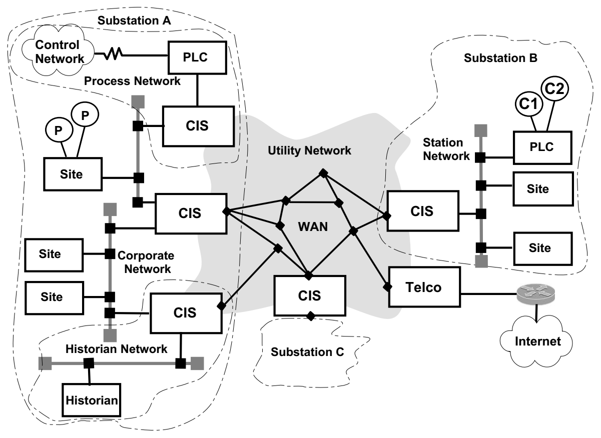


Figure 1. WAN-of-LANs connected by CIS.

A CIS can not be a simple firewall since that would put the infrastructure at most at the level of security of current Internet systems, which is not acceptable since intrusions in those systems are constantly being reported [19]. Instead, a CIS is a distributed protection device based on a sophisticated access control model. The main characteristics of the CIS are the following. Firstly, it has similarities to a *distributed firewall* [4], since CIS can be deployed not only on the network border but inside the networks to better protect critical equipment. Secondly, the CIS uses a *rich access control model* that takes into account the involvement of different organizations and allows the access control rules to depend on context information [1]. Thirdly, the CIS is *intrusion-tolerant* [17, 41], i.e., it operates correctly even if there are intrusions in some of its components and withstands a high degree of such hostility from the environment, seeking unattended perpetual operation [35].

The paper is essentially about the last topic, the design of an intrusion-tolerant CIS. The CIS is replicated in a set of machines. If there are intrusions in some of the replicas, the CIS masks these faults and follows its specification. This mechanism is hardened by proactively recovering replicas periodically [30], in order to avoid that an attacker successively corrupts replicas until it controls enough to corrupt the behavior of the CIS. This paper extends previous work on *proactive recovery* by using also *reactive recovery* to recover replicas that are detected to be compromised.

Several intrusion-tolerant services have been proposed in the literature, either based on Byzantine quorum systems (BQS) [26, 44] or state machine replication (SMR) [33, 10, 2]. However, the CIS design presents two very interesting challenges that make it essentially different from those services. The first is that a firewall-like component has to be transparent to protocols that pass through it, so it can not modify the protocols themselves to obtain intrusion tolerance. This also means that recipient nodes will ignore any internal CIS intrusion-tolerance mechanisms, and as such they can not protect themselves from messages forwarded by faulty replicas not satisfying the security policy. The paper shows that these two challenges are not trivial and presents a solution that is based neither on BQS nor on SMR.

As mentioned before, the initial motivation for developing the CIS was the protection of critical infrastructures. However, the CIS is sufficiently generic to allow its use in other scenarios with a high level of protection requirements.

The paper has the following main contributions: (1) it presents the design of an intrusion-tolerant firewall-like component adequate for protecting networked infrastructures and services from outsider and insider accidental or malicious threats, such as critical information infrastructures; (2) it proposes a novel *proactive-reactive recovery scheme and algorithm* that not only proactively recovers replicas but also forces recovery of replicas detected to be faulty; and (3) to our knowledge it presents the first implementation of an intrusion-tolerant service in a single machine, using logical replication based on virtual machines. The evaluation of this prototype has shown that the limited resources of a single machine constrain the performance of the service, but lead to an economic solution. However, more hardware-lavish (and more performant) solutions are not precluded, as it will be shown.

2 The CIS Protection Service

The CIS works mainly like a firewall. It captures packets that pass through it, checks if they satisfy the security policy being enforced, and forwards the approved packets, discarding those that do not satisfy the policy. However, several other characteristics of the CIS make it a unique protection device. These characteristics are presented in this section.

Distributed firewall. CIS can be used in a redundant way, enforcing the same policies in different points of the network. An extreme case in the SCADA/PCS side is to have a CIS in each gateway interconnecting each substation network, and a CIS specifically protecting each critical component of the SCADA/PCS network. The concept is akin to using firewalls to protect hosts instead of only network borders [4], and is specially useful for critical information infrastructures given their complexity and criticality, with many routes into the control network that can not be easily closed (e.g., Internet, dial-up modems, VPNs, wireless access points) [8].

Application-level firewall. Critical infrastructures have many legacy components that were designed without security in mind, thus do not employ security mechanisms like access control and cryptography [15]. Since these security mechanisms are not part of the SCADA/PCS protocols and systems, which *must still be protected*, protection must be deployed in some point between the infrastructure and the hosts that access it. The CIS has to inspect and evaluate the messages considering application-level semantics because, as already said, the application (infrastructure) itself does not verify it.

Rich access control model. Besides the capacity to inspect application-level messages, the CIS needs to support a rich access control policy that takes into account the multi-organizational nature of the critical infrastructures as well as their different operational states. Taking the Power System as an example, there are several companies involved in generation, transmission and distribution of energy, as well as regulatory agencies, and several of these parties can execute operations in the power grid. Moreover, almost all Power System operation is based on a classical state model of the grid [16]. In each state of this model, specific actions must be taken (e.g., actions defined in a defense plan, to avoid or recover from a power outage) and many of these actions are not allowed in other states (e.g., a generator can not be separated automatically when the Grid is in its normal state). These two complex facets of access control in critical infrastructures require more elaborated models than basic mandatory, discretionary or role-based access control. To deal with this, in the architecture of CRUTIAL we adopt a more elaborated model, OrBAC (Organization Based Access Control) [1]. It allows the specification of security policies containing permissions, prohibitions, obligations and recommendations, taking into account the role of the subject, who is part of an organization, the action it wants to execute, the target object of this action, and the context in which it is executed. An example: "In context 'emergency', operators from company C can execute maintenance operations on device D."

Intrusion-tolerant firewall. As discussed in the introduction, the level of security of current systems connected to the Internet is not adequate for the infrastructures we are concerned with, given their criticality. To improve the security and dependability of the CIS, it is designed to be intrusion-tolerant [41]: it is replicated in n machines and follows its specification as long as at most f of these machines are attacked and have their behavior corrupted. To enhance the resilience of the CIS, we employ proactive recovery to periodically rejuvenate the replicas, guaranteeing perpetual correct operation as long as no more than f replicas become faulty in a given recovery period [30]. These mechanisms can be integrated to a basic, non-replicated, design of the CIS in an incremental way. We can first build a non-replicated CIS, that works much like a firewall, and then add proactive recovery to it. After, a replicated version is built using the protocols described in this paper. The replicated version can be built using virtual machines in the same host or different physical machines. These two designs can be later extended to consider additional replicas and maintain the system liveness even when some replicas are recovering. All these different designs are discussed in more detail in Section 3.7.

In this paper, we address the problem of designing an intrusion-tolerant distributed firewall. Other complex questions related with the CRUTIAL security architecture, like policy dissemination and consistency between different CIS in the same security domain, was left as future work.

3 CIS Intrusion Tolerance

In this section we describe the design of the intrusion-tolerant CIS, starting with the design rationale and with the definition of the algorithms it executes, then we present some considerations on how to support statefull firewalls, and finally we describe different ways of implementing a CIS offering different levels of guarantees in terms of criticality and overhead.

3.1 Design Rationale

To understand the *rationale of the design* of the intrusion-tolerant CIS, consider the problem of implementing a replicated firewall between a non-trusted WAN and the trusted LAN that we want to protect. Further assume that we wish to ensure that only the correct messages (according to the deployed policy) go from the WAN side, through the CIS, to the *station computers*[2] in the LAN. A first problem is that the traffic has to be received by all n replicas, instead of only 1 (as in a normal firewall), so that they can perform their active replication mechanisms. A second problem is that up to f replicas can be faulty and behave maliciously, both towards other replicas and towards the station computers.

Our solution to the first problem is to use a device (e.g., an Ethernet hub) to broadcast the traffic to all replicas.

[2]Station computers in SCADA/PCS networked systems are the front-ends of control devices.

These verify whether the messages comply with the OrBAC policy and do a vote, approving the messages if and only if at least $f+1$ different replicas vote in favor. A message approved by the CIS is then forwarded to its destination by a distinguished replica, the *leader*, so there is no unnecessary traffic multiplication inside the LAN. The attentive reader will identify three problems, addressed later in the paper: *dealing with omissions in the broadcast; ensuring that there is always one and only one leader; and that it is correct*.

The existence of faulty replicas is usually addressed with masking protocols of the Byzantine type, which extract the correct result from the n replicas, despite f maliciously faulty: only messages approved by $f+1$ replicas should go through (one of which must be correct since at most f can be faulty). Since the result must be sent to the station computers, either it is consolidated at the source, or at the destination.

The simplest and most usual approach is to implement a front-end in the destination host that accepts a message if: (1) $f+1$ different replicas send it; or (2) the message has a certificate showing that $f+1$ replicas approve it [5]; or (3) the message has a signature generated by $f+1$ replicas using a threshold cryptography scheme [14]. This would imply modifying the hosts's software. However, modifying the software of the SCADA/PCS system can be complicated, and the traffic inside the protected LAN would be multiplied by n in certain cases (every replica would send the message to the LAN), so this solution is undesirable.

So we should turn ourselves to consolidating at the source, and sending *only one, but correct, forwarded message*, in a way similar to an active replication scheme of Delta-4 [11]. However, what is innovative here is that source-consolidation mechanisms should be transparent to the (standard) station computers. Moreover, a faulty replica (leader or not) has access to the LAN (contrarily to the proposal of [11] where only the fail-silent adapters had access to the LAN) so it can send incorrect traffic to the station computers, which typically can not distinguish faulty from correct replicas. This makes consolidation at the source a hard problem.

The solution to the second problem lies on using IPSEC, a set of standard protocols that are expected to be generalized in SCADA/PCS systems, according to best practice recommendations from expert organizations and governments [29]. Henceforth, we assume that the IPSEC Authentication Header (AH) protocol [23] runs both in the station computers and in the CIS replicas. The basic idea is that station computers will only accept messages with a valid IPSEC/AH *Message Authentication Code* (MAC), which can only be produced if the message is approved by $f+1$ different replicas. However, IPSEC/AH MACs are generated using a shared key[3] K and a hash function, so it is not possible to use threshold cryptography. As the attentive reader will note, the shared key storage becomes a vulnerability point that can not be overlooked in a high resilience design, therefore, *there must be some secure component that stores the shared key and produces MACs for messages approved by $f+1$ replicas*.

As a final facet of our design, in order to ensure perpetual correctness we employ proactive recovery, which avoids resource exhaustion due to accidental or malicious corruption of components [35]. The idea is simple: components are periodically rejuvenated to remove the effects of malicious attacks/faults. If the rejuvenation is performed sufficiently often, then an attacker is unable to corrupt enough resources to break the system. Therefore, using proactive recovery we can increase the resilience of the replicated CIS: an unbounded number of intrusions may occur during its lifetime, as long as no more than f occur between rejuvenations. Given that proactive recovery can only be implemented with synchrony assumptions [36, 35], a final catch is that *there must be some local secure and timely component responsible for triggering and executing periodic recoveries*.

The requirements in the previous two paragraphs, for secure and/or timely components in an otherwise asynchronous and Byzantine-on-failure environment, call for a hybrid architecture that includes *a distributed secure and timely wormhole* [39]. This wormhole can be deployed as a set of local trustworthy components (one per replica) connected by a separate control channel. Figure 2 depicts the intrusion-tolerant CIS architecture.

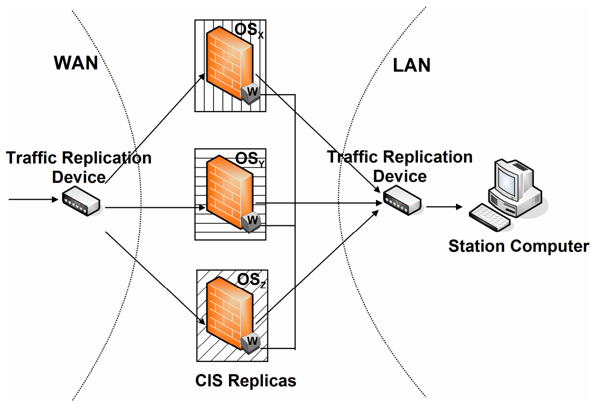


Figure 2. Intrusion-tolerant CIS architecture.

Each CIS replica is deployed in a different operating system (e.g., Linux, FreeBSD, Windows XP), and the operating systems are configured to use different passwords and different internal firewalls (e.g., iptables, ipf, Windows firewall). The wormhole (represented by the small W boxes) is synchronous and secure enough that it can trigger the proac-

[3]We assume that IPSEC/AH is used with manual key management [24].

tive recovery on time and execute a simple voting protocol that produces a MAC for a message if at least $f+1$ replicas approved it (if the wormhole is secure, no malicious replica can force it to sign an unapproved message). Moreover, the wormhole also simplifies the election of a new leader, as discussed later. A final feature of the wormhole has to do with recovery. As already said in the introduction, in this paper we extend the proactive recovery model to support also *reactive recovery*: when the wormhole recognizes a replica as faulty, it recovers the replica as soon as possible, ensuring nevertheless that there is always an amount of replicas to sustain system's correct operation.

A second traffic replication device (see figure, right hand side) is used precisely for the replicas to receive whatever the others send to the LAN. When a (correct) replica believes that another replica is faulty (e.g., if it sends non-approved messages to the LAN, or it is the leader and does not forward approved messages), this replica is suspected to be faulty. When a quorum of replicas suspect some replica, it is recovered.

The CIS does not provide exactly-once semantics, i.e., messages can be lost when the traffic is high and the reception buffers of the replicas become full, or when the leader is recovered. This is not different from firewalls, except that the CIS operation is more complex so the throughput is expected to be lower. However, it is important to notice that almost all wide area control protocols, and specially the ones designed for Power Systems like ICCP [21] and IEC 61850 [22], are designed on top of TCP, which ensures reliable communication even if the network (in this case the CIS) loses messages.

3.2 System Model

The system is composed by n CIS replicas $CIS = \{CIS_1,...,CIS_n\}$. These replicas are deployed in the intersection between the WAN and the LAN in such a way that all data crossing the boundaries of one of these networks must pass through the CIS. The hybrid system model encompasses two parts [39]: the *payload* and the *wormhole*.

Payload. *Asynchronous system* with $n \geq 2f+1$ replicas in which at most f can be subject to *Byzantine failures* in a given recovery period. If a replica does not fail between two recoveries it is said to be *correct*, otherwise it is said to be *faulty*. Every CIS replica has a local clock which is assumed only to make progress. These clocks are not synchronized. We assume *fault independence* for the replicas, i.e., the probability of a replica be compromised is independent of another replica failure. This assumption can be substantiated in practice using several kinds of diversity [28]. Some concerns about this issue are presented in Section 4. Finally, we assume also that the station computers can not be compromised[4].

Wormhole. *Synchronous subsystem* $W = \{W_1,...,W_n\}$ in which at most $f_c \leq f$ local wormholes can *fail by crash*. We assume that when a local wormhole W_i crashes, the corresponding payload replica CIS_i crashes together. The control channel used by the wormhole is isolated and synchronous, offering reliable multicast semantics. Each local wormhole and the station computers share a symmetric key K, and the station computers only accept messages authenticated using this key.

Networks. Besides the CIS replicas model, we have to define the assumptions underlying LAN and WAN communication. We consider that the messages that arrive at CIS replicas both from the WAN and the LAN have *unreliable fair multicast* semantics, an adaptation of the fair links abstraction to multicast: if a message is multicasted infinitely many times it will be received by all its receivers infinitely many times. The two primitives offered by this service are: *U-multicast*(G,m), to multicast a message m to the group G, and *U-receive*(G,m), to receive m that was multicast to G. This is substantiated in practice by the traffic replication devices. We assume that *all* communication between replicas and other machines from the WAN and the LAN are based in these primitives. For the LAN, we assume also that communication is source-authenticated and the sender of a message m is denoted by *sender*(m). Later we will show how these assumptions were substantiated in our prototype.

Cryptography. The MACs used in IPSEC/AH are assumed to inherit the *collision resistance* property from the hash functions in which they are based [29], i.e., that it is infeasible to find two messages that for a key K have the same MAC.

3.3 Service Properties

First of all, let us define the concept of legal message: a message is said to be *legal* if it is in accordance with the current deployed policy P. A message not in accordance with P is said to be *illegal*. Moreover, a message is said to be *processed* by its destination machine if its content is delivered to the application layer (e.g., the SCADA system). The basic properties offered by the CIS are the following:

- *Validity*: A legal message received by at least one correct CIS replica that is not recovered for long enough, is forwarded to its destination;
- *Integrity*: An illegal message is never processed by its destination machine.

[4]It is the trusted network that we aim to protect. Moreover, it does not make sense to protect something that can be compromised and attack the firewall.

Notice that these two properties are sufficiently weak to be satisfied by a system with unreliable fair multicast communication and strong enough to ensure that only legal messages will be processed at LAN hosts. The notion of a correct CIS replica not being recovered for "long enough" is formalized in Appendix A. It captures the idea that if only one correct replica receives the message and is recovered immediately after, the message is lost and is not forwarded. Notice that this type of property is a consequence of using proactive and reactive recovery. Any system enhanced with this type of recovery mechanisms will need to do something useful in the interval within recoveries, or else the system will be always recovering without executing any of its functionalities.

The two properties described above are satisfied in a system with n CIS replicas if the maximum number of faulty replicas in a given recovery period is $t \leq f$.

3.4 Message Processing Algorithm

This section presents the main algorithm executed by the CIS to process messages incoming from the WAN to the protected LAN. The same algorithm is used to handle messages coming from the opposite direction (possibly with a more relaxed policy).

The policy verification is made in a *policy engine* accessed through the function *PolEng_verify*.

Wormhole interface. The interactions between a replica CIS_i and its local wormhole W_i are made through a well defined interface, offering the following services:

- $W\_sign(m)$: returns a MAC of message m obtained with the shared key K, if and only if at least $f+1$ CIS replicas call $W\_sign(m)$ in their local wormholes;
- $W\_verify(m_\sigma)$: returns *true* if σ is a MAC of m produced with K, and *false* otherwise;
- $W\_leader()$: returns the id of the current leader replica;
- $W\_suspect\_silent(j)$ and $W\_suspect\_malicious(j)$: notify the wormhole, respectively, that the replica CIS_j appears to be silent or that the replica CIS_j executed some malicious action. If $f+1$ replicas suspect of CIS_j, it is recovered. This recovery can be done immediately, without availability concerns, in the presence of at least $f+1$ malicious suspicions, given that in this case at least one correct replica detected that replica CIS_j executed some malicious action. However, if the number of malicious suspicions is less than $f+1$, i.e., if some of the suspicions are of the silent type, then replica CIS_j may be correct. Therefore, the recovery needs to be coordinated with the periodic proactive recoveries in order to guarantee that a minimum number of correct replicas is always available. This mechanism will be explained in Section 3.5.2. Moreover, if the replica to be recovered is the leader, a new one is elected by the wormhole.

Algorithm 1 CIS payload (replica CIS_i).

```
{Parameters}
integer $T_{vote}$ {Expected time to vote a message}
integer $Ot$ {Omission threshold for the leader}
integer $T_{forward}$ {Expected time to forward a message}
{Variables}
integer $leader = 0$ {Current leader}
integer $leader\_omissions = 0$ {Leader omissions counter}
set $Voting = \varnothing$ {Messages being voted}
set $Pending = \varnothing$ {Not yet forwarded messages}
set $TooEarly = \varnothing$ {Messages forwarded before their arrival}
upon $U\text{-}receive(WAN, m)$
 1: if $PolEng\_verify(m)$ then
 2:    $Voting \leftarrow Voting \cup \{m\}$
 3:    $\sigma \leftarrow W\_sign(m)$
 4:    $Voting \leftarrow Voting \backslash \{m\}$
 5:    if $m_\sigma \in TooEarly$ then
 6:       $TooEarly \leftarrow TooEarly \setminus \{m_\sigma\}$
 7:    else
 8:       $Pending \leftarrow Pending \cup \{m_\sigma\}$
 9:       if $leader = i$ then $U\text{-}multicast(LAN, m_\sigma)$ end if
10:    end if
11: end if
upon $U\text{-}receive(LAN, m_\sigma)$
12: if $m_\sigma \in Pending$ then
13:    $Pending \leftarrow Pending \setminus \{m_\sigma\}$
14: else if $W\_verify(m_\sigma)$ then
15:    $TooEarly \leftarrow TooEarly \cup \{m_\sigma\}$
16: else
17:    $W\_suspect\_malicious(sender(m_\sigma))$
18: end if
for each $T_{vote}$ time units that $Voting \neq \varnothing$
19: $U\text{-}multicast(WAN, Voting)$
for each $T_{forward}$ time units that $m_\sigma \in Pending$
20: $U\text{-}multicast(WAN, m)$
21: $leader\_omissions \leftarrow leader\_omissions + 1$
22: if $leader\_omissions > Ot$ then $W\_suspect\_silent(leader)$ end if
upon $leader \neq W\_leader()$
23: $leader \leftarrow W\_leader()$
24: $leader\_omissions \leftarrow 0$
25: if $leader = i$ then
26:    for all $m_\sigma \in Pending$ do
27:       $U\text{-}multicast(LAN, m_\sigma)$
28:    end for
29: end if
```

The Algorithm. The CIS replicas execute Algorithm 1. There are three parameters: T_{vote}, the expected time required to receive, vote and sign a legal message; Ot, the omission threshold, i.e., the maximum number of omissions of a leader before it is suspected; and $T_{forward}$, the expected time required for the leader to forward a message to the station computer. Additionally, the algorithm uses five variables: *leader*, the id of the current leader; *leader_omissions*, a counter that stores how many times the current leader did not forward some message; *Voting*, the set of messages being voted; *Pending*, the set of messages received and approved by the replica that were already signed by the wormhole but not forwarded by the leader to the station computer; and *TooEarly*, the set of correctly signed messages forwarded by the leader but not yet received (from the WAN) by the replica.

The algorithm begins when a replica receives a message coming from the WAN (lines 1-11). If the received message is legal, all correct replicas will approve it (line 1) and sign it using the wormhole *W_sign* service (line 3). While the message is being voted and signed by the wormhole, it is stored in the *Voting* set (lines 2 and 4). Then, the message is inserted in the *Pending* set (line 8) and it stays in this set until it is received in the LAN (lines 12-13). Finally, if the replica is the leader, it forwards it to the LAN (line 9).

The algorithm contains several controls to deal with message losses, replica failures and abnormal message ordering. The first of these controls deals with message omissions on the WAN: when a replica receives and approves a message m, it stores it in the *Voting* set (line 2) before requesting the wormhole to sign it. The message is removed from the set when the message is signed (line 4). For each T_{vote} time units that *Voting* is not empty, its content is multicasted to the network (line 19). This ensures that the message being voted will eventually be received by other correct CIS replicas (due to the fairness assumption). The most important problem that the replicas must deal with is the failure of the leader. The leader can fail maliciously, forwarding illegal messages to the LAN, or in benign way, crashing, staying silent or even being too slow. In the first case, the illegal message diffusion will be perceived by correct replicas that will verify that this message is not pending (line 12) and was not correctly signed (line 14). Since we assume that the LAN traffic is source-authenticated, the wormhole will suspect the leader and recover it (line 17). There is also a counter to deal with leader crashes and omissions (lines 20-22). For each $T_{forward}$ that a message stays in the *Pending* set, it is retransmitted in the WAN and an omission counter is increased (lines 20-21). When this counter is greater than *Ot*, the leader is suspected (line 22). With this mechanism, if the leader crashes (or stays silent), it eventually will omit more than *Ot* messages, be suspected by all correct replicas and recovered. It is possible that the leader forwards a correctly signed message to the LAN without some replicas having received it from the WAN. In order to deal with these “early messages” on the LAN, we use the *TooEarly* set. When a not pending legal message is received in the LAN, it will be stored in this set (lines 14-15) and stay there until it is received from the WAN (lines 5-6). This control is used to ensure that an already forwarded message will not stay pending forever in some replica.

When a new leader is elected by the wormhole, all replicas will discover it and the new leader will forward all messages in its *Pending* set to the LAN (lines 23-29).

The proof of correctness of this algorithm is presented in Appendix A. The algorithm is composed of five code blocks and in our proofs we assume that each of these blocks is executed by a different thread. Moreover, we assume the existence of synchronization mechanisms that manage the concurrent access of the various threads to the shared variables. We chose to not include such mechanisms explicitly in the algorithm in order to ease its readability. These considerations apply also to the remaining algorithms presented in this paper.

3.5 Wormhole

In this section we present the algorithms executed inside the CIS wormhole. These algorithms are executed as threads in a real-time environment with a *preemptive scheduler* where static priorities are defined from 1 to 3 (priority 1 being the highest). In these algorithms we do not consider explicitly the clock skew and drift, since we assume that these deviations are compensated in the protocol parameters.

We start by describing the interactive services that are directly accessed by the replica payload (i.e., by Algorithm 1), then explain the proactive-reactive recovery service.

3.5.1 Interactive Services

The interactive services are presented in Algorithm 2. The shared key K (see Section 3.2) is used as a parameter in some services. Moreover, the services provided by each local wormhole W_i use four variables: *current_leader* stores the id of the current leader; V is a vector of sets in that each set $V[\sigma]$ stores the ids of the replicas that are currently approving the message with MAC σ; and the sets $Suspect_m$ and $Suspect_s$ contain the processes that suspect replica i of being, respectively, malicious or silent.

$W\_sign(m)$ calculates the MAC σ of m using the shared key K and then sends this MAC in a VOTE message to all local wormholes (lines 1-2). When such a message is received, the id of its sender is inserted in set $V[\sigma]$ (line 5). After the reception of $f+1$ messages containing σ (line 3), this MAC is returned to the client (line 4). It ensures that the message was approved by at least one correct replica, so it is legal.

Algorithm 2 Wormhole interactive services (local w. W_i).

```
{Parameters}
key $K$ {Service Symmetric key – used for authentication}
{Variables}
integer $current\_leader = n$ {Current leader wormhole}
set $\forall\sigma : V[\sigma] = \varnothing$ {Processes approving message with hash $h$}
set $Suspect_m = \varnothing$ {Processes suspecting me of being malicious}
set $Suspect_s = \varnothing$ {Processes suspecting me of being silent}
{All interactive service threads with priority 3}
service $W\_sign(m)$
  1: $\sigma \leftarrow \mathsf{MAC}(m,K)$
  2: $R\text{-}multicast(W, \langle \text{VOTE}, i, \sigma \rangle)$
  3: wait until $|V[\sigma]| \geq f+1$
  4: return $\sigma$
upon $R\text{-}receive(W, \langle \text{VOTE}, j, \sigma \rangle)$
  5: $V[\sigma] \leftarrow V[\sigma] \cup \{j\}$
service $W\_verify(m_\sigma)$
  6: return $\mathsf{MAC}(m,K) = \sigma$
service $W\_leader()$
  7: return $current\_leader$
service $W\_suspect\_silent(j)$
  8: $send(j, \langle \text{SUSPECT-SILENT} \rangle)$
service $W\_suspect\_malicious(j)$
  9: $send(j, \langle \text{SUSPECT-MALICIOUS} \rangle)$
upon $receive(j, \langle \text{SUSPECT-SILENT} \rangle)$
  10: $Suspect_s \leftarrow Suspect_s \cup \{j\}$
upon $receive(j, \langle \text{SUSPECT-MALICIOUS} \rangle)$
  11: $Suspect_m \leftarrow Suspect_m \cup \{j\}$
```

$W\_verify(m_\sigma)$ receives as input a message m allegedly signed with K. Returns *true* if the MAC for m obtained using K corresponds to σ, and *false* otherwise (line 6).

$W\_leader()$ returns the id of the current leader (line 7).

$W\_suspect\_silent(j)$ and $W\_suspect\_malicious(j)$ send, respectively, a SUSPECT-SILENT or SUSPECT-MALICIOUS message to W_j (lines 8-9). When a local wormhole W_i receives such a message from W_j, j is inserted in one of the *Suspect* sets according to the type of the message (lines 10-11). The content of these sets may trigger a recovery procedure, as explained in the next section.

3.5.2 The Proactive-Reactive Recovery Service

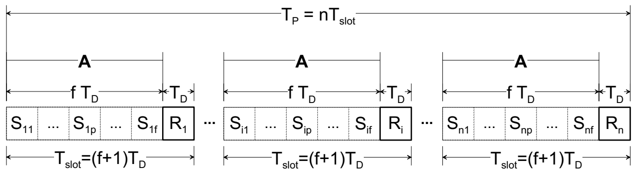
Proactive-reactive recovery initiates recoveries both periodically (time-triggered) and whenever something bad is detected or suspected (event-triggered). Periodic recoveries $R_1, \ldots, R_i, \ldots, R_n$ are done in sequence, so no more than one CIS replica is restarting at the same time. However, the interval between these recoveries is not tight. Instead we allocate f intervals for recovery between periodic recoveries such that they can be used by event-triggered recoveries.

Algorithm 3 presents an implementation of this service, which uses two main parameters: T_P and T_D. T_P defines the maximum time interval between consecutive triggering of the *recover* procedure; and T_D defines the worst case execution time of the a recovery. In Appendix B we prove that under these assumptions the system operates correctly perpetually.

The approach is based on real-time scheduling with an *aperiodic server task* to model aperiodic tasks [37]. The idea is to consider the recovery as the resource and ensure that no more than one correct replica will be restarting simultaneously. This condition is important to ensure that the system always stays available. Two types of real-time tasks are utilized by the proposed mechanism:

- task R_i: represents the recovery of node CIS_i. All these tasks have worst case execution time T_D and period T_P;
- task A: is the aperiodic server task, which can handle at most f recoveries every time it is activated. This task has worst case execution time fT_D and period $(f+1)T_D$.

Task R_i is executed at local wormhole W_i, while task A is executed in all wormholes, but only the ones with the payload suspected of being faulty are recovered. The time needed for executing one A and one R_i is called the *recovery slot i* and is denoted by T_{slot}. Every slot i has f *recovery subslots* belonging to the A task, each one denoted by S_{ip}, plus a R_i. Figure 3 illustrates how time-triggered periodic and event-triggered aperiodic recoveries are combined.



$T_P = nT_{slot}$
A
A
A
$f\ T_D$
T_D
$f\ T_D$
T_D
$f\ T_D$
T_D
S_{11} ... S_{1p} ... S_{1f} R_1 ... S_{i1} ... S_{ip} ... S_{if} R_i ... S_{n1} ... S_{np} ... S_{nf} R_n
$T_{slot}=(f+1)T_D$
$T_{slot}=(f+1)T_D$
$T_{slot}=(f+1)T_D$

Figure 3. Recovery schedule.

Notice that a reactive recovery only needs to be scheduled according to the described mechanism if the replica CIS_i to recover is not assuredly malicious, i.e., if less than $f+1$ replicas have called $W\_suspect\_malicious(i)$ (but at least $f+1$ replicas have called either $W\_suspect\_malicious(i)$ or $W\_suspect\_silent(i)$). If the wormhole W_i knows with certainty that replica CIS_i is faulty, i.e., if a minimum of $f+1$ replicas have called $W\_suspect\_malicious(i)$, CIS_i can be recovered without extra care, since it is accounted as one of the f faulty replicas.

The proactive-reactive recovery service is presented in Algorithm 3. This algorithm uses five variables: t_{last} stores the instant when the last periodic recovery was triggered by local wormhole W_i; the *Crashed* set stores the id of the crashed replicas (at most f_c); *current_leader*, $Suspect_m$, and $Suspect_s$, correspond to the variables with the same name in Algorithm 2.

The *proactive_recovery*() procedure is triggered by each local wormhole W_i at boot time (lines 1-7). It starts by calling a routine that (i.) synchronizes the clocks of the local wormholes with the goal of creating a virtual global clock, and (ii.) blocks until all local wormholes call it and can start at the same time (line 1). The primitive *global_clock*() returns the current value of the global clock. After the initial synchronization, the variable t_{last} is initialized (line 2) in a way that local wormholes trigger periodic recoveries according to their id order, and the first periodic recovery triggered by every local wormhole is finished within T_P from the execution of line 1. After this initialization, the procedure enters an infinite loop where a periodic recovery is triggered within T_P from the last triggering (lines 3-7).

Reactive recoveries can be triggered in two ways: (1) if the local wormhole W_i receives at least $f+1$ SUSPECT-MALICIOUS messages, then recovery is initiated immediately (line 8); (2) otherwise, if $f+1$ SUSPECT-MALICIOUS or SUSPECT-SILENT messages arrive, then replica i is at best silent. In both cases, the $f+1$ bound ensures that at least one correct CIS detected a problem with replica i. In the silence scenario, recovery does not have to be started immediately because the replica might only be slow. Instead, the aperiodic task finds the closest slot where the replica can be recovered without endangering the availability of the replicated CIS. The idea is to allocate one of the (reactive) recovery subslots depicted in Figure 3. This is done through the function *allocate_subslot*() (line 9 – explained later). Notice that if the calculated subslot S_{jp} is located in the slot where the replica will be proactively recovered, i.e., if $j = i$, then the replica does not need to be reactively recovered (line 10). If this is not the case, then W_i waits for the allocated subslot and then recovers the corresponding replica (lines 11-12). Notice that the expression *global_clock*() mod T_P returns the time elapsed since the beginning of the current period, i.e., the position of the current global time instant in terms of the time diagram presented in Figure 3.

The *recover*() procedure (lines 14-22) starts by checking if the current leader is going to be recovered. If this is the case, a new leader is elected and this information is sent to all local wormholes (lines 15-16). The leader election is done through a deterministic local algorithm (i.e., no distributed communication is needed) that calculates the replica most recently rejuvenated through a periodic recovery (lines 23-27). After leader election, the payload operating system (OS) is shutdown (line 18). Then, the OS and CIS codes are restored (e.g., from a read-only medium such as a Read-Only-Memory or a write-protected hard disk), and finally the OS is booted from the clean code, bringing the replica to a correct state (lines 19-20). We assume that the local CIS replica code is automatically started once the OS finishes the booting process. The last two lines of the *recover*() procedure re-initialize some variables because the CIS replica is now assuredly correct (lines 21-22).

The protocol deals also with the crash of at most f_c wormholes (and, consequently, the corresponding CIS replicas). These events are detected by a perfect failure detector that can be easily implemented in a crash-prone synchronous system. When a leader is detected to be crashed, its id is added to the *Crashed* set and a new leader is elected (lines 29-30). Notice that the leader election procedure does not elect leader a crashed replica (lines 24-25).

Subslot management is based on a simple state machine replication scheme (lines 34-36). A subslot is allocated by W_i by invoking the function *allocate_subslot*(), which sends an ALLOC message using total order multicast (line 34) to all local wormholes and waits until this message is received. At this point the function *local_allocate_subslot*(i) is called and the next available subslot is allocated to the replica (line 36). Total order multicast ensures that all local wormholes allocate the same subslots in the same order (line 37). The exact algorithm of the *local_allocate_subslot*(i) function is not presented in order to ease the readability of the overall algorithm. The goal of *local_allocate_subslot*(i) is to manage the various recovery subslots and to assign them to the replicas that request to be recovered.

Notice that the CIS may become unavailable during a recovery if f replicas (different from the one that is being proactively recovered) are faulty [34]. In order to avoid this unavailability problem, the CIS should tolerate the "pseudo-crash" provoked by a recovery. Therefore, given that CIS replicas recover in sequence, one at a time (formally proved in Appendix B), we need $n \geq 2f+2$ replicas in order to guarantee availability during recoveries. This lower-bound results from applying the reasoning presented in [34] to CIS: if one assumes that at most k replicas recover at the same time, $n \geq 2f+k+1$ replicas will be needed to ensure CIS availability.

3.6 Supporting Statefull Firewalls

The CIS design applies to stateless firewalls, which is largely the most common type of firewall used. However, some applications require statefull security policies, in which traffic is approved or denied taking into consideration the messages approved/denied in the past (e.g., some data message is only forwarded if some connection message was sent before). As already said, several critical infrastructures applications are not secure-aware and must rely on external protection mechanisms (like the CIS) for its security. Such applications need statefull firewalls in order to enforce

Algorithm 3 Wormhole proactive-reactive recovery service (local wormhole W_i).

```
{Parameters}
integer $T_P$ {Periodic recovery period}
integer $T_D$ {Recovery duration time}
{Constants}
integer $T_{slot} \triangleq (f+1)T_D$ {Slot duration time}
{Variables}
integer $t_{last} = 0$ {instant of the last periodic recovery start}
set $Crashed = \varnothing$ {Crashed replicas (will not be recovered)}
integer $current\_leader = n$ {Current leader wormhole}
set $Suspect_m = \varnothing$ {Processes suspecting me of being malicious}
set $Suspect_s = \varnothing$ {Processes suspecting me of being silent}
{Periodic recovery thread with priority 1}
procedure $proactive\_recovery()$
 1: $synchronize\_global\_clock()$
 2: $t_{last} \leftarrow global\_clock() - T_P + ((i-1)T_{slot} + fT_D)$
 3: loop
 4:     wait until $global\_clock() = t_{last} + T_P$
 5:     $t_{last} = global\_clock()$
 6:     $recover()$
 7: end loop
{Threads with priority 2}
upon $|Suspect_m| \geq f+1$
 8: $recover()$
upon $(|Suspect_m| < f+1) \wedge (|Suspect_s + Suspect_m| \geq f+1)$
 9: $\langle j,p \rangle \leftarrow allocate\_subslot()$
10: if $j \neq i$ then
11:     wait until $global\_clock()$ mod $T_P = (j-1)T_{slot} + (p-1)T_D$
12:     $recover()$
13: end if
procedure $recover()$
14: if $current\_leader = i$ then
15:     $elect\_new\_leader()$
16:     $R\text{-}multicast(W, \langle \text{NEW-LEADER}, i, new\_leader \rangle)$
17: end if
18: $shutdown\_OS()$
19: $restore\_code()$
20: $boot\_OS()$
21: $Suspect_m \leftarrow \varnothing$
22: $Suspect_s \leftarrow \varnothing$
procedure $elect\_new\_leader()$
23: $new\_leader \leftarrow last\_periodic\_recovered\_node()$
24: while $new\_leader \in Crashed$ do
25:     $new\_leader \leftarrow ((new\_leader - 2) \bmod n) + 1$
26: end while
27: $current\_leader \leftarrow new\_leader$
upon $R\text{-}receive(W, \langle \text{NEW-LEADER}, l, new\_leader \rangle)$
28: if $l = current\_leader$ then $current\_leader \leftarrow new\_leader$ end if
upon $current\_leader$ is faulty
29: $Crashed \leftarrow Crashed \cup \{current\_leader\}$
30: $elect\_new\_leader()$
procedure $last\_periodic\_recovered\_node()$
31: $t_{round} \leftarrow global\_clock() \bmod T_P$
32: $j \leftarrow (\lfloor \frac{t_{round}}{T_{slot}} \rfloor \bmod n)$
33: if $j = 0$ then return $n$ else return $j$ end if
procedure $allocate\_subslot()$
34: $TO\text{-}multicast(W, \langle \text{ALLOC}, i \rangle)$
35: wait until $TO\text{-}receive(W, \langle \text{ALLOC}, i \rangle)$
36: return $local\_allocate\_subslot(i)$
upon $TO\text{-}receive(W, \langle \text{ALLOC}, j \rangle) \wedge j \neq i$
37: $local\_allocate\_subslot(j)$
```

powerful security policies.

The CIS policy engine can provide statefull semantics as long as one ensures that all messages are verified in the same order in all CIS replicas. In other words, we need an agreement protocol to ensure that all messages are evaluated in total order. This protocol could require either f more replicas and some timing assumptions (e.g., [10]) or the same number of replicas and a distributed secure and timely wormhole [12] (which is already part of the design but does not implement the required services). Moreover, if the policy engine is statefull, its state needs to be transferred from other replicas after each recovery. State transfer is needed because the recovering replica will not receive the messages that pass through the firewall during the recovery process. Additionally, the recovering replica may have been compromised before the recovery, and it may be necessary to transfer a clean state from remote replicas. In [10], it is presented an efficient mechanism to perform checkpoints and state transfer, specially tailored for replication mechanisms subject to Byzantine faults.

3.7 Deployment Space

The intrusion-tolerant CIS presented in the previous sections can be implemented and deployed in several ways, depending on the criticality of the LAN being protected and the tolerance to unexpected delays of the services deployed in this LAN. Figure 4 presents the main four design options.

In this figure it is possible to identify two main dimensions in the design space. If the LAN protected by the CIS provides a very critical service, the implementation must be based on the use of different physical replicas for each CIS replica, allowing tolerance to physical and software faults. However, since price is always a major concern, a much more cost effective solution can be attained by resorting to virtualization. The various replicas are deployed in the same host, using virtual machines (VM) to isolate the different runtime environments, preventing intrusions from propagating from one replica to the others. If the service protected is part of an application with hard real-time guarantees, without tolerance to unexpected delays, there must

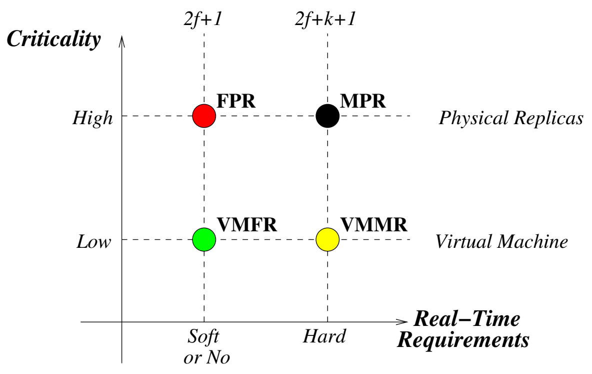


Figure 4. Deployment space for the intrusion-tolerant CIS.

be at least $2f+k+1$ replicas to ensure safe operation and availability even in case of f faults an k recovering replicas. On the other hand, if occasional delays are not a problem, $2f+1$ replicas suffice to maintain correct operation of the system. The four main options are the following:

- *VMFR (Virtual Machines with Few Replicas)*: $2f+1$ replicas are deployed as virtual machines running in the same host. The system does not tolerate physical faults or bugs in the virtual machine monitor, but offers some level of protection if a virtual machine is subject to an intrusion;
- *VMMR (Virtual Machines with Many Replicas)*: There are $2f+k+1$ logical replicas running in different virtual machines. The tolerance to malicious behavior is the same of VMFR but now the system continues to verify and forward messages even with f faulty and k recovering replicas;
- *FPR (Few Physical Replicas)*: We expect this to be the most common form of deployment, as depicted in Figure 2. There is some tolerance to accidental hardware and software faults as well as intrusions;
- *MPR (Many Physical Replicas)*: This is similar to FPR but with k more replicas, to allow availability even with faulty replicas.

These deployments can be used in different points of the critical infrastructure to protect different applications [18]. For example, a MPR deployment can be used to protect a critical grid defense system (e.g., Automatic Grid Separation System – that performs automatic grid separations in emergency states) while a VMMR implementation can be adequate to protect a substation data historian network.

4 CIS Prototype

We have chosen to implement the VMFR and VMMR designs for two main reasons: first, our group already experimented a wormhole construction with different physical replicas [13] but to the best of our knowledge there are no previous experiments with this kind of system deployed in virtual machines to obtain intrusion tolerance; second, a typical critical infrastructure has hundreds of station computers and other hosts to be protected, and a solution based on virtual machines can reduce the deployment cost[5].

Our implementation is based on the XEN virtual machine monitor [3] with the Linux operating system. The architecture is presented in Figure 5. A XEN system has multiple layers, the lowest and most privileged of which is XEN itself. XEN may host multiple guest operating systems, every one executed within an isolated VM or, in XEN terminology, a *domain*. Domains are scheduled by XEN to make effective use of the available physical resources (e.g., CPUs). Each guest OS manages its own applications. The first domain, *dom0*, is created automatically when the system boots and has special management privileges. Domain *dom0* builds other domains (*dom1, dom2, dom3, ...*) and manages their virtual devices. It also performs administrative tasks such as suspending and resuming other VMs, and it can be configured to execute with higher priority than the remaining VMs.

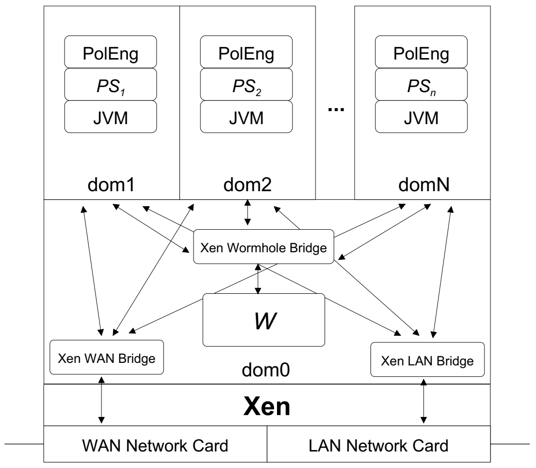


Figure 5. VM-based CIS architecture.

As depicted in Figure 5, a centralized version of the wormhole (W) was deployed in domain *dom*0. This version (implemented in C) emulates the n different local wormholes, and each of these wormholes receives requests and

[5]For example, the Italian Power System contains about 1000 station computers, thus using three machines and two hubs to protect every one of these computers would require 3000 and 2000 extra machines and hubs, respectively.

sends replies to their corresponding replica. The control channel is emulated using shared memory. In the payload side, each replica *i* runs on a different domain *domi*. The replica payload is composed of the policy engine, the CIS protection service (PS_i), and a Java Virtual Machine (JVM), given that both policy engine and protection service are implemented in Java. In the current prototype, the policy engine is very simple, since it just checks if certain strings are present in the received packets.

Given that the various payloads and wormhole run in different VMs, the communication between them is done through a virtual bridge. This bridge emulates a private pipe between each replica and its corresponding local wormhole. The two remaining bridges emulate the WAN and the LAN networks. These two bridges are configured to operate in broadcast mode, and the LAN bridge authenticates the source address of the messages sent by replicas (i.e., replicas can only send messages using their own id)[6]. This way, the authentication assumption in Section 3.2 is substantiated.

To simplify the design and to avoid changes in the kernel, the CIS prototype operates at the UDP level, instead of IP level as most firewalls do. Therefore, there was no need to implement packet interception because packets are sent directly to the CIS. Moreover, authentication is not done at the IP level (as when using IPSEC/AH), but in alternative the wormhole calculates the HMAC[7] of the payload UDP packet, and then the two are concatenated. Notice that this type of authentication implies the same type of overhead of IP authentication.

The periodic recovery resets the state and restores the code of a replica. While this is useful to restore the correctness of the replica, it would be interesting if we were able to introduce diversity in the recovery process. For instance, each recovery could randomize the address space of the replica (e.g., using PAX[8]) in order to minimize the usefulness of the knowledge obtained in the past to increase the chances of future attacks. The XEN and PAX communities are currently making efforts to release a joint distribution, and we plan to integrate this mechanism in the prototype when it is available.

Domain *dom*0 executes with higher priority than the remaining VMs, but since it is based on a normal Linux OS, it provides no real-time guarantees. Currently, only modified versions of Linux and NetBSD can be run on *dom*0. However, XEN is continuously being improved and we expect that in the near future one can use a real-time OS.

[6]In a physical switch, a similar effect can be attained by using the failopen mode to force broadcast, and the switch's access control lists to prevent replicas from spoofing their MAC addresses, which can be used as identifiers.

[7]HMAC is a standard for calculating MACs, and in the prototype we used the SHA-1 hash function.

[8]Available at `http://pax.grsecurity.net/`.

5 Experimental Evaluation

We set up a VM-based CIS and connected it to two 100 Mbps switches representing the WAN and the LAN. A PC emulated the station computer and, in the WAN side, two PCs were deployed: a *good* sender trying to transmit legal traffic to the station computer, and a *malicious* sender causing a denial of service attack. The CIS executed on a dual-processor 2.4 GHz Pentium Xeon PC with 2 GB RAM and XEN 3.0; the good sender a 500 MHz Pentium 3 PC with 256 MB RAM; the malicious sender a 2.8 GHz Pentium 4 PC with 2 GB RAM; the station computer a 1.8 GHz Celeron PC with 512 MB RAM. The malicious sender run Linux 2.6.11, and the others run Linux 2.6.18. The JVM was Sun JDK 1.5.09.

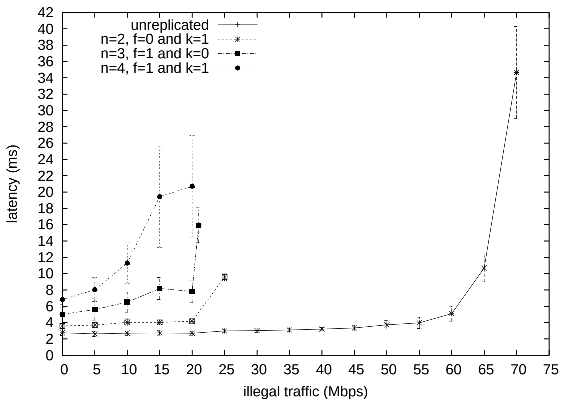
During the experiments, the malicious sender is constantly injecting illegal traffic at a certain rate. The IPERF tool[9] was employed to perform this task. The good sender transmits legal packets (of 1470 bytes) at a different rate. In the throughput and loss rate experiments, legal packets are sent at a rate of 124 packets/second and the obtained results correspond to the worst case values (lowest throughput and highest loss rate) that were observed. Notice that this is a very high traffic for a SCADA system, since it represents, for instance, 124 MMS commands being sent to a device per second. In the latency experiment, a legal packet is transmitted after the arrival of an acknowledgement of the previous packet. The displayed results correspond to the average and standard deviation values of 1000 transmissions, after excluding the 5% most distant from the average.

Figure 6 presents the latency, throughput and loss rate of several CIS configurations, when distinct levels of illegal traffic were being injected in the network. The configurations correspond basically to a system with different number of replicas, resulting from various values of f and k (see Sections 3.4 and 3.5). Notice that the results for the configurations stop at distinct maximum levels of illegal traffic. This happens because after certain traffic levels, the configurations become unstable, leading to high latencies and/or loss rates.

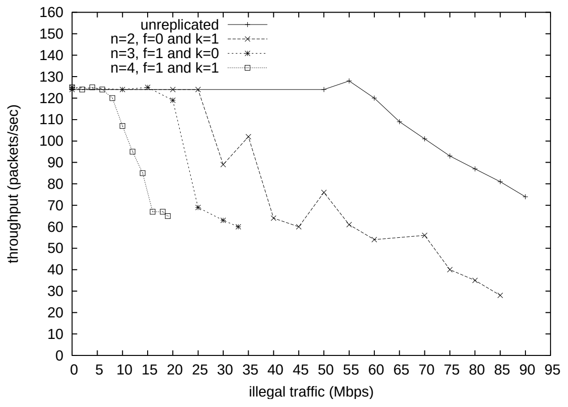
As expected, the results show that latency and message loss increases, while throughput decreases, when the amount of illegal traffic directed from the WAN to the LAN becomes larger. There are no pre-established acceptable limits for these parameters, except for the latency that in power systems is typically satisfactory if it remains lower than 200 milliseconds [18]. In the figure it is possible to observe that latency stayed way below this limit for very reasonable levels of traffic.

Comparing the maximum amount of illegal traffic that the several configurations can deal with, one can see that

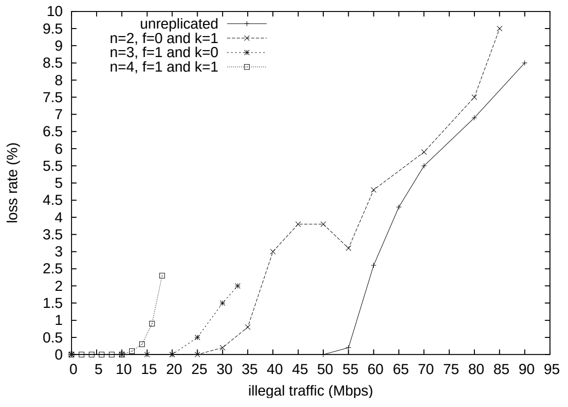
[9]Available at `http://dast.nlanr.net/Projects/Iperf/`.



(a) Latency



(b) Minimum Throughput



(c) Maximum Loss Rate

Figure 6. Experimental evaluation of the CIS prototype.

this threshold is directly proportional to the number of replicas n. This happens because all replicas share the *same* physical machine and, consequently, the amount of available resources has to be divided. In particular, our experiments allowed us to conclude that the amount of memory allocated for each VM is a critical factor in how much illegal traffic the CIS supports. For a setup with 4 replicas, resisting 20 Mbps of illegal traffic is a good achievement, specially if we take into account that it is very easy to limit this traffic at the WAN gateway.

In a setup with more resources, such as one with n physical replicas, we expect that the overall behavior of the CIS will be closer to the non-replicated case in the figure (with a minor degradation due to the distributed implementation of W_sign). This means that the physically distributed CIS would maintain acceptable operation even in the presence of illegal traffic of up to 70 Mbps, which is a value close to the network capacity, 100 Mbps. This result is even more interesting if we consider that the CIS was implemented at application level and not inside the kernel.

6 Related Work

Work on *intrusion-tolerant services* has been fundamentally done considering two replication models: Byzantine quorum systems (e.g., [26, 44]) and state machine replication (e.g., [10, 2]). The main difference between these approaches is on the way they maintain state consistency among the replicas: quorum systems are based on intersections between different quorums (sets of servers) while state machine replication relies on total order multicast protocols to ensure that all replicas evolve in the same way. Neither of these approaches is used in the CIS algorithms. The replicated CIS forwards a message if it is accepted by $f+1$ replicas. Due to this particularity, CIS requires significantly less replicas ($n \geq 2f+1$) than the usual $n \geq 3f+1$ bound.

There was also some previous research on using proactive recovery to increase system resilience [44, 10, 35]. To the best of our knowledge, we are the first to combine reactive and proactive recovery in a single approach.

Some intrusion-tolerant services have been proposed in the past: file/storage systems [17, 10, 27], certification authorities [32, 44], and DNS [9]. The only other research that we are aware of about intrusion-tolerant firewalls is the privacy firewall described by Yin et al. [43]. The privacy firewall was proposed to enforce confidentiality in Byzantine services. Since this objective differs from ours, the CIS design requires much less replicas ($n \geq 2f+1$ instead of $n \geq (f+1)^2$) and does not need expensive threshold signatures. Most important, the privacy firewall requires modification on both sides of the firewall, to be able to verify approved messages, i.e., to verify if incoming messages are correctly signed. In addition, the privacy firewall does not recover replicas. Moreover, from the perspective of the service provided, the CIS is closer to normal firewalls than to the privacy firewall.

Regarding the *dependability of critical infrastructures*, several works talk about how to integrate security in these systems. Most of this research either analyzes current security incidents (e.g., [8]) or focus on the use of traditional security mechanisms (like firewalls) to protect SCADA/PCS systems (e.g., [6]). However, people have argued that the critical infrastructures requirements are different from the ones in corporate networks, and, more importantly, it has been shown that most firewalls available today are unable to deal with these requirements [7].

There is one project that investigates an advanced protection and communication infrastructure for critical infrastructures, GRIDSTAT and its successor TCIP (Trustworthy Cyber Infrastructure for the Power Grid). This project advocates the use of a publish-subscribe infrastructure to provide secure and QoS-enabled communications between different control centers in a power grid. Much of the work being developed in these projects is orthogonal to the work presented in this paper and to the CRUTIAL project in general, since it is focused on QoS, real-time communication, and trust management [20].

Regarding the use of virtualisation technology to construct intrusion-tolerant services, the VM-FIT architecture proposed in [31] provides basic support for intrusion-tolerant replication of services. Similarly to our VM-based CIS prototype, VM-FIT uses virtualisation to provide a trusted component (i.e., a wormhole) on each machine. However, the goal of the VM-FIT wormhole is to intercept client-service interaction and to distribute requests to the replica group. Overall, the goal of the VM-FIT architecture is conceptually different from the goal of CIS. VM-FIT is a generic architecture that allows to transform any deterministic centralized service in an intrusion-tolerant replicated one, whereas CIS is a specific intrusion-tolerant replicated service.

7 Conclusions and Future Work

This paper presented the CIS protection service, implemented by highly resilient distributed devices, aimed at protecting critical information infrastructures. The CIS has three distinguishing features: it is intrusion-tolerant, it seeks unattended perpetual operation and it supports a rich access control model that takes into account application semantics. The design of this protection device is based on well-know techniques – replication and proactive recovery – combined and extended in innovative ways, to solve the non-trivial problem of ensuring transparent intrusion-tolerant operation for standard protocols and components.

Additional results are the combination of reactive and proactive recovery in a single approach, without losing availability, and the first implementation of an intrusion-tolerant service in a single machine, using logical replication based on virtual machines. Preliminary experiments were conducted on the VM-based CIS prototype, in order to evaluate its behavior in face of attacks. The results exhibited an overall good performance in terms of latency, throughput and packet loss rate. Moreover, they helped us to realize that replication based on VMs must be used judiciously, since it implies less resources for each individual replica.

As future work, we expect to advance our understanding of the design and implementation of intrusion-tolerant firewalls, through the evolution of the VM-based prototype presented in this paper, and the implementation of a distributed version of CIS with physical replicas. Additionally, we will use the protection device presented in this paper as a building block of the CRUTIAL protection framework, which must take into account several other issues such as policy dissemination and consistency across different security domains.

Acknowledgments

We warmly thank Hans P. Reiser and Mário Calha for their valuable comments on improving this paper.

References

[1] A. Abou El Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *Proc. of 4th IEEE Int. Workshop on Policies for Distributed Systems and Networks – Policy'03*, June 2003.

[2] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Scaling byzantine fault-tolerant replication to wide area networks. In *Proc. Int. Conf. on Dependable Systems and Networks*, pages 105–114, June 2006.

[3] P. Barham, B. Dragovic, K. Fraiser, S. Hand, T. Harris, A. Ho, R. Neugebaurer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. of the 19th ACM Symp. on Operating Systems Principles - SOSP'03*, Oct. 2003.

[4] S. M. Bellovin. Distributed firewalls. *;login:*, Nov. 1999.

[5] G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing - PODC'84*, pages 154–162, Aug. 1984.

[6] British Columbia Institute of Technology. NISCC good practice guide on firewall deployment for SCADA and process control networks. Technical report, National Infrastructure Security Co-ordination Centre, London, UK, Feb. 2005.

[7] E. Byres, D. Hoffman, and N. Kube. The special needs of SCADA/PCN firewalls: Architectures and test results. In *Proc. of the 10th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, 2005.

[8] E. Byres and J. Lowe. The myths and facts behind cyber security risks for industrial control systems. In *Proc. of VDE Kongress*, 2004.

[9] C. Cachin and A. Samar. Secure distributed DNS. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'04)*, pages 423–432, 2004.

[10] M. Castro and B. Liskov. Practical Byzantine fault-tolerance and proactive recovery. *ACM TOCS*, 20(4):398–461, 2002.

[11] M. Chérèc, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron. Active replication in Delta-4. In *Proc. 22th Symp. on Fault-Tolerant Computing - FTCS'92*, pages 28–37, July 1992.

[12] M. Correia, N. F. Neves, and P. Verissimo. How to tolerate half less one Byzantine nodes in practical distributed systems. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems - SRDS 2004*, pages 174–183, Oct. 2004.

[13] M. Correia, P. Verissimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Proceedings of the Fourth European Dependable Computing Conference*, Toulouse, France, Oct. 2002.

[14] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures (extended abstract). In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology - CRYPTO'92*, pages 457–469, Aug. 1992.

[15] D. Dzung, M. Naedele, T. P. V. Hoff, and M. Crevatin. Security for industrial communication systems. *Proc. of the IEEE*, 93(6):1152–1177, 2005.

[16] L. H. Fink and K. Carlsen. Operating under stress and strain. *IEEE Spectrum*, Mar. 1978.

[17] J. Fraga and D. Powell. A fault- and intrusion-tolerant file system. In *Proceedings of the 3rd International Conference on Computer Security*, pages 203–218, 1985.

[18] F. Garrone (editor). CRUTIAL: Analysis of new control applications, Jan. 2007. Deliverable D2, EC project IST-2004-27513, http://crutial.cesiricerca.it.

[19] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson. 2006 CSI/FBI computer crime and security survey. Computer Security Institute, 2006.

[20] C. H. Hauser, D. E. Bakken, I. Dionysiou, K. H. Gjermundrød, V. S. Irava, J. Helkey, and A. Bose. Security, trust and QoS in next-generation control and communication for large power systems. *International Journal of Critical Infrastructures*, 2007. to appear.

[21] International Electrotechnical Commission. Inter control center protocol (ICCP). IEC 60870-6 TASE.2, 1997.

[22] International Electrotechnical Commission. Communication networks and systems in substations. IEC 61850, 2003.

[23] S. Kent. IP Authentication Header. RFC 4302 (Proposed Standard), Dec. 2005.

[24] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dec. 2005.

[25] V. Madani and D. Novosel. Getting a grip on the grid. *IEEE Spectrum*, 42(12):42–47, Dec. 2005.

[26] D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, Oct. 1998.

[27] M. A. Marsh and F. B. Schneider. CODEX: A robust and secure secret distribution system. *IEEE Transactions on Dependable and Secure Computing*, 1(1):34–47, Jan. 2004.

[28] R. R. Obelheiro, A. N. Bessani, L. C. Lung, and M. Correia. How practical are intrusion-tolerant distributed systems? DI-FCUL TR 06–15, Dep. of Informatics, Univ. of Lisbon, 2006.

[29] R. Oppliger. Security at the internet layer. *IEEE Computer*, 31(9):43–47, Sept. 1998.

[30] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *Proc. 10th ACM Symp. on Principles of Distributed Computing*, pages 51–59, 1991.

[31] H. P. Reiser and R. Kapitza. VM-FIT: Supporting intrusion tolerance with virtualisation technology. In *Proceedings of the Workshop on Recent Advances on Intrusion-Tolerant Systems*, March 2007.

[32] M. K. Reiter, M. K. Franklin, J. B. Lacy, and R. N. Wright. The omega key management service. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 38–47, 1996.

[33] F. B. Schneider. Implementing fault-tolerant services using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, Dec. 1990.

[34] P. Sousa. Proactive resilience. In *Sixth European Dependable Computing Conference (EDCC-6) Supplemental Volume*, pages 27–32, Oct. 2006.

[35] P. Sousa, N. F. Neves, A. Lopes, and P. Verissimo. On the resilience of intrusion-tolerant distributed systems. DI/FCUL TR 06–14, Dep. of Informatics, Univ. of Lisbon, Sept 2006.

[36] P. Sousa, N. F. Neves, and P. Verissimo. How resilient are distributed f fault/intrusion-tolerant systems? In *Proc. of Int. Conf. on Dependable Systems and Networks (DSN'05)*, pages 98–107, June 2005.

[37] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1), 1989.

[38] M. van Eeten, E. Roe, P. Schulman, and M. de Bruijne. The enemy within: System complexity and organizational surprises. In M. Dunn and V. Mauer, editors, *Int. CIIP Handbook 2006*, volume II, pages 89–110. CSS, ETH Zurich, 2006.

[39] P. Verissimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1), 2006, http://www.navigators.di.fc.ul.pt/docs/abstracts/ver06travel.html.

[40] P. Verissimo, N. F. Neves, and M. Correia. CRUTIAL: The blueprint of a reference critical information infrastructure architecture. In *Proc. of CRITIS'06 1st Int. Workshop on Critical Information Infrastructures Security*, Aug. 2006.

[41] P. Verissimo, N. F. Neves, and M. P. Correia. Intrusion-tolerant architectures: Concepts and design. In *Architecting Dependable Systems*, volume 2677 of *LNCS*. 2003.

[42] C. Wilson. Terrorist capabilities for cyber-attack. In M. Dunn and V. Mauer, editors, *Int. CIIP Handbook 2006*, volume II, pages 69–88. CSS, ETH Zurich, 2006.

[43] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement form execution for Byzantine fault tolerant services. In *Proc. 19th ACM Symp. on Operating Systems Principles - SOSP'03*, pages 253–267, 2003.

[44] L. Zhou, F. Schneider, and R. Van Rennesse. COCA: A secure distributed online certification authority. *ACM TOCS*, 20(4):329–368, Nov. 2002.

A Correctness of the Payload Protocol

This appendix proves that the CIS message processing protocol satisfies the conditions defined in Section 3.3, assuming the correctness of the wormhole protocols, which we prove in Appendix B. The proof starts with a lemma showing that the retransmission mechanism defined for the protocol ensures that a received message eventually will be signed.

Lemma 1 *In Algorithm 1, if a legal message is received by one correct CIS replica that is not recovered for long enough, it will eventually be signed.*

Proof (sketch): When a legal message m is received by a correct CIS replica, it will be verified by the policy engine (line 1), stored in the *Voting* set (line 2) and then passed to the wormhole for signing (line 3). Knowing that the W_sign service of the wormhole only returns at some replica if it is called by at least $f+1$ replicas, we will show that this call eventually returns. By the algorithm, if a replica stays more than T_{vote} blocked at line 3, the retransmission of line 19 will be triggered periodically (at each T_{vote} time units) until $Voting = \varnothing$ (which happens only if the replica executes line 4, after the wormhole signs the message). Since we assume that the unreliable multicast is fair (see Section 3.2), if a replica retransmits m many times, eventually all correct replicas will receive it. Given that $n \geq 2f+1$, there will be always $f+1$ correct replicas that will receive m and call $W\_sign(m)$. When this happens, the wormhole will produce the signature and return it to the payload. ■

Now we present theorems for the Validity and Integrity properties.

Theorem 1 *(*Validity*) Algorithm 1 ensures that a legal message received by at least one correct CIS replica that is not recovered for long enough, is forwarded to its destination.*

Proof (sketch): By Lemma 1, we know that a message m received by some correct replica will be signed. Since a message is signed only if $f+1$ processes approved it, we know that at least one correct replica that signed m will store it on the *Pending* set. The message will stay in this set until it is received in the LAN, i.e. it was forwarded (lines 12-14). While stored in this set m will be retransmitted periodically (line 19). After *Ot* retransmissions, the current leader replica will be suspected (line 22). Due to the fair broadcast assumption, all other correct replicas eventually will receive m and approve, sign and store it in their *Pending* sets. Consequently, if the message was not forwarded by the leader l, eventually $f+1$ replicas will suspect it, and the wormhole will change the leader replica. This procedure of leader change can happen at most f times until a correct replica becomes the leader, forwards *m*, and stores in its *Pending* buffer (lines 23-29).

The notion of a replica not being recovered for long enough should now be clear. Suppose only one correct replica CIS_i receives the message. In that case, CIS_i can not be recovered until other correct replicas, that also are not recovered for long enough, receive it. Notice, however, that the normal is for all correct replicas to receive the same messages, since the traffic replication device multicasts the messages to all replicas. ■

Theorem 2 *(*Integrity*) Algorithm 1 ensures that an illegal message is never processed by its destination machine.*

Proof (sketch): Recall that *(i.)* in our system model, the destination machine can not be corrupted and it only processes messages signed with key K; and *(ii.)* the key K is stored in the wormhole, and a message m is only signed with this key if at least $f+1$ CIS replicas call $W\_sign(m)$. Given these two facts, it is almost direct to see that illegal messages (approved by at most f processes) can not be signed (due to *(ii.)*) and, would not be processed by their destination machines even if forwarded by a malicious leader (due to *(i.)*). Consequently, an illegal message will never be processed by its destination machine. ■

B Correctness of the Wormhole Protocols

In this appendix we prove the correctness of the wormhole protocols.

Theorem 3 *(*Signature*) Algorithm 2 ensures that the $W\_sign(m)$ service returns a MAC produced with the shared key K for message m, if and only if at least $f+1$ CIS replicas called it in their local wormholes.*

Proof (sketch): When the service $W\_sign(m)$ is called at a local wormhole, it blocks until receiving a VOTE message with the correct MAC σ (produced with the shared key K for message m) from at least $f+1$ local wormholes (line 3). A VOTE message with the correct MAC σ is sent by a local wormhole if and only if $W\_sign(m)$ is called (lines 1 and 2), since we assume collision-resistant MACs. Consequently, the $W\_sign(m)$ service returns a MAC produced with the shared key K for message m, if and only if at least $f+1$ CIS replicas called it in their local wormholes. ■

Theorem 4 *(*Leader Change*) Algorithms 2 and 3 ensure that a leader i is changed if at least $f+1$ CIS replicas call either $W\_suspect\_silent(i)$ or $W\_suspect\_malicious(i)$.*

Proof (sketch): Consider that replica i is the leader and that at least $f+1$ CIS replicas call either $W\_suspect\_silent(i)$ or $W\_suspect\_malicious(i)$. When these services are called, a SUSPECT-SILENT or SUSPECT-MALICIOUS message is sent to replica i. Therefore, if the services are called by at least $f+1$ replicas, then at least $f+1$ suspect messages are sent with different sender ids (Algorithm 2, lines 8 and 9). The rest of the proof depends on the specific suspect service being called.

1. If at least $f+1$ replicas call $W\_suspect\_malicious(i)$, then at least $f+1$ different ids will be added to the set $Suspect_m$ of replica i (Algorithm 2, line 11). This immediately triggers the recovery of replica i (Algorithm 3, line 8). Given that replica i is the current leader, a new leader is elected (Algorithm 3, lines 14-15).

2. If at least $f+1$ replicas call either $W\_suspect\_malicious(i)$ or $W\_suspect\_silent(i)$, then at least $f+1$ different ids will be added to the union of the sets $Suspect_m$ and $Suspect_s$ of replica i (Algorithm 2, lines 10-11). This triggers the allocation of a slot and a subslot to perform a sporadic recovery (Algorithm 3, line 9). If the allocated slot corresponds to the one of replica i, then it means that replica i will be recovered through periodic proactive recovery. Otherwise, replica i will be recovered in the allocated slot. ■

Theorem 5 *(*Number of Recovering Replicas*) Algorithms 1, 2 and 3 ensure that at most one correct replica is recovering at any time.*

Proof (sketch): Consider a replica i. A recovery can only be triggered due to one of the following three reasons:

1. periodic proactive recovery (Algorithm 3, line 6);

2. reactive recovery because replica i is compromised (Algorithm 3, line 8);

3. reactive recovery because replica i is suspected to be silent (Algorithm 3, line 12).

We have to show that if replica i is recovered due to reasons 1 or 3, then it is guaranteed that no other correct replica is recovering at the same time. If replica i is recovered due to reason 2 then it is compromised, and thus nothing has to be proved.

If replica i is recovered due to a periodic proactive recovery (reason 1), then the proactive recovery algorithm (Algorithm 3, lines 1–7) guarantees that replica i always start recoveries T_{slot} time units after the previous periodic recovery. Given that $T_{slot} \geq T_D$ and that it is assumed than T_D is the maximum recovery duration time, then it is ensured that when a replica i starts a periodic recover, the previous periodic recovery has already finished. Moreover, the reactive recovery algorithm (Algorithm 3, lines 9–13) guarantees that replica i always start recoveries at least T_D time units after the previous reactive recovery. Therefore, it is ensured that when a replica *i* starts a proactive recovery, the previous proactive and reactive recoveries have already finished.

If replica *i* is recovered due to a reactive recovery because it is suspected to be silent (reason 3), then the reactive recovery algorithm (Algorithm 3, lines 9–13) guarantees that replica *i* always start recoveries at least T_D time units after the previous periodic recovery. Given that T_D is the maximum recovery duration time, then it is ensured that when a replica *i* starts a reactive recovery of this nature, the previous periodic recovery has already finished. Moreover, the reactive recovery algorithm guarantees that replica *i* always start recoveries at least T_D time units after the previous reactive recovery. Therefore, it is ensured that when a replica *i* starts a reactive recovery because it is suspected to be silent, the previous proactive and reactive recoveries have already finished. ■

Theorem 6 *The CIS protection service is exhaustion-safe (i.e., at most f replicas are failed at the same time) if it is possible to lower-bound the time needed for an attacker to produce $f+1$ replica failures by a known constant $T_{exh_{min}}$, and $T_P + T_D < T_{exh_{min}}$.*

Proof (sketch): In addition to T_P and T_D, the Proactive Resilience Model (*PRM*) [35] defines a third parameter T_π, but it only has meaning if the recovery is a distributed procedure (with a set of nodes recovering at the same time) and therefore it does not apply here given that CIS replicas recover one at a time. As shown in [35], if it is possible to lower-bound the time needed for an attacker to produce $f+1$ replica failures by a known constant $T_{exh_{min}}$, then it is guaranteed that no more than f replicas will be faulty at the same time as long as $T_P + T_D < T_{exh_{min}}$. ■