An Intrusion-Tolerant Web Server based on the DISTRACT Architecture[*]

Rafael Ferraz   Bruno Gonçalves   João Sequeira   Miguel Correia   Nuno F. Neves   Paulo Veríssimo
Faculdade de Ciências da Universidade de Lisboa
{*rferraz,bmg,jms*}*@lasige.di.fc.ul.pt*   {*mpc,nuno,pjv*}*@di.fc.ul.pt*

Abstract

The Web Server is currently the most widely deployed type of distributed data server. This paper presents an intrusion-tolerant web server based on the Deterministic IntruSion ToleRance ArChiTecture (DISTRACT), which is also introduced. The objective of this architecture is to support fault- and intrusion-tolerant services based on the state machine approach. DISTRACT uses a set of intrusion-tolerant protocols based on the TTCB, a secure and synchronous distributed component.

This paper reports on the first implementation of an intrusion-tolerant replicated service based on the TTCB. The solution proposed requires no modifications either on the clients or the servers, which are respectively web browsers and standard web servers. An evaluation of the performance of the replicated web server is provided.

1. Introduction

The Web Server is currently the most widely deployed type of distributed data server. This paper presents an intrusion-tolerant web server based on the *Deterministic IntruSion ToleRance ArChiTecture (DISTRACT)*, which is also introduced. The objective of this architecture is to support fault- and intrusion-tolerant services based on the state machine approach [20]. This approach consists in implementing a service as a set of replicas running a deterministic program and starting in the same state. Requests to the service are issued using an atomic multicast protocol, ensuring that all replicas execute the same commands in the same order.

The DISTRACT architecture contains a set of servers (or replicas) and a set of proxies (see Figure 1). The purpose of the proxies is to hide the replication from the service clients

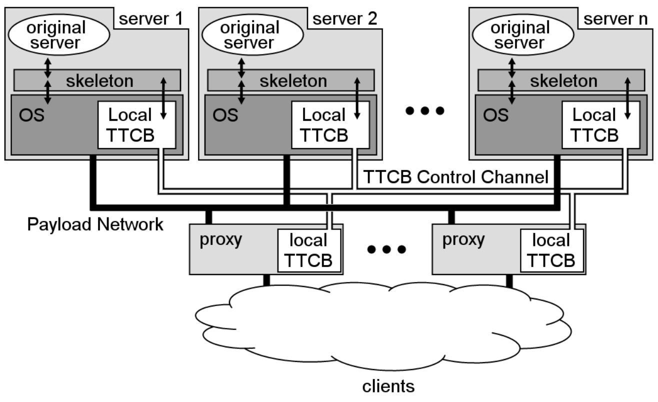

Figure 1. The DISTRACT Architecture

(e.g., the web browsers). A client issues a request to the service by sending a message to a proxy. This message is sent to the skeleton in each server, which calls the original server and replies to the proxy. When the proxy has a number of replies, it then replies to the client. The proxy machines can also include a firewall to provide a first line of defense for the servers.

DISTRACT supports fault- and intrusion-tolerant services, meaning that the services provide correct results despite the failure of some of the replicas. These failures include not only benign faults, like crashing, but also malicious faults, like a replica sending corrupted messages or failing to follow the protocol in some way. DISTRACT is based on an approach to intrusion tolerance [22] that we have been investigating for some years now. The idea is to rely on a secure and synchronous (i.e., real-time) distributed component to execute some "small" but crucial steps of the protocols. This component is called Trusted Timely Computing Base (TTCB) [9], and this kind of components have been called generically Wormholes [21]. DISTRACT uses a set of intrusion-tolerant protocols based on the TTCB.

The paper presents an intrusion-tolerant web server based on DISTRACT. Each replica is a PC running a standard, unmodified, web server. The proxies multicast the

[*] This work was partially supported by the FCT through project POSI/1999/CHS/33996 (DEFEATS), project POSI/CHS/39815/2001 (COPE) and the Large-Scale Informatic Systems Laboratory (LASIGE).

HTTP [10] requests they receive to the servers and reply to the calling browsers when they get a number of replies. One common attack to web servers that is tolerated in our system, is the defacement of web pages.

This paper reports on the first full implementation of an intrusion-tolerant replicated service based on a wormhole. An evaluation of its performance is also provided. There are three other intrusion-tolerant services based on the state machine approach in the literature: Rampart [18], BFT [4] and FS-NewTOP [16]. The system proposed in this paper has mainly three advantages: (1) the intrusion-tolerance of the service is transparent to the clients, which are not modified in any way; (2) the servers can be standard unmodified different servers, which are accessed transparently through a skeleton; (3) DISTRACT has no notion of a leader/master, therefore it is not possible to delay the service by causing successive suspicions on leaders, a problem existing in both Rampart and BFT.

The support for different servers is important since replication is useful only if replicas fail independently. With malicious faults, this requires that different servers have different vulnerabilities. In practice, this has to be achieved by using different operating systems, servers and skeletons in each machine. Distinct versions of the code might be obtained using N-version programming [1].

There are some other related work in intrusion-tolerant services. PASIS uses quorum systems to support intrusion-tolerant storage systems that also guarantee confidentiality [12]. COCA is a secure and fault-tolerant certification authority also based on quorums [23]. AgileFS is an intrusion-tolerant file system based on secret-sharing [13]. SINTRA is a solution to implement replicated services using randomized protocols [2].

We have recently shown that using a wormhole it is possible to tolerate $f = \lfloor \frac{n-1}{2} \rfloor$ faulty replicas, where n is the number of replicas [8]. This paper uses a previous set of protocols, which tolerate only $f = \lfloor \frac{n-1}{3} \rfloor$ faulty replicas. However, DISTRACT would support the improved resilience with a simple substitution of protocols.

2. System Model

The system architecture is depicted in Figure 1. With the exception of the TTCB, the system is asynchronous, i.e., there are no assumptions about bounds on the communication or processing delays. The system is also essentially Byzantine on failure, i.e., the servers and the clients can fail arbitrarily: they can stop, omit messages, send incorrect messages, send several messages with the same identifier, collude with other malicious processes to break the protocols, etc. Despite this failure model, we assume that the number of servers that can fail is limited to $f = \lfloor \frac{n-1}{3} \rfloor$. Moreover, we assume that the proxies can only fail by crashing, not arbitrarily. This assumption is reasonable due to the simplicity of the proxy. We discuss how it can be enforced in Section 5. A client, server or proxy that does not fail during the execution of a protocol is said to be *correct*.

```
int getTCBGlobalTimestamp(descriptor_t *d,
TCBtimestamp *gts);

void propose(descriptor_t *d, elist_t elist,
TCBtimestamp tstart, decision_t decision,
value_t value, propose_out_t *out);

void decide(descriptor_t *d, tag_t tag,
decide_out_t *out);
```

Figure 2. Excerpt of the TTCB C-language API

Besides the networks and hosts, the system includes the TTCB wormhole. This component is distributed: it has local parts in the hosts (local TTCBs) and its own private communication channel (the TTCB control channel). The TTCB, both its local parts and control channel, are assumed to be secure, i.e., resistant to any possible attack. In this short paper we simply consider this to be an assumption. How it can be enforced in practice, is the matter of another paper [9].

The TTCB provides just a few services. The protocols used by DISTRACT use mostly two: the Trusted Absolute Timestamping service and the Trusted Block Agreement service (TBA). The first provides timestamps, i.e., readings from the local TTCB clock. The other service, the TBA, makes agreements among sets of processes. Each process proposes a value with a limited size (currently 20 bytes), and the TBA returns a value obtained applying a function to the values proposed. The protocols in the paper use two of these functions: one that chooses the value proposed by the first process; and another one that chooses the most proposed value. The API of the two services is depicted in Figure 2.

3. DISTRACT

The DISTRACT architecture is presented in Figure 1 and briefly discussed in the introduction. Here we delve into some of the details.

A client contacts the service by sending a message with a request to one of the proxies, using the application protocol. The case in which the proxy is unavailable is also handled in an application-dependent way. In Section 4 we discuss the specific case of the web server.

The proxies and the servers communicate using secure channels, which ensure that the messages are not corrupted in the network and are eventually received. The channels are end-to-end but a multicast can be emulated using one channel for each recipient. This multicast does not guarantee that all recipients receive the same messages, since a malicious sender can send different messages for each recipient.

The state machine approach requires that all servers deliver the same requests in the same order. These properties are guaranteed by a protocol similar to the *atomic multicast* in [6]. When a proxy receives a request R, it obtains a timestamp ts from the TTCB, and multicasts *(MCAST, R, ts)* to all the servers, using a set of secure channels (*MCAST* is the message type). The order of the messages is defined by the timestamp ts, but the servers also have to agree on the set of messages to order and deliver. The protocol is sketched in Figure 3 for the skeleton in server s_i. The protocol has two states: Normal and Agreement. When the system is idle, it is in the first state. The second state corresponds to the execution of the agreement about the delivery of a set of messages (PICK). This agreement is similar to the consensus protocol in [7].

A server decides to engage in the PICK protocol using a combination of two criteria (COLLECT, line 3): when it received a number of messages Na; or when it has at least one message and an interval of time Ta passed from the last agreement termination (this condition is tested using the TTCB timestamping service). We have omitted this second condition in the protocol for simplicity. The agreement protocol usually runs a single TBA, although more can be executed until $2f + 1$ servers engage in the protocol (PICK, line 2).

When the agreement protocol terminates, each server executes the agreed upon requests in the order of their corresponding timestamps ts. When each request is executed by each server, a reply is returned to the proxy. When a proxy has $f + 1$ identical replies from different servers, it can be sure that this is the correct reply (at most f servers can be malicious), so it forwards the reply to the client.

The service replicas can be standard servers, like the Apache servers[1] used in our intrusion-tolerant web server prototype. The server side of the protocols is executed by the skeleton in each host, which locally calls the standard server.

4. Web Server

The implementation of an intrusion-tolerant web server using DISTRACT is straight forward. Each server replica is an Apache web server. Each proxy emulates a web server for the clients, which are browsers. When a browser sends a request to the server using the HTTP protocol over TCP, the proxy receives the request and multicasts it to the skeletons in the servers. When it has the $f + 1$ replies from the servers, it sends an HTTP reply transparently to the browser.

1 http://www.apache.org/

COLLECT

1. When *(MCAST, R, ts)* is received, multicast *(INFO,* s_i*, R, ts)* to all servers;
2. When $f + 1$ *(INFO, *, R, ts)* messages received from different servers, if s_i did not multicast *(INFO,* s_i*, R, ts)*, multicast it to all servers;
3. When $2f+1$ INFO messages received from different servers for Na requests, if the state is Normal, put the state in Agreement and start the PICK protocol;
4. When protocol PICK terminates, execute all requests in the order of their timestamps ts; Put the state to Normal;

PICK

1. Propose TBA a hash of the requests with $2f+1$ messages received; Wait until *decide* returns the result of the TBA (most proposed value);
2. Repeat the TBA until at least $2f + 1$ servers propose;
3. If s_i proposed the hash that was decided, multicast a message with the corresponding requests to all servers; return;
4. Otherwise, wait for a message with the requests corresponding to the hash; return;

Figure 3. Protocols executed by server s_i

In the Internet, a service is usually known by its domain name, which is translated into an IP address by the Domain Name System (DNS) [15]. The DNS can have several IP addresses associated to a single domain name, returning each IP according to some policy. This is the mechanism that DISTRACT assumes to be used: the domain name of the service should be associated to the IPs of the proxies in the DNS. If a client obtains from the DNS the IP of a proxy inaccessible for some reason (e.g., crashed), it has to ask for another IP and resend the request. This happens frequently in today's Web, and is done simply by refreshing the browser.

State machine replication requires that the state machines are deterministic, i.e., that the same command executed in the same initial state generates the same final state and returns the same result. For web servers, with static web pages the first part is obviously true. If the pages are generated dynamically, they cannot depend on three items that can generate non-determinism: random data, time and machine specific information (e.g., IP address or host name). In relation to the second part, several web servers with the same pages, usually return slightly different replies to identical requests. The differences occur in three pieces of data in the header of the reply [10]: (1) the date stamp (*Date*); (2) the entity tag (*Etag*), which uniquely identifies the entity, i.e., the HTTP reply; (3) the server identification (*Server*). An example header can be found in Figure 4. The prox-

```
HTTP/1.1 200 OK
Date: Thu, 27 Feb 2003 18:08:30 GMT
Server: Apache/2.0.50 (Unix)
Last-Modified: Thu, 27 Feb 2003 18:08:30 GMT
ETag: W/"144332-65-af1fe480"
Accept-Ranges: bytes
Content-Length: 101
Keep-Alive: timeout=15
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
```

Figure 4. Example HTTP reply header

ies have to wait for $f+1$ identical replies from the servers, as discussed above. To solve this problem of determinism, the proxies simply discard the three lines of the header, and compare only the rest of the reply. Notice that the proxy can be adapted for many different TCP/IP services with minor changes. The single change in the skeleton would be the port of the service.

The security of the communication between the clients and the proxies is not part of DISTRACT, but application-dependent. In the case of the web server, the communication with HTTP is not secure. However, there is a standard solution for securing it, which is to use HTTP over TLS [19]. This would involve a proxy that supported this protocol.

5. Implementation

All the system components were implemented in C language. The proxies are assumed to be secure but this property can be ensured with a high coverage for two reasons: (1) the software is simple, currently approximately 700 lines of code; (2) the host with the proxy needs to have only the TCP port 80 open, since the communication with the clients and servers is all done using HTTP (HTTP over TLS was not implemented). This sums up to a component simple enough to protect using typical techniques: minimizing the number of operating system components; closing all network services and disabling all ports except TCP/80; patching known vulnerabilities; and assuring that all inputs are validated.

The TTCB is also assumed to be secure, therefore it has to be isolated from the rest of the system [9]. Recall the TTCB architecture, represented in white in Figure 1. In the current implementation, the local TTCB resides inside the kernel, which is hardened in order to be secure. The solution is essentially to remove some Linux capabilities from the *capability bounding set*, thus preventing the superuser from accessing the kernel memory. This version of the local subsystem has less coverage than a solution using a hardware appliance, e.g., a PC/104 board with its own processor and memory, but has the advantage of allowing the free distribution of the implementation by the research community[2]. The control channel is a Fast-Ethernet LAN, which can be can be assumed to be secure since it is a short-range inside-premises closed network. The implementation of the local TTCB requires a real-time kernel, in order to be synchronous. The current prototype uses RTAI [5][3], a real-time engineering of Linux that runs on standard PC hardware. The local TTCB is composed by a set of modules executed inside this kernel. The control-channel has also to be predictable in terms of time behavior, something that can be enforced by controlling the amount of traffic [3].

The DISTRACT protocols are executed on the top of secure channels. In the current implementation these channels are implemented using the Secure Socket Layer v3 (SSL) [11]. The implementation used is OpenSSL[4]. The protocols use the MD5 hash function [14].

6. Performance

The performance was assessed using four replicas ($n = 4$, $f = 1$) and a single proxy, each one running on an Intel Pentium III 500Mhz PC with 256MB SDRam PC133. Each PC had two 3Com 10/100 network adapters. The payload network and the TTCB control channel were both Fast-Ethernet 100Mb switched LANs. The versions of the software used were: TTCB 1.11, RTAI 24.1.10, Apache 2.0.50, OpenSSL 0.9.7d and C compiler gcc 2.96. SSL used 128 bit-keys with stream cipher RC4 and hash function MD5.

The experiments were performed using a pseudo-browser that we designed with the purpose of sending HTTP requests and receiving replies, while measuring latency and throughput. Every experiment involved at least 1000 requests. Each request was sent by the pseudo-browser to the proxy, which multicasted it to the replicas, waited for $f + 1 = 2$ identical replies, and finally sent the web page back to the pseudo-browser. All experiments were performed with no failed replicas, except experiment 4.

Experiment 1 consisted in measuring the throughput of the service with a single client and a page of 100 bytes (Figure 5). The requests were sent sequentially by the client. The parameter Na changed from 1 to 21 requests, i.e., the replicas waited for those numbers of requests to start the PICK protocol. The figure shows that the throughput slightly improves with Na. This result may seem counterintuitive since, apparently, the higher the value of Na, the longer the replicas would have to wait to start PICK. However, when the rate of requests is high, the time PICK takes to run is enough to receive more than Na requests, so

2 Available at: http://www.navigators.di.fc.ul.pt/software/ttcb/

3 http://www.rtai.org/

4 http://www.openssl.org/

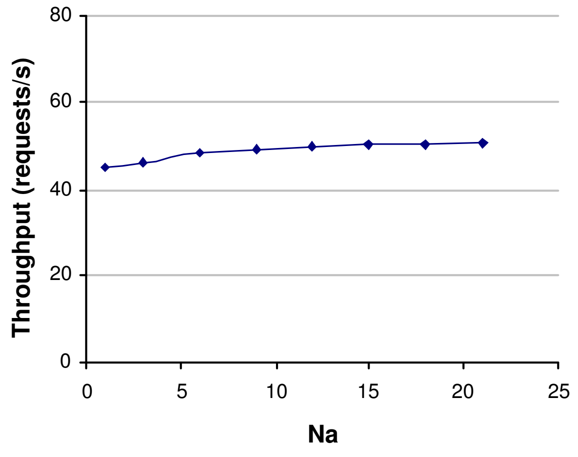

Figure 5. Experiment 1 (100 byte-pages)

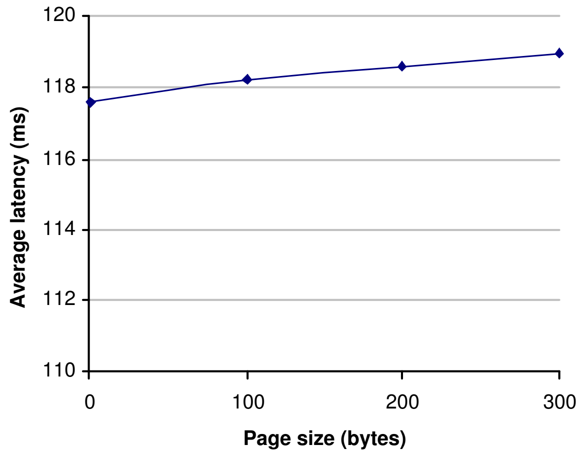

Figure 7. Experiment 3 ($Na = 6$)

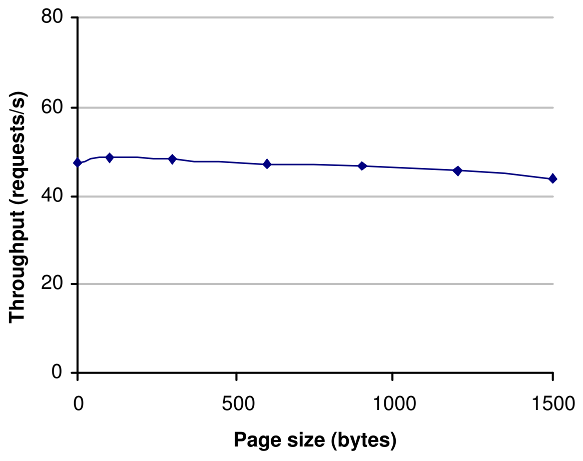

Figure 6. Experiment 2 ($Na = 6$)

when PICK terminates another one starts almost immediately. Therefore, what constrains the throughput is the rate of requests the proxy can multicast, which is constrained by the rate of TBAs that can be executed by the TTCB.

Experiment 2 is similar to experiment 1 but Na was set to 6 and the web page size was varied from 0 to 1500 bytes. Figure 6 shows that the throughput reduces slightly with the page size. From 100 to 1500 bytes the size of the page increased 10 times but the impact on the throughput was low. The reason for this behavior is that the constraint for the throughput is not the bandwidth or the server processing power as in typical Internet services, but the rate of TBAs that the TTCB can execute and the cryptographic operations (SSL channels and hashes in the multicast and PICK protocols).

Experiment 3 evaluated the average latency of a request (Figure 7). There was a single client, $Na = 6$ and the page size varied from 0 to 300 bytes. Once more, the times are not considerably affected by the page size for the same reasons as in experiment 1.

Experiment 4 measured the average latency and the throughput when a server is silent, i.e., does not interact with the protocol either because it crashed or an intrusion occurred. With a message of 100 bytes and $Na = 6$, the average latency was 195 ms, considerably worse than the case when all servers were correct. The cause of this delay was the TBA service, that usually runs considerably faster when all processes propose a value [9]. The throughput was 39.9 requests per second, which is slightly below the value when all servers are correct.

All these values are considerably worse than the values we measured for a non-replicated single Apache server with pages of 100 bytes: 1400 requests per second. This is one of the costs of making the server intrusion-tolerant, using complex protocols, several servers, a proxy mediating each request/reply, and cryptographic primitives.

Comparing the performance of different systems is complicated since the test settings are usually different. In this case, our results can not even be compared directly with others in the literature because we have an additional intermediary, the proxy, which introduces an additional delay. However, for the reader to have an idea, the throughput obtained is better than Rampart's [17] but worse than BFT's [4]. Previous measurements we have made with other prototypes show us that the current DISTRACT implementation can still be considerably optimized.

7. Discussion

The paper presents an architecture for intrusion-tolerant services based on the state machine approach, DISTRACT. The paper reports on our experience implementing an intrusion-tolerant web server using this architecture. The solution shows several interesting benefits: standard clients and servers can be used without modification; and server diversity is supported transparently. The implementation can be adapted for many TCP/IP services with minor changes. The current performance results look promising when compared with similar systems.

Future work will be pursuit in implementing more efficient wormholes that can support faster services. The prototype will also be made more efficient, by redesigning some parts of the code.

References

[1] A. Avizienis. The N-version approach to fault tolerant software. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, Dec. 1985.

[2] C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the Internet. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 167–176, June 2002.

[3] A. Casimiro, P. Martins, and P. Veríssimo. How to build a Timely Computing Base using Real-Time Linux. In *Proceedings of the IEEE International Workshop on Factory Communication Systems*, pages 127–134, Sept. 2000.

[4] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, Nov. 2002.

[5] P. Cloutier, P. Mantegazza, S. Papacharalambous, I. Soanes, S. Hughes, and K. Yaghmour. DIAPM-RTAI position paper. In *Real-Time Linux Workshop*, Nov. 2000.

[6] M. Correia, N. F. Neves, L. C. Lung, and P. Veríssimo. A wormhole-based intrusion-tolerant group communication system - WIT-GCS. In *The 5th Cabernet Plenary Workshop*, Nov. 2003.

[7] M. Correia, N. F. Neves, L. C. Lung, and P. Veríssimo. Low complexity Byzantine-resilient consensus. *Distributed Computing*, 2004. To appear.

[8] M. Correia, N. F. Neves, and P. Veríssimo. How to tolerate half less one Byzantine nodes in practical distributed systems. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems*, Oct. 2004.

[9] M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Proceedings of the Fourth European Dependable Computing Conference*, pages 234–252, Oct. 2002.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. IETF Request for Comments: RFC 2068, Jan. 1997.

[11] A. Frier, P. Karlton, and P. Kocher. The SSL 3.0 protocol. Netscape Communications Corp., Nov. 1996.

[12] G. Goodson, J. Wylie, G. Ganger, and M. Reiter. Efficient Byzantine-tolerant erasure-coded storage. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, June 2004.

[13] S. Lakshmanan, M. Ahamad, and H. Venkateswaran. Responsive security for stored data. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, May 2003.

[14] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[15] P. Mockapetris. Domain names – concepts and facilities. IETF Request for Comments: RFC 1034, Nov. 1987.

[16] D. Mpoeleng, P. Ezhilchelvan, and N. Speirs. From crash tolerance to authenticated Byzantine tolerance: A structured approach, the cost and benefits. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 227–236, June 2003.

[17] M. Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80, Nov. 1994.

[18] M. K. Reiter. The Rampart toolkit for building high-integrity services. In *Theory and Practice in Distributed Systems*, volume 938 of *Lecture Notes in Computer Science*, pages 99–110. Springer-Verlag, 1995.

[19] E. Rescorla. HTTP over TLS. IETF Request for Comments: RFC 2818, May 2000.

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

[21] P. Veríssimo. Uncertainty and predictability: Can they be reconciled? In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 108–113. Springer-Verlag, 2003.

[22] P. Veríssimo, N. F. Neves, and M. Correia. Intrusion-tolerant architectures: Concepts and design. In R. Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems*, volume 2677 of *Lecture Notes in Computer Science*, pages 3–36. Springer-Verlag, 2003.

[23] L. Zhou, F. Schneider, and R. van Renesse. COCA: A secure distributed on-line certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, Nov. 2002.