# Hardware Support for High Performance, Intrusion- and Fault-Tolerant Systems

G. P. Saggese†, C. Basile†, L. Romano*, Z. Kalbarczyk†, R. K. Iyer†
†University of Illinois at Urbana-Champaign 1308 W. Main St., 61801 Urbana, Illinois
*Università degli Studi di Napoli Federico II Via Claudio 21, 80125 Napoli, Italy

*Abstract*— **The paper proposes a combined hardware/software approach for realizing high performance, intrusion- and fault-tolerant services. The approach is demonstrated for (yet not limited to) an Attribute Authority server, which provides a compelling application due to its stringent performance and security requirements. The key element of the proposed architecture is an FPGA-based, parallel *crypto-engine* providing (1) optimally dimensioned *RSA Processors* for efficient execution of computationally intensive RSA signatures and (2) a *KeyStore* facility used as tamper-resistant storage for preserving secret keys. To achieve linear speed-up (with the number of RSA Processors) and deadlock-free execution in spite of resource-sharing and scheduling/synchronization issues, we have resorted to a number of performance enhancing techniques (e.g., use of different clock domains, optimal balance between internal and external parallelism) and have formally modeled and mechanically proved our crypto-engine with the Spin model checker. At the software level, the architecture combines *active replication* and *threshold cryptography*, but in contrast with previous work, the code of our replicas is multithreaded so it can efficiently use an attached parallel crypto-engine to compute an Attribute Authority partial signature (as required by threshold cryptography). Resulting replicated systems that exhibit nondeterministic behavior, which cannot be handled with conventional replication approaches. Our architecture is based on a *Preemptive Deterministic Scheduling* algorithm to govern scheduling of replica threads and guarantee strong replica consistency.**

## I. Introduction

Combining intrusion and fault tolerance is an effective approach to handle security and reliability issues and has attracted significant research interest [1]–[4]. In meeting security and reliability requirements, however, existing solutions often sacrifice performance, a loss that is not acceptable for many critical applications (e.g., e-commerce, e-procurement). Also, most of the security mechanisms proposed are purely software based, which simplifies design and implementation but reduces resilience to security attacks [5]. In an attempt to improve security, smart-cards have been proposed as tamper-resistant devices to implement access control mechanisms [6]. Current smart-card technology, however, provides quite limited computational and storage capabilities; moreover, its tamper-resistance property has been questioned by experimental investigations [7].

This study leverages current research on intrusion- and fault-tolerant architectures and combines software approaches with the use of reconfigurable hardware devices to provide substantially improved performance and security. While it is clear that a hybrid approach can be superior to a software-only approach (e.g., our experiments show about an order of magnitude in speed-up), the effects on an overall system architecture are less understood. Consider, for instance, that the efficient combination of parallel hardware with multithreaded software can result in systems exhibiting nondeterministic behavior, which cannot be handled with conventional replication approaches (such as the Byzantine dissemination quorums used in COCA [2]).

Our approach is demonstrated in (yet not limited to) the context of attribute certification systems [8], [9], which provide a compelling application due to their stringent performance and security requirements. Specifically, this paper presents the design, implementation, and evaluation of a distributed, RSA-based Certificate Engine (the core element of an Attribute Authority) that can tolerate both accidental and malicious faults yet provide high performance. (The concepts and the techniques we propose also apply to RSA-based Certification Authorities, since the procedures for assembling and signing certificates are quite similar [10], [11].)

The key component of our architecture is a hardware *crypto-engine* that integrates, in a single FPGA device, a large number of *RSA Processors* to accelerate computationally expensive RSA operations and a *tamper-resistant KeyStore* to preserve secret keys; this is done seamlessly with threshold-cryptography support. Implementing RSA Processors and the KeyStore in a single chip provides significant improvement in security and performance. (A secret key kept in the KeyStore is directly accessed by the RSA Processors without ever being transfered outside the FPGA device.) While the crypto-engine approach might seem straightforward in principle, serious technical challenges must be overcome to provide an actual implementation. A solid design must provide linear speed-up (with the number of RSA Processors) and deadlock-free execution in spite of resource-sharing and scheduling/synchronization of the multiple units executing concurrently. To achieve these goals, we have resorted to a number of performance enhancing techniques (e.g., use of different clock domains, optimal balance between internal and external parallelism) and have formally modeled and mechanically proved our crypto-engine with the Spin model checker [12]. In addition, our crypto-engine design is general and can serve a broad range of security applications (e.g., SSL connection establishment, elliptic curve operations).

At the software level, the proposed architecture combines active replication and threshold cryptography to detect and

Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)
1060-9857/04 $ 20.00 IEEE
Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 12,2026 at 01:17:31 UTC from IEEE Xplore. Restrictions apply.

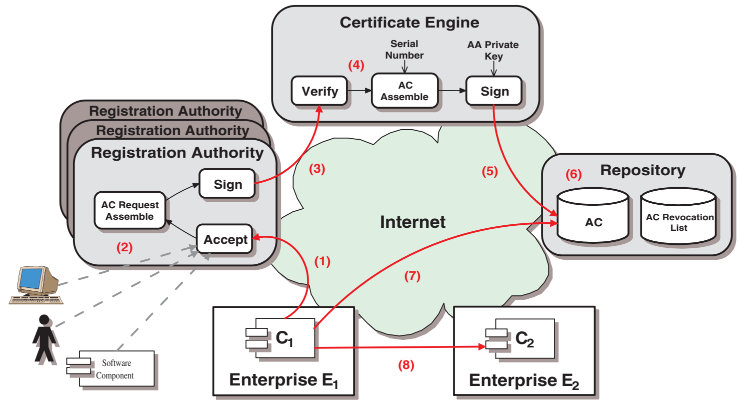
mask a minority of faulty and/or compromised replicas (i.e., replicas whose results are invalid due to accidental errors or replicas whose secret key shares have been stolen/modified by a malicious attacker). In contrast with previous work [2]–[4], the code of our replicas is multithreaded so it can efficiently use an attached parallel crypto-engine to compute an Attribute Authority partial signature (as required by threshold cryptography). The use of multithreading introduces replica nondeterminism, which we handle by employing the *Preemptive Deterministic Scheduling (PDS)* algorithm to govern scheduling of replica threads and guarantee strong replica consistency [13]. The resulting multithreaded replication scheme incorporates threshold cryptography and is based on a *fanout/combiner* component, which in addition to performing majority voting on replica outputs, assembles replica partial signatures to generate Attribute Authority signatures.

## II. Case Study: Intrusion- and Fault-Tolerant Attribute Authority

Digital certificates based on public key cryptography come in two flavors: *Public Key Certificates (PKCs)* and *Attribute Certificates (ACs)*. A PKC is an attestation that specifies a binding between the identity of a principal (i.e., an individual or a hardware/software component, which is also referred to as the certificate's subject) and the public key associated with the principal's private key. An AC is an attestation that specifies a binding between an attribute (i.e., a piece of authorization information about a principal) and a PKC issued for that principal. A digital certification system is a distributed system that handles a digital certificate issue, revocation, and query. The three major components of the system are a Registration Authority, a Certificate Engine, and a Repository. The *Registration Authority* is the system's front-end; it interacts with certificate applicants and creates certificate requests on behalf of the applicants. The *Certificate Engine* is the core of the system and issues digital certificates upon request of the Registration Authority. The *Repository* publishes both a list of the currently valid certificates and a list of revoked certificates. Digital certification systems for PKCs and ACs are usually referred to as *Certification Authorities* and *Attribute Authorities*, respectively [9]–[11]. Figure 1 illustrates the structure and operation of an Attribute Authority.

Among an Attribute Authority's components, the Certificate Engine is the most critical, as it owns the Attribute Authority secret keys and is heavily used by multiple Registration Authorities. In addition, the Certificate Engine is subjected to intensive loads because (1) attribute certificates have, in general, a short lifetime, since they correspond to temporary privileges of the applicant (as opposed to PKCs, which correspond to permanent characteristics, such as identity), and (2) the population of certificate applicants can be quite large (e.g., consider distributed, e-commerce applications).

This study focuses on the design, implementation, and evaluation of a high performance, intrusion- and fault-tolerant, RSA-based Certificate Engine for Attribute Authorities. The focus on RSA cryptography is justified by its vast acceptance




1) Principal $C_1$ applies for an attribute certificate (AC).
2) The Registration Authority acts on behalf of the certificate applicant and signs a certificate request using its own private key.
3) The signed certificate request is sent to the Certificate Engine.
4) The Certificate Engine verifies the authenticity and integrity of the received request (checking the included signature) and creates an AC signed with the Attribute Authority's private key.
5) The issued AC is sent to the Repository.
6) The Repository publishes the received AC via a directory service. The AC is then available to all certificate users (including the AC's applicant) for retrieving and query.
7) $C_1$ retrieves the requested AC from the Repository.
8) $C_1$ sends an electronic order message to $C_2$, enclosing the obtained $AC$. $C_2$ verifies validity of the received message and the enclosed AC to complete transaction with $C_1$.

Fig. 1. High-Level Architecture and Operation of an Attribute Authority.

in commercial applications, while the focus on Attribute Authorities is due to their more stringent performance requirements as compared to Certification Authorities (as discussed above). Nevertheless, the concepts and the techniques we propose also apply to RSA-based Certification Authorities, since the procedures for assembling and signing PKCs and ACs are quite similar [10], [11].

The proposed Certificate Engine architecture (depicted in Figure 2) combines an FPGA-based crypto-engine (described in § III) with active replication and threshold cryptography. In a pure replication approach, Certificate Engine replicas are perfectly identical, and hence, all use the Attribute Authority's secret key. By attacking a single replica, an adversary can obtain the Attribute Authority's secret key and compromise the whole system. In our hybrid approach, Certificate Engine replicas are not entirely identical as they use different secret key shares. Threshold cryptography guarantees that an adversary can obtain the Attribute Authority's secret key only by compromising a majority of replicas.

In contrast with previous work [2]–[4], the code of our replicas is multithreaded so it can efficiently use an attached parallel hardware crypto-engine to compute an Attribute Authority partial signature (as required by threshold cryptography). This scheme improves replica throughput linearly with the number of available RSA Processors. As a side effect, however, multithreading causes nondeterminism in replica behavior. Replica determinism is necessary to guarantee that certificates issued by different replicas have the same unique serial number, and hence, that the overall system can adhere to the X.509 standard [10]. We overcome this problem by


Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 12,2026 at 01:17:31 UTC from IEEE Xplore. Restrictions apply.

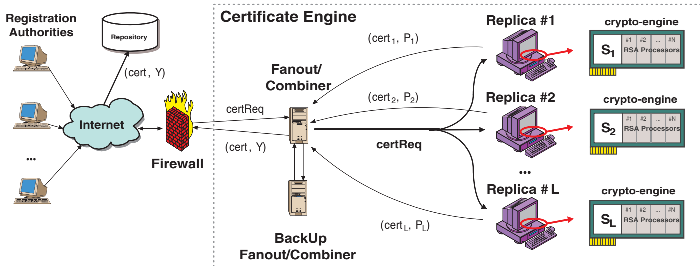



Fig. 2. Architecture of the High Performance, Intrusion- and Fault-Tolerant Certificate Engine.

employing the *Preemptive Deterministic Scheduling (PDS)* algorithm [13] to schedule replica threads and guarantee strong replica consistency.

An integral part of the proposed architecture is a PDS-enabled *fanout/combiner* process, which votes on replica outputs (on a per-thread basis) and detects and excludes faulty/compromised replicas. Specifically, the fanout/combiner is responsible for (1) reliably broadcasting a certificate request ($certReq$) to all replicas, (2) collecting each replica $i$'s output ($cert_i, P_i$), which includes a certificate $cert_i$ and its partial signature $P_i$, [1] (3) performing majority voting on the collected certificates to determine the agreed certificate $cert$ and to detect and exclude faulty replicas, (4) assembling partial signatures $P_i$ to reconstruct a valid Attribute Authority signature $Y$ of the agreed certificate $cert$, (5) verifying the reconstructed $Y$ to identify and terminate compromised replicas, and (6) communicating the agreed certificate $cert$ with its signature $Y$ to the Repository (from which it can be retrieved by the certificate applicant).

## III. Design and Implementation of the Crypto-engine

Performing a cryptographic operation in software can be quite computationally intensive,[2] and dedicated hardware coprocessors provide an efficient alternative. To the best of our knowledge, current commercial security coprocessors can only perform one cryptographic operation at a time and do not support threshold cryptography directly [14]–[16]. Such shortcomings limit their applicability to high-performance, intrusion-tolerant systems.

This section proposes a hardware crypto-engine that integrates, in a single chip, a large number of *RSA Processors* to accelerate computationally expensive RSA operations, and a *tamper-resistant KeyStore* to preserve (shares of) secret keys. The integration is done seamlessly with threshold-cryptography support by using Shoup's algorithm [17] in our architecture. In contrast with commercial products, our implementation is based on FPGA technology, which is an effective choice for low-volume and low-cost embodiments. Nonetheless, the proposed architecture can be readily realized in ASIC technology, if appropriate development resources are available.

The crypto-engine architecture is general and can serve a broader range of security applications (e.g., SSL connection establishment, elliptic curve operations). Indeed, the internal design implements generic functions that any cryptographic coprocessor is likely to require (e.g., dispatching operation requests to multiple functional units, loading secret keys in the device). Also, as a modular design approach, the implementation of the different communication protocols among the several hardware entities (e.g., RSA Processors, KeyStore, Memory Interface) is abstracted under a single, coherent interface that comprises the same control signals (start/end handshake pairs) and the same registers to exchange parameters.

### A. Crypto-engine Architecture

The crypto-engine architecture (depicted in Figure 3) is centered on a *Main Controller* component, which supervises the operation of the other crypto-engine components (e.g., RSA Processors and KeyStore) and the communication with the host system (i.e., the computer system hosting the crypto-engine). Our design is heavily based on a number of performance-enhancing techniques such as the use of different clock domains—to allow the several parts of the system to work at their maximum speeds—and optimal balancing between internal parallelism and functional parallelism.

**RSA Processor**. An RSA Processor decomposes modular exponentiation, involved in an RSA operation, into a series of modular multiplications and squares, which are efficiently computed employing the Montgomery algorithm [18]. The processor data-path includes (1) a P-processor, computing modular multiplications, (2) a Z-processor, computing modular squares, and (3) a register file, used to store partial results (see Figure 4). The RSA Processor operates serially on words of $S$ bits, where $S$ is a design parameter that can be chosen from among the values 32, 64, 128, and 256. This approach (1) makes the processor design modular and scalable with the length of the RSA modulus and exponents (thus enabling the use of threshold cryptography) and (2) allows trading-off performance versus area occupation, which can be used when dimensioning the crypto-engine (see § VI). An early RSA Processor was introduced in [19]. Major extensions were necessary to integrate it into the full crypto-engine architecture and to manage long RSA exponents as required by the threshold cryptography algorithm adopted [17].

**KeyStore**. The KeyStore component provides tamper-resistant storage and fast access to a number of RSA keys used by the RSA Processors. Each KeyStore entry includes a modulus $N$, a generic exponent $Exp$, and a factor $W$. Exponent $Exp$ can be the public exponent $E$, the private exponent $D$ of a standard RSA key, or a secret key share $S_i$ [17]. Factor $W$ depends only on $N$ and is used to convert the input data in the residue representation needed by the Montgomery algorithm [18]. The KeyStore can accommodate multiple key entries, and this feature can be used when

[1] $P_i$ is the signature of $cert_i$ computed by using share $S_i$.

[2] For instance, on an AMD Athlon 2600 XP+ at 2 GHz, an RSA signature of 1024-bit data with a 1024-bit key takes approximately 100 ms.


Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 12,2026 at 01:17:31 UTC from IEEE Xplore. Restrictions apply.

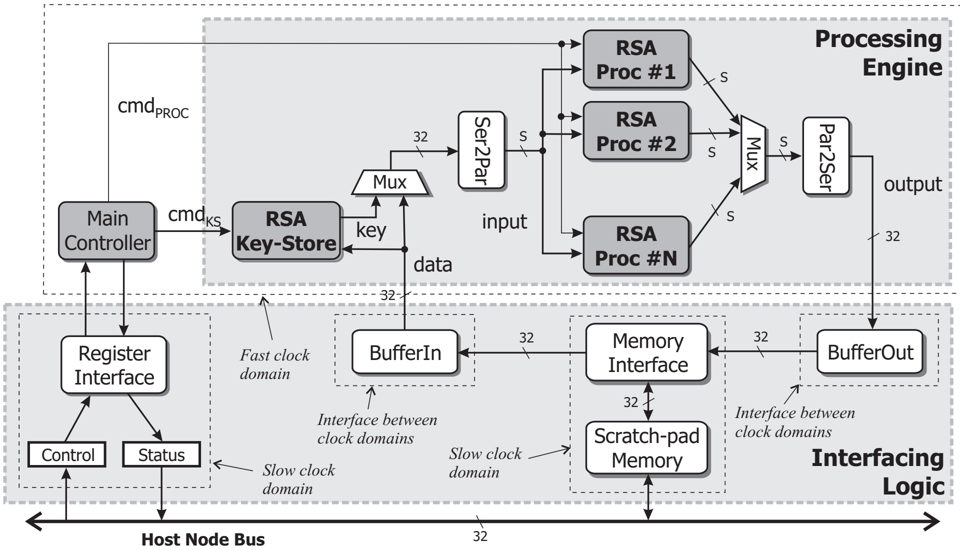



Fig. 3. Architecture of the FPGA-based Crypto-engine.

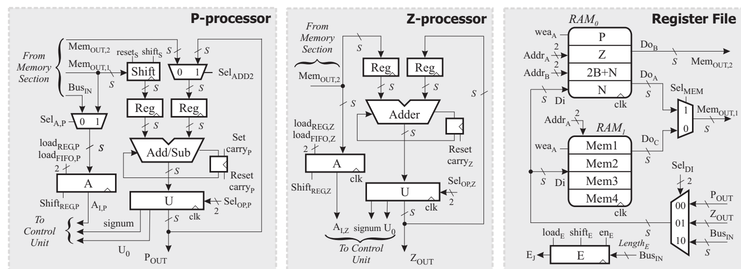



Fig. 4. Architecture of an RSA Processor's Components.

signatures with multiple RSA keys are required (e.g., a single Attribute Authority can use different RSA key pairs for issuing attribute certificates with different policies [11]).

**Main Controller**. An important design goal was to maximize the parallelism offered by the multiple RSA Processors, and hence, to provide linear speed-up with the number of RSA Processors. The Main Controller achieves this goal by accepting exponentiation requests from the host system at any time (possibly in batch), by parsing and dispatching the requests to the appropriate RSA Processors, and by promptly notifying the host system of an RSA Processor's execution completion so that the host system can issue a new computation to that RSA Processor. In addition, the Main Controller incorporates most of the communication and synchronization logic needed by the crypto-engine: it orchestrates the interactions among RSA Processors, KeyStore and the host system, enables reliable communication between entities in different clock domains, and arbitrates accesses to shared resources (e.g., data buses, On-Board RAM). The RSA Processors and KeyStore provide more primitive operations, such as RSA exponentiation, outputting results on a bus, and reading inputs from a bus. Although this approach strengthens re-usability of the individual functional units, it complicates the design of the Main Controller, which results in an implementation combining hardware logic with micro-code in a large state machine (over 250 states) described by over 4,700 VHDL lines.

**Glue Logic**. Additional logic is necessary for the operation of the crypto-engine, as shown in Figure 3:

- The *Ser2Par*/*Par2Ser* blocks convert 32-bit words (used by the KeyStore and the Memory Interface) to/from $S$-bit words (used by the RSA Processors)—$S$ is the internal parallelism of an RSA Processor and is optimally determined in § VI.
- The *Register Interface* is used to exchange data with the Control and Status ports, which are on the FPGA board and constitute the interface between the crypto-engine and the host system. The *Memory Interface* is used to access the On-Board RAM (described below). Note that the frequencies of Register Interface and Memory Interface are limited by the maximum sustainable frequency of the PCI bus and the On-Board RAM, respectively. Therefore, these two components use distinct clocks that operate at a lower frequency than the clock used by other parts of the crypto-engine (e.g., RSA Processors).
- The *On-Board RAM* (placed on the FPGA board) is organized into three logic blocks, mapped on different physical RAMs, to permit concurrent accesses to the different blocks, as for an interleaved memory. The first block contains the table of the RSA operation requests issued by the host system, where each request indicates both the RSA Processor and the RSA key to use. The second block is used as a temporary buffer for the host system to load an RSA key into the KeyStore. The third block contains the input data to be processed by the RSA Processors and the corresponding results.

## B. Crypto-engine Operation

The host system and the several entities making up the crypto-engine architecture interact in a complex fashion to implement a crypto-engine operation (i.e., RSA exponentiation


Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 12,2026 at 01:17:31 UTC from IEEE Xplore. Restrictions apply.
IEEE COMPUTER SOCIETY

or load of a secret key into the KeyStore). The interaction starts with the host system submitting a batch of requests to the crypto-engine using (1) Control Port and Status Port to synchronize and (2) On-Board RAM to exchange data (e.g., request details, the key to be loaded, and the data to be processed by the RSA processors). Specifically, the Main Controller expects the host system to write a command byte in the Control Port, after which the host system waits for the Main Controller to write an acknowledgment byte in the Status Port. The supported command bytes and their use are as follows:

- A `READ_KEY` command indicates that a new secret key is to be downloaded in the KeyStore. Before issuing this command, the host system copies into a predetermined position of the On-Board RAM the key length, the key data, and the index of the KeyStore entry in which the key is to be placed. Once the Control Port is written, the Main Controller pilots the On-board RAM—through the Memory Interface—forcing it to output the corresponding sequence of 32-bit words on the *data* bus. These words are at the same time sampled by the KeyStore. On completion, the Main Controller erases the contents of the On-Board RAM to avoid redundant copies of the key and writes a void value in the Status Port. This operation wakes up the host systems through an interrupt; consequently, the host can submit further requests to the crypto-engine.
- A `READ_REQ_TABLE` command indicates that a batch of new RSA requests is ready. Before issuing this command, the host system copies into the first block of the On-Board RAM a request table having an entry for each RSA Processor. A generic entry indicates its validity (i.e., there is a new operation for the associated RSA Processor), which operation to perform (encryption or decryption), and which key (in the KeyStore) to use. The input data is separately copied into the third block of the On-Board RAM.
  When the Control Port is written, the Main Controller dispatches the valid requests to the appropriate RSA Processors. Each request dispatch occurs in three macro phases: (1) the key is passed from the KeyStore to the RSA Processor; (2) the input data is passed from the On-board RAM, through the Memory Interface, to the RSA Processor; and (3) the RSA processor is activated. Phases (1) and (2) consist of a number of micro phases, which are complex due to communication across clock domains and different parallelism of the communicating entities. Once all requests have been dispatched, the Main Controller moves to polling mode, which is discussed next.
- A `POLLING` command indicates that no new RSA request is available; consequently, the Main Controller moves to polling mode. When in this modality, the Main Controller constantly checks an array of *RSAProc_done* signals, each of which is associated with an RSA Processor and set to 1 by a processor that completes. On detecting an RSA Processor that completed, the Main Controller instructs (1) that processor to output its internal result on the *output* bus and (2) the On-Board RAM to sample the data (through the Memory Interface). Then, the Main Controller resets the RSA Processor and writes the processor's id in the Status Port. This last operation wakes up the host system, which can read the result from the On-Board RAM and submit further requests to the crypto-engine.

To increase our confidence in the resulting crypto-engine design over the extensive testing campaign performed, we have formally modeled (see Fig. 5) and mechanically proved the algorithm executed by the Main Controller using the Spin model checker [12]. We have proved both safety properties, expressed in the form of `assert` statements embedded in the code, and a liveness property. The liveness property formalizes the requirement that if the software running on top of the crypto-engine issues a request for a given RSA Processor, then eventually the processor completes and the software is notified. As an example verification result, consider the polling algorithm described above. An important detail of the algorithm is that, after RSA Processor $i$ completes, the Main Controller restarts processor polling from the $(i+1)$-th $RSAProc\_done$ signal. Formal verification with the Spin model checker indicates that with this policy we can avoid execution runs in which an RSA Processor completes but the Main Controller never releases it due to continuous requests for the other RSA Processors. In contrast, a different polling policy could cause deadlock of one or more RSA Processors.

### C. Crypto-engine Implementation

The proposed crypto-engine architecture is implemented with a Celoxica RC1000 board. The board is a standard 32-bit PCI card that contains a Xilinx FPGA device, four 2MB SRAMs accessible by both the host system (through DMA transfers) and the FPGA, a Control port, and a Status port. The Xilinx Virtex series of FPGAs consists of a configurable logic cell (CLB) and interconnection circuitry, tiled to form a chip. A CLB consists of two slices, each of which contains two 4-input look-up tables, two flip-flops, and carry chain logic. The particular FPGA used is a Xilinx VirtexE2000-8, which has 19200 slices (i.e., 9600 CLBs) and incorporates 160 fully synchronous, dual-ported, 4096-bit block memories (BRAM). As tools, Aldec Active VHDL 4.2 is used for the design, and Synplicity Synplify Pro 7.1 for synthesis, the latter being integrated in the Xilinx ISE 5.1i design flow.

## IV. Design and Implementation of the Software Architecture and Replication Scheme

The software architecture of a Certificate Engine replica is sketched in Figure 6. Its top layer implements the Certificate Engine functionalities and uses few core functions of OpenSSL [20] for executing non-computationally-critical cryptographic operations in software (e.g., SHA-1 hashing). Expensive RSA exponentiations are executed through the attached crypto-engine.


Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 12,2026 at 01:17:31 UTC from IEEE Xplore. Restrictions apply.

```
inline polling() {

  printf("polling\n");

  do
    :: RSAProc_done_req[j] == 1 -> break;
    :: else -> j = (j + 1) % N_PROC;
  od;

  assert(RSAProc_busy[j] == 1);
  assert(running > 0);
  printf("polling - proc %d finished\n", j);
  assert(RSAProc_done_ack[j] == 0);
  RSAProc_done_req[j] == 1;
  RSAProc_done_ack[j] = 1;
  RSAProc_done_req[j] == 0;
  RSAProc_done_ack[j] = 0;
  RSAProc_busy[j] = 0;
  running--;

  /* write result */

  printf("getting result data\n");
  k = 0;
  do
    :: k < DATA_SZ ->
        assert(bus_req == 0);
        bus_req = 1;
        (bus_ack == 1);
        printf("receiving %d\n", bus);
        mem[j].bytes[k] = bus;
        bus_req = 0;
        (bus_ack == 0);
        k++;
    :: else -> break;
  od;
  k = 0;
  sp!j;
  /* increment j */
  j = (j+1) % N_PROC;
}


active proctype MC() {

  printf("MC is up\n");

  bit RSAProc_busy[N_PROC];
  byte running = 0;

  byte i;
  byte cmd;
  byte j;
  byte k;

  /* main cycle */

  do
    :: cp?cmd ->
        if
          :: cmd == READ_REQ_TABLE ->
              printf("MC - cmd = process frame\n");
              dispatch_requests();
              polling();
          :: cmd == MC_READ_KEY ->
              printf("MC - cmd = load key\n");
              k = 0;
              do
                :: k < KEY_SZ ->
                    keyStore[keyId].bytes[k] = keyMem.bytes[k];
                    k++;
                :: else -> break;
              od;
              k = 0;
              sp!1;
          :: cmd == MC_POLLING ->
              printf("MC - cmd = polling\n");
              polling();
        fi;
  od;

}
```

Fig. 5. Excerpt form Spin Formalization of the Main Controller's Algorithm.

The *Crypto-engine Library* offers primitives such as sending RSA exponentiation requests to the crypto-engine and retrieving the corresponding results, and loading RSA keys into the KeyStore. The Crypto-engine Library supports multithreading and thus enables using multiple RSA Processors at the same time. Used by Crypto-engine Library, the *Crypto-engine Driver* is a Linux device driver for accessing the FPGA board of the crypto-engine (see Figure 3). I/O primitives are offered by this driver for writing an 8-bit Control port, reading an 8-bit Status port, and performing a DMA-based data transfer between the central memory of the host system and the On-Board RAM.

On the replication side, the *Virtual Socket Layer* provides transparent replication to the Certificate Engine code, while the *Preemptive Deterministic Scheduler* guarantees replica determinism [13]. At the bottom, a *Group Communication System* is used for reliable multicast/unicast communication. Due to space limitations, the reader is referred to [21] for further information.

## V. Security Analysis

In contrast with a common misconception, combining hardware and software approaches does not necessarily improve a system's security as one would expect. In designing our crypto-engine architecture, we have carefully studied the interaction between the software and hardware domains and have employed a range of techniques that can minimize this potential security bottleneck. These issues are discussed in detail through an overall analysis of the security benefits of the crypto-engine approach over software-only intrusion-tolerant architectures (e.g., [2]).

The analysis considers attackers whose ultimate goal is to forge Attribute Authority signatures. To accomplish this goal, attackers need to obtain the Attribute Authority's private key and/or to take control of the Certificate Engine system, without being detected. (Denial-of-service attacks are not covered in this study.) The proposed architecture uses threshold cryptography to guarantee that the private key cannot be reconstructed if fewer than half of the private key shares are disclosed, i.e., if fewer than half of the Certificate Engine replicas are compromised (e.g., due to personnel bribing).[3] The remainder of this section focuses on a malicious attack on a single replica and argues that the crypto-engine approach makes such a task substantially more difficult than on a system implemented entirely in software.

Having selected a target replica node, an attacker can succeed by either (1) hardware-level intrusion, if he/she has physical access to the replica node, or (2) software-level intrusion, if he/she has only remote access to the node.

**Hardware-Level Intrusion**. Our analysis is based on the attack categories identified in [22]: physical attack, read-back attack, and side-channel attack. A *physical attack* aims at uncovering the FPGA design by opening up the FPGA package

[3] To limit the success scope of malicious attacks, actual deployments can use diversified replicas, i.e., replicas that have different implementations and run on diverse operating systems and hardware platforms.


Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 12,2026 at 01:17:31 UTC from IEEE Xplore. Restrictions apply.

and probing (undocumented) points inside the chip without damaging the device. Due to increasing FPGA complexity, this attack can be achieved only with advanced inspection methods (e.g., Focused Ion Beam), which are quite costly and are probably possible only for large organizations (e.g., intelligence services).

A *read-back attack* accesses/reads the FPGA configuration file from the FPGA chip (using the read-back functionality generally available on the FPGA device for debugging purposes), after which the attacker reverse-engineers the obtained bit-stream. To prevent the read-back attack, most manufacturers provide the option of disabling the read-back functionality. Moreover, even though theoretically possible to interpret and/or to modify the bit-stream of an FPGA, major vendors (e.g., Xilinx, Actel) maintain that it is virtually impossible. The irregular row and column pattern of the hierarchical interconnection network exacerbates the inherent complexity of the reverse-engineering process [23].

A *side-channel attack* exploits unintentional information-leakage sources (e.g., power consumption, timing behavior, electromagnetic radiations) in the implementation. At present, little work has investigated the feasibility of such attacks against FPGAs. Nevertheless, attacks using power consumption and specific to RSA are known in the literature. For instance, Simple Power Analysis and Differential Power Analysis exploit the fact that a straightforward implementation of the Right-to-Left Binary Algorithm (widely used in RSA hardware circuits, including our RSA Processor) has power consumption that changes in time with the bit-sequence of the RSA key (thus, monitoring the FPGA power consumption allows discovering the RSA key). Simple countermeasures can be found in [24]. In our case, power attacks are more difficult to launch, since multiple RSA Processors operate concurrently and asynchronously, effectively masking the information that can be revealed by the overall FPGA power consumption.

**Software-Level Intrusion**. An important property of our crypto-engine design is that a secret key share $S_i$ (once loaded in the KeyStore) cannot leave the crypto-engine's FPGA device and cannot be read by software (there is no hardware support for such an operation). As a consequence, the attacker cannot obtain the value of $S_i$ solely by software means. At most, he/she can take control of the crypto-engine and use $S_i$ indirectly, through malicious RSA operation requests. Importantly, malicious usage of a compromised crypto-engine requires continuous attacker activity on the target machine, which can be detected by an Intrusion Detection System. In contrast, in software-only Certificate Engine implementations, a remote attacker can obtain the value of a private key share (e.g., by reading it from the file system) and transfer it to a remote system he/she possesses for unchecked usage. (In [5], the authors show how an attacker with marginal computing skills and resources can locate and steal secret keys on the disk within few minutes.)

To prevent unauthorized crypto-engine usage, our Crypto-engine Driver enforces access control [25]. An authorized thread/process is uniquely identified by a *Task Authentication Info (TAI)*, which is a pair consisting of the UNIX PID of the thread/process and the value of the `jiffies` kernel cycle counter, sampled at the thread/process's creation time. (`jiffies` values are included in TAIs to prevent loop attacks.[4]) At a replica node's startup time (and only at that time), the Crypto-engine Driver is loaded with a table of TAIs. The driver stores the table in nonswappable kernel memory to ensure that TAIs are not available while the microprocessor executes in user mode (e.g., to a root shell launching a carefully crafted disk access to modify the TAI table when it resides on the disk) but only in kernel mode.

In this setup, a remote attacker can defeat the Crypto-engine Driver's authentication mechanism by rebooting a replica's node after tampering with the TAI loading procedure, the kernel image file, or the Certificate Engine executable file (case 1). Without rebooting the replica node, the attacker can tamper with the memory image of an authorized application, i.e., the Certificate Engine (case 2), load a malicious kernel module (case 3), or tamper with the memory image of the Crypto-engine Driver (case 4). Case 1 can be handled by reboots supervised by the administrator, where those critical pieces of code are loaded from some read-only media or other trusted source in order to eliminate any installed Trojan horse. Case 2 can occur due to a vulnerability in the Certificate Engine code (e.g., by means of a buffer overflow exploit) or a run-time manipulation of its memory image. A number of countermeasures are available in the literature [26], [27]. Case 3 is easily prevented by statically compiling the Crypto-engine Driver within the kernel image and by disabling dynamic kernel module loading. Case 4 requires exploiting a kernel vulnerability in order for a remote shell to execute malicious code in kernel mode. Discoveries of kernel vulnerabilities are definitely less frequent than discoveries of user application vulnerabilities [28]; thus, it is reasonable to rely on the administrator to patch an insecure kernel promptly.

## VI. Performance Evaluation

This section first discusses optimal dimensioning the crypto-engine architecture of § III for an actual embodiment, and then provides a performance evaluation of a resulting implementation.

**Dimensioning the Crypto-engine**. Having fixed the total area $A_{Tot}$ available for the RSA Processors, the crypto-engine's overall throughput can be maximized by balancing the throughput of a single RSA Processor, through the processor parallelism $S$,[5] and the number $N$ of RSA Processors that can execute concurrently. The crypto-engine's maximum throughput can be approximated by the ratio $N/T_S$, where $T_S$ is the maximum computation time of an RSA Processor

[4] A loop attack consists of spawning a new process until one is created with a desired PID. Usually, UNIX implementations are such that the PID of a newly created process is equal to the PID of the previously created process incremented by one. This results in PIDs forming a periodic sequence whose period is determined by the maximum value used by an internal kernel counter (e.g., 32767 on Linux).

[5] $S$ is the bit length of the words on which an RSA Processor operates (see § III).


Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 12,2026 at 01:17:31 UTC from IEEE Xplore. Restrictions apply.

| S | $A_{Tot}$ [slices] | $A_S$ [slices] | N | $T_{Clk}$ [ns] | $T_S$ [ms] | $A_S \cdot T_S$ | Throughput [ACs / sec] |
|---|---|---|---|---|---|---|---|
| 256 | 18535 | 2906 | 6 | 32.2 | 0.170 | 494 | 35.29 |
| 128 | 18687 | 1876 | 9 | 19.8 | 0.188 | 353 | 47.83 |
| **64** | **18792** | **1194** | **15** | **13.0** | **0.233** | **279** | **64. 28** |
| 32 | 18800 | 998 | 18 | 9.8 | 0.341 | 341 | 52.72 |

TABLE I

CRYPTO-ENGINE PARAMETERS FOR DIFFERENT IMPLEMENTATIONS.

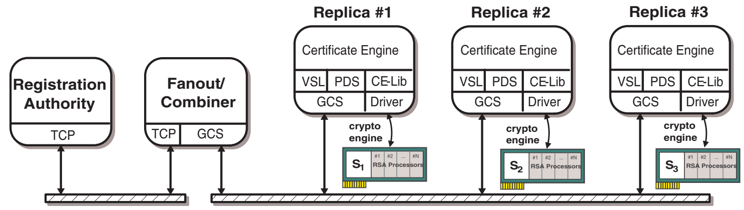



Fig. 6. The Experimental Setup for the Deployed Certificate Engine.

(and is a function of $S$). Also, let $A_S$ be the area occupied by a single RSA Processor. By equating the total required area $N \cdot A_S$ to the total available area $A_{Tot}$, it is easy to derive that maximizing the ratio $N/T_S$ corresponds to minimizing the product $A_S \cdot T_S$. While $A_S$ is computed by the synthesizer, $T_S$ and $A_{Tot}$ are obtained as follows:

- Time $T_S$ is given by the product of the number $N_{RSA}$ of clock ticks required to perform an RSA exponentiation and the clock period of the integrated system ($T_{Clk}$), which depends on $S$ and is computed after the place-and-route phase of the synthesis. For our architecture, it can be shown that $N_{RSA} = \Big((2M+2)+(B+3)\cdot M\Big)\cdot \Big(H+2\Big)+2M+1$, where $B$ is the bit length of the RSA key modulus, $M$ is equal to $\lceil (B+3)/S \rceil$, and $H$ is the maximum bit length of the RSA key exponents.
- Area $A_{Tot}$ is computed from the total FPGA area (19200 slices and 160 BRAMs) by subtracting the area for the KeyStore (22 slices and 6 BRAMs for accommodating four distinct RSA keys), the area for the Main Controller (340 slices and 2 BRAMs), and the area for remaining glue logic (ranging from 303 slices for $S = 256$ to 38 slices for $S = 32$).

For the implementation setup described in § III-C, Table I reports the resulting minimum clock periods of the overall crypto-engine ($T_{Clk}$), the product $A_S \cdot T_S$, and the crypto-engine maximum throughput, for different values of $S$ and for $B = H = 1024$. The table indicates that the optimal value of $S$ is 64, which corresponds to 15 RSA Processors integrated into a single crypto-engine.

**Experimental Evaluation**. We now present a performance evaluation of our Certificate Engine prototype in different hardware/software configurations and under the workload generated by a synthetic benchmark. The studied implementation is based on a 15 RSA Processor crypto-engine (dimensioned as discussed above). The experimental setup is reported in Figure 6 and is a simplified[6] deployment of the system in Figure 2. Specifically, the setup consists of two Ethernet 100 Mbps LANs: one connecting the clients (modeling the Registration Authorities) to a fanout/combiner process, the other connecting the fanout/combiner to three replicas. Replicas and fanout/combiner execute on Pentium III 500 MHz-based machines running Linux 2.4. Ensemble 1.40 [29] is used as group communication system,[7] and replication is provided by the Virtual Socket Layer [13].

A synthetic benchmark application models requests generated from a set of 50 Registration Authorities. This is achieved through a single client process composed of 50 threads, each of which continuously generates a random certificate request and waits for the corresponding attribute certificate to arrive. On a Certificate Engine replica, a pool of 15 worker threads serves requests coming concurrently from the client threads. (Our experiments indicate that this setup is sufficient to study the maximum server throughput.)

Three main configurations are considered: (1) *baseline*, consisting of a simplex Certificate Engine without replication instrumentation and threshold cryptography support; (2) *triplicated Certificate Engine with no threshold cryptography*, where each replica possesses the Attribute Authority private key $D$ and can generate a complete Attribute Authority signature (note that this is not the system described in § II); and (3) *triplicated Certificate Engine with threshold cryptography*, where each replica $i$ only possesses a share $S_i$ of $D$ (as indicated in § II). Each configuration comes in two flavors: a strictly software-based implementation (SW) and a crypto-engine-based implementation (HW). In order to ensure deterministic replica behavior, triplicated configurations can use either the Preemptive Deterministic Scheduler (PDS) discussed in [13] or a Non-Preemptive Deterministic Scheduler (NPDS), which is an algorithm alternative to PDS and is based on [31]. In hardware-based configurations, Certificate Engine replicas use the crypto-engine only for RSA signature of certificate digests; our experiments indicate that this operation is by far the most computationally intensive of those involved in a certificate signature (e.g., certificate assembling, SHA-1 hashing). In the threshold cryptography configurations, the fanout/combiner implements, entirely in software, the cryptography operations needed to combine replicas' partial signatures; our experimental results show that this does not constitute a bottleneck for the overall system's performance.

For a given hardware/software configuration ($C$), *throughput* is defined as the number of attribute certificates issued per second. To measure performance impact due to the use of replication and threshold cryptography in $C$, configuration $C$'s

[6]The experimental setup in Figure 6 differs from the system in Figure 2 due to the absence of the backup fanout/combiner and the absence of the Repository (whose role is mimed by the client threads). We expect that in the considered scenario these differences have a marginal impact on the system's performance.

[7]Note that Ensemble is not resilient to Byzantine failures, and hence, actual Certificate Engine embodiments should employ protocols such as [30]. Nevertheless, this limitation does not affect the qualitative results of our performance study, whose primary objective is to show a substantial performance improvement due to the hardware crypto-engine.


Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 12,2026 at 01:17:31 UTC from IEEE Xplore. Restrictions apply.

*overhead* is defined as the ratio between the throughput of the hardware baseline (i.e., of the simplex Certificate Engine with crypto-engine) and the throughput of the hardware version of $C$ (i.e., of configuration $C$ while using the crypto-engine), after having subtracted 100%. To measure performance improvement due to the crypto-engine, $C$'s *speedup* is defined as the ratio between the throughput of the hardware version of $C$ and the throughput of the software baseline (i.e., of the simplex Certificate Engine without crypto-engine).

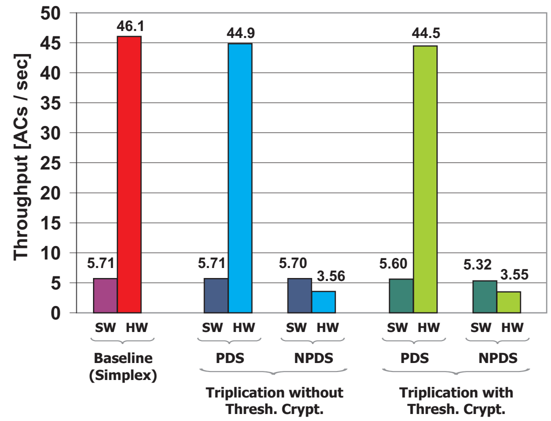
Figure 7 and the associated table show the measured throughput, overhead, and speedup for the different configurations introduced above. We now summarize the major findings from the experiments:

- Hardware acceleration provides remarkable speedup for all configurations (approximately 8-fold).
- PDS is able to fully exploit the functional parallelism offered by the hardware crypto-engine, resulting in replicated systems with low overhead (3-4%). In contrast, NPDS cannot use more than one RSA Processor at a time, and the resulting overhead is very large (1200%).
- Adoption of threshold cryptography has a low performance impact with respect to the triplicated configuration with no threshold cryptography (0.9% = $(44.9/44.5-1)\times 100$). This result indicates that implementing the voter/combiner entirely in software has negligible performance impact.

One could argue that the measured 8-fold speedup is because the experiments use relatively slow microprocessors (500 MHz Pentium III). We claim that the proposed hardware-based Certificate Engine can also deliver significantly improved performance with respect to software-only implementations of a Certificate Engine executing on a currently available, fast microprocessor. For example, the measured throughput of a software baseline configuration executing on an AMD Athlon 2600 XP+ at 2GHz is approximately 12.1 certificates per second. Consequently, the expected speedup of a triplicated system employing our crypto-engine is about 4-fold (44.5/12.1). This is a rather conservative estimate for two major reasons: (1) the throughput of 44.5 used in the calculation is obtained for 500 MHz Pentium III machines (and not for Athlon machines, as for the throughput of 12.1), and (2) the FPGA device used in this study can be considered as a medium-performance fabric (we expect a crypto-engine implemented on a high-end FPGA Xilinx Virtex2 Pro 125 to provide 10 times better throughput with respect to the studied implementation). Finally, observe that employing the latest generation of microprocessors leads to moderately improved throughput, indicating that crypto-engine approaches are attractive solutions in accelerating complex RSA computations.

## VII. Related Work

A significant volume of work exists on: (1) fault- and intrusion-tolerant protocols, and schemes for replication and distribution of trust [3]; (2) digital certificates and security infrastructures [32]; and (3) practical experiences in the application of attribute certificates [33], [34]. In this section, we limit our attention to projects whose specific objective was to develop certification systems for supporting access control mechanisms.




| Certificate Engine Configuration | Throughput [ACs/sec] SW | HW | Overhead | Speedup |
|---|---|---|---|---|
| Baseline | 5.71 | 46.1 | N/A | 8.1 |
| Trip. No Thresh. Crypt. (PDS) | 5.71 | 44.9 | 2.7% | 7.9 |
| Trip. No Thresh. Crypt. (NPDS) | 5.70 | 3.56 | 1200% | 0.62 |
| Trip. Thresh. Crypt. (PDS) | 5.60 | 44.5 | 3.6% | 7.8 |
| Trip. Thresh. Crypt. (NPDS) | 5.32 | 3.55 | 1200% | 0.62 |

Fig. 7. Experimental Results for Several Configurations of the Certificate Engine.

From a high level, the COCA system [2] provides functionalities similar to our system; however, the resulting architecture and the contributions made are quite different. COCA is more of a shared-variable service than a Certification System (i.e., a Certification Authority or an Attribute Authority). Two functions, namely `Update` and `Query`, are used to write to and read shared certificate variables, respectively. Application-specific ordering is used for update requests and can cause query operations not to return the most recent certificate, which is incompatible with PKIX recommendations [9]. The system we study is an Attribute Authority that complies with PKIX recommendations and X.509 formats [10]. The major contribution of COCA is in the communication and recovery protocols (in support of threshold cryptography), which can operate under an asynchronous system model. The main contribution of our work is the hardware accelerator, which provides both high performance and tamper resistance capabilities to the individual nodes.

In [35], [36], and [6], the authors propose an authorization scheme based on a fault- and intrusion-tolerant authorization server where smart-cards are used on the client side to implement access control for the hosts participating in an application. In contrast, in our architecture, an FPGA board implements the cryptographic routines and stores the private key shares of the server nodes. Both choices have their advantages and drawbacks: smart cards are removable but have quite poor performance (as such, they are more suitable for the client side); in contrast, FPGA boards are


IEEE COMPUTER SOCIETY
Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 12,2026 at 01:17:31 UTC from IEEE Xplore. Restrictions apply.

(typically) permanently attached to the host node but provide high performance (as such, they are more suitable for the server side).

The Akenti system [37] provides access control by means of three types of certificates: identity certificates, use-condition certificates, and attribute certificates. While identity certificates are issued by external Certificate Authorities, use-condition and attribute certificates are created by Akenti's Certificate Generators. As the main focus of Akenti is the distributed definition and automated handling of access policies, the Certificate Generators were implemented as Java applications that sign (Akenti-specific) text certificates. It should be possible to integrate our Certificate Engine in Akenti's Certificate Generators, thus providing them with a high-performance core that can also cope with faults and intrusions.

## VIII. Conclusions

Through a combined hardware/software approach, we contribute the design, implementation, and evaluation of a distributed, RSA-based Certificate Engine for Attribute Authorities that can tolerate both accidental and malicious faults yet deliver high performance. The key element of the proposed architecture is an FPGA-based, parallel *crypto-engine* that provides *RSA Processors* for efficient execution of computationally intensive RSA signatures (involved in issuing a certificate) and a *KeyStore* facility, which provides tamper-resistant storage for preserving secret keys. A performance evaluation shows that executing computationally intensive RSA operations in hardware (and concurrently on multiple RSA Processors) enables deployment of high-performance security services (e.g., with 8 times the throughput of software implementations). Also, combining *active replication* and *threshold cryptography* allows tolerating both accidental faults and malicious attacks of system components with only a small performance overhead (approximately 3%). Finally, combining hardware and software mechanisms can make malicious attacks substantially more difficult than in pure software implementations.

## Acknowledgments

This work is supported in part by NSF grants CCR 00-86096 ITR and CCR 99-02026. This work is also supported in part by the Italian National Research Council (CNR), by Ministero dell'Istruzione, dell'Universitá e della Ricerca (MIUR), and by Regione Campania, within the framework of following projects: SP1 Sicurezza dei documenti elettronici, Gestione in sicurezza dei flussi documentali associati ad applicazioni di commercio elettronico, Centri Regionali di Competenza ICT, and Telemedicina. We thank Fran Baker for insightful editing of our manuscript.

## References

[1] Y. Deswarte, L. Blain, and J.-C. Fabre, "Intrusion tolerance in distributed computing systems," in *IEEE Symp. on Security and Privacy*, 1991.
[2] L. Zhou et al., "Coca: A secure distributed online certification authority," *ACM Trans. on Computer Systems*, vol. 20, no. 4, 2002.
[3] "MAFTIA Project," http://www.newcastle.research.ec.org/maftia/, 2003.
[4] M. Cukier, et al., "Intrusion tolerance approaches in itua," in *In Supp. of DSN*, 2001.
[5] A. Shamir and N. van Someren, "Playing hide and seek with stored keys," *LNCS*, vol. 1648, 1999.
[6] Y. Deswarte et al., "Intrusion-tolerant authorization scheme for internet applications," in *Proc. of DSN*, 2002.
[7] O. Kommerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," in *Proc USENIX Workshop on Smartcard Technology*, 1999.
[8] R. L. Rivest, et al., "A method for obtaining digital signature and public-key cryptosystems," *Commun. of ACM*, vol. 21, 1978.
[9] A. Arsenault and S. Turner, "Internet X.509 public key infrastructure: Roadmap," IETF, 2002.
[10] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," IETF — RFC 3280, 2002.
[11] S. Farrell and R. Housley, "An internet attribute certificate profile for authorization," IETF — RFC 3281, 2002.
[12] G. Holzmann, "The SPIN model checker," *IEEE Trans. on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.
[13] C. Basile, Z. Kalbarczyk, and R. Iyer, "Preemptive deterministic scheduling algorithm for multithreaded replicas," in *Proc. of DSN*, 2003.
[14] "Motorola mpc185," http://e-www.motorola.com/, 2003.
[15] "Sun crypto accelerator 1000," http://www.sun.com/, 2003.
[16] J.G. Dyer, et al., "Building the IBM 4758 secure coprocessor," *Computer*, vol. 34, no. 10, 2001.
[17] V. Shoup, "Practical threshold signatures," *LNCS*, vol. 1807, pp. 207–218, 2000.
[18] P. L. Montgomery, "Modular multiplication without trial division," *Math. of Computation*, vol. 44, no. 170, pp. 519–521, 1985.
[19] A. Mazzeo, et al., "An FPGA-based implementation of the RSA algorithm," in *Proc. of DATE*, 2003.
[20] "OpenSSL Project," http://www.openssl.org/, 2003.
[21] G. P. Saggese, et al., "An intrusion- and fault-tolerant attribute authority using programmable hardware and software replication," UIUC, Tech. Rep., 2004.
[22] T. Wollinger, J. Guajardo, and C. Paar, "Cryptography on FPGAs: State of the art implementations and attacks," *ACM Trans. on Embedded Computing Systems*, 2003.
[23] Xilinx, "Configuration issues: Power-up, volatility, security, battery back-up," Application Note XAPP 092, 1997.
[24] M. Joye, "Recovering lost efficiency of exponentiation algorithms on smart cards," *Electronics Letters*, vol. 38, no. 19, pp. 1095–1097, 2002.
[25] A. Mazzeo, et al., "An FPGA-based key-store for improving the dependability of security services," University of Naples Federico II, Italy, Tech. Rep., 2003, http://cds.unina.it/~lrom/download/papers.
[26] A. Baratloo, N. Singh, and T. Tsai, "Transparent run-time defense against stack smashing attacks," in *Proc. of USENIX Annual Technical Conference*, 2000.
[27] C. S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation - tools for software protection," *IEEE Trans. on Software Engineering*, vol. 28, no. 8, 2002.
[28] "CERT Advisories," http://www.cert.org/advisories, 2003.
[29] M. Hayden, "The Ensemble system," Ph.D. dissertation, Dept. of Computer Science, Cornell University, USA, 1997.
[30] C. Cachin and J. Poritz, "Secure intrusion-tolerant replication on the internet," in *In Proc. of DSN*, 2002.
[31] R. Jimenez-Peris, et al., "Deterministic scheduling for transactional multithreaded replicas," in *Proc. of SRDS*, 2000.
[32] "SDSI-SPKI Project," http://www.syntelos.com/spki/, 2003.
[33] "I-Care Project," http://www.cert-i-care.org/, 2003.
[34] "HARP Project," http://www.telecom.ntua.gr/ HARP/HARP/HARP.htm, 2003.
[35] V. Nicomette and Y. Deswarte, "An authorization scheme for distributed object systems," in *Proc. of Int'l Symp. on Security and Privacy*, 1997, pp. 21–30.
[36] Y. Deswarte, et al., "An internet authorization scheme using smartcard-based security kernels," in *Proc. of Int'l Conf. on Research in Smart Cards (e-Smart)*, 2001, pp. 71–82.
[37] Mary Thompson, et al., "Certificate-based access control for widely distributed resources," in *Proc. of USENIX Security Conference*, 1999, pp. 215–228.


Authorized licensed use limited to: University of Maryland College Park. Downloaded on January 12,2026 at 01:17:31 UTC from IEEE Xplore. Restrictions apply.