Hardware/Software Security Support

R. R. Brooks, Associate Professor
S. T. Sander, Assistant professor
Holcombe Department of Electrical and Computer Engineering
Clemson University
Clemson, SC 29634-0915
Tel. 864-656-0920
Fax. 864-656-1347
email: rrb@acm.org

1. Introduction

It is necessary to combine cryptographic primitives, compiler optimizations and adaptive hardware to create a truly secure mobile computing environment. Here we consider embedded applications, but this approach can also create secure co-processors for desktop systems. FPGA hardware can use standard symmetric and public key cryptography approaches.

All chip input and output can be encrypted putting a strain on chip performance. Caches on the FPGA alleviate this strain. Data and program security in memory, peripherals, and communications will rely on the complexity guarantees of cryptography algorithms.

Security of the underlying hardware is based on two premises: (*i*) Fielded devices will be created and/or initialized in a secure facility. (*ii*) Devices will have physical safeguards against tampering and directly reading the internal state of the processor.

Covert channels, including monitoring resource consumption, are an important vulnerability. To counter attacks using covert channels use a secure instruction set approach, which masks resource usage when processing sensitive information. Extensions can include physically and temporally isolating processing on the chip minimizing information leakage. Physically isolated processes share no common memory.

An interpreter embedded on the chip can execute programs. The interpreter associates keys with users. Access rights and security policies are enforced using in-line reference monitoring. All programs are monitored during execution and terminated before an active policy can be violated.

System integrity can use the approach in [1] to maintain a trust hierarchy. The top layer of the hierarchy is an integrity monitor that uses secure hashing to guarantee that the level beneath it has not been corrupted. The level beneath it includes the interpreter, encryption primitives, and key management functions.

Hardware integrity is assumed. No mechanism will exist on the FPGA for modifying the integrity monitor. The integrity monitor verifies that the level beneath it has not been corrupted. Lower levels in the hierarchy guarantee that changes are made only by authorized entities, and that no corruption occurs after the fact. Trust in the rest of the system rests on the sanctity of the higher levels. Violations of trust result in the system restoring the initial state of the violator, terminating the process, or signaling the violation.

2. Detailed approach

This creates a secure computing environment by using hardware and software co-design. Consider a processor with an embedded interpreter executing arbitrary programs, using a set of cryptographic primitives. We consider the system secure, when the following are true:

- Basic infrastructure is not corrupted.
- The system's security policy is maintained.
- Confidential information is kept secret.
- Covert channels are removed.
- Data storage and communications are either encrypted or physically shielded.

This paper sketches a research agenda for creating this type of secure environment using adaptive hardware.

Table 1 provides initial implementation results for symmetric key applications on a small FPGA. Public key implementation in hardware is possible, but most public key algorithms rely on modular exponentiation. Modular exponentiation implementation in hardware is problematic. An alternative, consistent with our vision, is to use an FPGA containing standard processor sub-units. The public key algorithms will execute in the standard processor. As we will explain, access to this standard processor can be controlled.

Table 1:

Virtex-II Pro XC2VP50	Frequency	Area (Slices)	Throughput	Percentage
Block Ciphers				
AES (128/128)	123.793 MHz	5539	1.585 Gbps	23.45443767
AES Encryptor	142.349 MHz	2661	1.822 Gbps	11.26778455
AES Decryptor	124.332 MHz	3364	1.591 Gbps	14.24457995
3DES	214.362 MHz	1070	807.009 Mbps	4.530826558
DES	219.394 MHz	441	825.954 Mbps	1.867378049
Secure Hash Functions				
SHA	97.914 MHz	1722	626.650 Mbps	7.291666667
MD5	58.261 MHz	1838	466.088 Mbps	7.782859079

In-line reference monitoring (IRM) will enforce security policies. Policies control issues like key management. The set of policies enforceable using IRM are strictly defined. Violations are detected before they occur, and prevented. The violating process can be stopped or restored to a safe state. The monitor is part of the interpreter embedded in the chip.

The processor will be hardened and all chip I/O encrypted. This lets us assume that the hardware retains integrity and the system's internal state is secret. All chip I/O is routed through the encryption primitives. Code signing authenticates the sources of information and identifies the keys used for decryption. A new cache structure is needed to minimize the impact on processor performance.

It should be possible to verify FPGA circuit consistency with security policies. Each hierarchy layer (Fig. 1) has:

- No access to configuration registers of higher layers
- Read-only access to its own configuration
- Full access to configuration registers of its subregions.

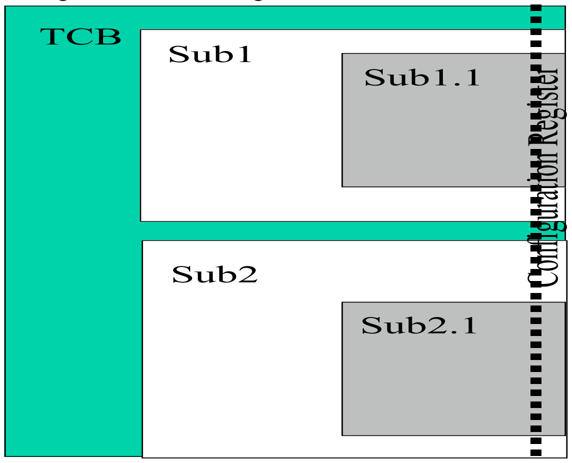

Fig. 1 Each hierarchy level can only access FPGA regions (incl. configuration registers) under its control. Only the integrity monitor has global read access.

The top layer of the trust hierarchy is an integrity monitor. These rules mean that no portion of the FPGA has write access to the integrity monitor. This monitor can verify its own integrity and signal an alarm should it be corrupted. The integrity monitor uses secure hashing to guarantee the integrity of level one components, including: interpreter, key management, and user entity roots. The equivalent of a secure bootstrap is provided by the security guarantees of the hardened processor and lack of support for modification of the integrity monitor. The circuit in Fig. 2 illustrates hardware enforcement of a policy restricting modification of an FPGA region. The hardware and integrity monitor together form the trusted computing base. Both are simple enough to allow exhaustive formal verification.

If the integrity monitor finds corruption at a lower layer, it can either stop the corrupted process or restore it to a trusted initial state. Corruption is thus detected, contained and controlled. It can only flow down the hierarchy. This provides security guarantees for the system in spite of the fact that many desirable security issues, like the presence of viruses or implementation correctness, are often undecidable.

Note in the following discussion the goal is not to create new key management schemes, but to create adaptive hardware support for key management. Keys are stored in protected on-chip memory regions. Users usually access the clear text information they are authorized to use, with no direct access to their keys.

Keys can be stored indirectly using hardware signatures, generated using physical unclonable functions (PUF). A PUF is a circuit outputting a random bit sequence constant on a given chip, but differing from chip to chip. At key setup time, an XOR of the key value and the FPGA digital signature is computed and stored. The key can be recovered using the stored XOR value and device-specific hardware signature. This ensures security because 1) the PUF cannot be determined without destroying it and 2) the XOR value is useless outside the given device.

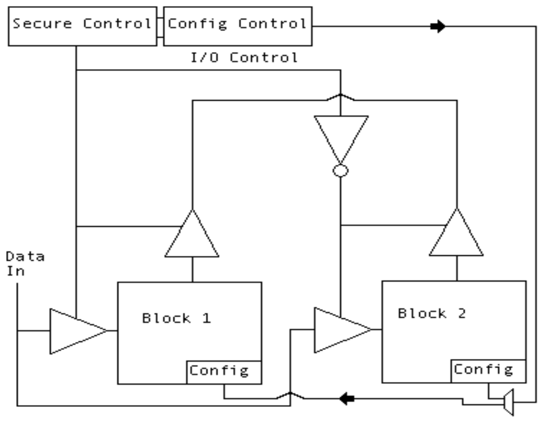

Fig. 2 Configuration memory can only be accessed by the secure control. Uni-directional bus and control multiplexer prevent communication or eavesdropping by other modules. Configuration memory readback is prevented.

Key generation can be external to the system or internal using pseudo-random number generators (like hash functions or block encryption ciphers). Only one key generation server is allowed.

Symmetric key management can use a binary tree structure. Key-encryption keys (KEK) are used to transmit session keys (SK). Each node in a tree of n participants is a leaf with an associated KEK. The root node of the tree provides a global KEK. Intermediate nodes have KEKs accessible to all leaves descended from the node. An SK can then be securely distributed to any group of participants without performing an excessive number of encryptions. In addition, each participant stores at most log n keys. This reduces the work needed to revoke a user's key.

For public key approaches, key management will use the X.509 standards. Public key approaches are difficult to implement in hardware. FGPA's with embedded processors are suited to PKI technology. We will also test hybrid key management schemes.

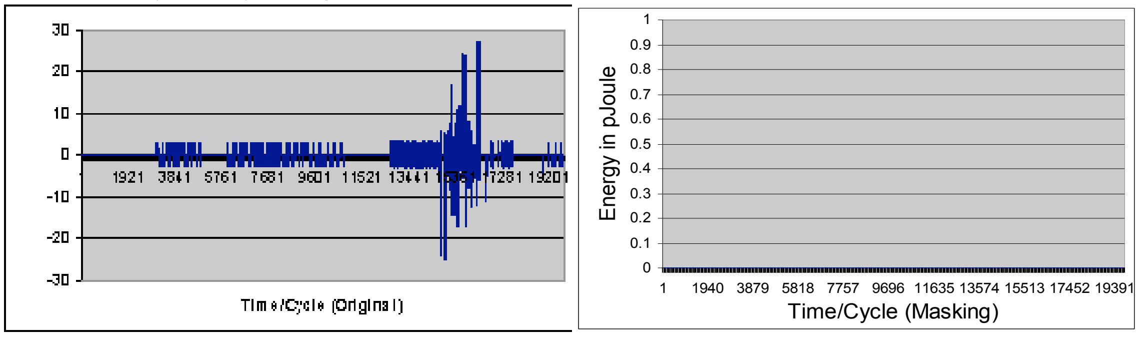

Fig. 3 DES encryption energy consumption difference with keys differing by one bit (top), using secure instructions (bottom) masks this energy consumption difference making it impossible to infer key contents.

To use a secure instruction set, variables and procedures are tagged with a security level. A separate set of hardware instructions are used for handling sensitive information. This approach successfully counters differential power analysis on smart cards. Using dual-rail logic for sensitive variables and masking the lower 6 bits of memory accesses, our DES implementation consumed the same power for any key and consumed 13% more energy than normal implementations (Fig. 3). Extensions are needed to mask covert channels other than power consumption. For example, an extension to the interpreter could mask memory access patterns when the FPGA accesses encrypted data external to the FPGA.

It is also possible to implement spatial and temporal isolation of processes in the FPGA. Spatial isolation restricts processes to a physical region (Fig. 4). If no communications channel exist, direct communications is impossible. Temporal isolation restricts operations from occurring concurrently. It is difficult for inactive processes to find covert communications channels.

Applications of these concepts for secret key algorithms include: (*i*) Spatial isolation operates on keys in blocks. They exist and are used only as fragments. The entire key never exists as a single entity at any given time. This allows parallelism since key operations are primarily XORs, which are bit-parallel (unlike addition). (*ii)* Symmetric key algorithms implicitly require temporal isolation, since the key is modified every round. Though the secure hash algorithm (SHA-1) does not have a key, initial constants used to operate on the data are stored independently as well.

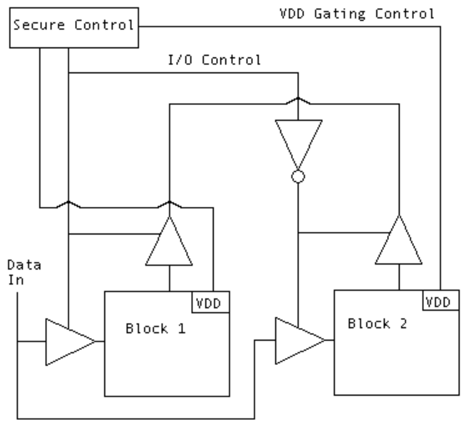

Fig. 4 Bus control is only accessible by a secure region. Blocks cannot control their own I/O. Similarly, the VDD control for each processing block is only accessible by the secure control. The control line's routing channel is accessible only to the secure control module. Bus control is mutually exclusive.