Design methods for Security and Trust

Ingrid Verbauwhede[1] and Patrick Schaumont [2]
[1] *ESAT/COSIC, Katholieke Universiteit Leuven*
[2] *Electrical and Computer Engineering Department, Virginia Tech*

Abstract

The design of ubiquitous and embedded computers focuses on cost factors such as area, power-consumption, and performance. Security and trust properties, on the other hand, are often an afterthought. Yet the purpose of ubiquitous electronics is to act and negotiate on their owner's behalf, and this makes trust a first-order concern. We outline a methodology for the design of secure and trusted electronic embedded systems, which builds on identifying the secure-sensitive part of a system (the root-of-trust) and iteratively partitioning and protecting that root-of-trust over all levels of design abstraction. This includes protocols, software, hardware, and circuits. We review active research in the area of secure design methodologies.

1 Introduction

In recent years, the mass media has repeatedly covered horror stories that describe abuse of electronic wireless tags (RFID). Weaknesses have been demonstrated that break the RFID's security by observation of RF signals [1], by clever cryptanalysis [2], or by software hacks [3]. These hacks demonstrate that, as we delegate more tasks to embedded electronics, we need these electronics to become *trustworthy*. An electronic ID card for example has to protect our personal information: it may release that information only to authorized persons.

This paper discusses implementation aspects of trust and security in embedded electronics. Trust and security are not a physical property like power, timing and area. This makes trust and security difficult to quantify. Indeed, secure design involves minimizing a risk instead of optimizing a quantity.

We will outline a methodology to support design for trust and security. Designers are already well aware with design for low-power, high-performance, and so forth. But secure electronic design is still an ad-hoc process. With a methodology for trust and security, we obtain systematic protection against the wide array of possible hacks.

The paper is organized as follows. We will first present a more accurate definition of trust, and explain two unique features of secure embedded systems: the root-of-trust and the security policy. In section 3, we review the scope of so-called 'attacks' that a secure embedded system has to withstand in order to remain trustworthy. As it turns out, all levels of design abstraction (protocols, software, micro-architecture, circuits) are susceptible to attacks. This leads to the methodology presented in section 4, called the Tree of Trust. By recursively partitioning the system into trusted and non-trusted parts over different abstraction levels, we can apply systematic countermeasures at each asbtraction level. In section 5, we will review some of the available countermeasure techniques and conclude the paper.

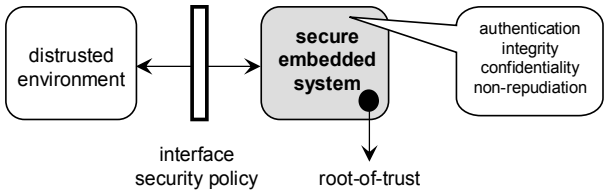

Figure 1: A secure embedded system implements a security policy to protect a root-of-trust.

2 Secure and ubiquitous embedded systems

A system S is said to *trust* a second system T when S makes the assumption that T will behave exactly as S expects [4]. A trusted system is one whose failure can make or break a *security policy* [5]. A security policy is a brief and clear description of the protection properties that a system must have, such as for example how secret keys must be managed.

Anderson points out two important aspects to a trusted system. First, trust is different from *trustworthy*. In a trustworthy system, the security policy does not fail, while the same is not always true for a trusted system. Secondly, trust is linked to the second systems' behavior and is thus also limited to the area of that behavior. It would not be the first time that a device is used (and trusted) for behaviors outside the originally defined security policy.

In the context of secure embedded devices, a system or a component (in hardware or software) is trusted if it provides a predictable and reliable behavior. This is achieved by means of a secure *implementation*. As shown in Figure 1, a secure implementation is one that protects a root-of-trust with a security policy. The security policy defines how the root of trust may be accessed. The policy

978-3-9810801-2-4/DATE07 © 2007 EDAA

relies on cryptographic techniques to provide authentication, confidentiality, integrity, and non-repudiation.

The *root-of-trust* is the core component upon which the trust and the security policy are based. The notion of a root-of-trust exists at multiple abstraction levels in a system, and can be software as well as hardware.

Consider the support of security of wireless cell phone conversations [6]. The root-of-trust is implemented in the cell phone's SIM card. The security policy requires conversation-confidentiality, as well as caller authentication to avoid impostors from taking on the identity of someone else and charging calls. The complete cell phone security policy is very elaborated and captured in standards, such as the 3GPP standards from ETSI [7].

Even tiny secure embedded systems use a root-of-trust and a security policy. RFID tags for example can be used to support anti-counterfeiting [8]. They use a security protocol based on public key cryptography, and their root-of-trust is based on a Physically Unclonable Device (PUF). It is a challenge to provide these security features in an area- and power-limited environment such as RFID.

These examples illustrate the challenge facing the embedded system designer. While optimizations for time, memory, power or energy are well known to the designer, security adds an extra dimension. A skilled attacker will try to find the weakest link, and will probe the security policy of the embedded system for weaknesses at all levels of abstraction, until he finds one that provides access to the root of trust. The next section describes some of the possibilities that the attacker has available.

3 Side-channel attacks

This section briefly reviews the security risks for embedded systems, emphasizing the risks resulting from side-channel information leaks, which are relatively easy to exploit. In general, the embedded-system-designer could consider different security levels (e.g. FIPS 140) and different protection levels for a particular implementation.

Cryptographic algorithms by themselves are only subject to cryptanalysis. The *implementation* of cryptographic algorithms on the other hand may result in side-channel information leaks. A side-channel is a bypass around the security policy that provides direct access to the root-of-trust. Side-channels information leaks can be accidentally introduced during the design process. Indeed, crypto-implementations consume time and energy, and these consumption patterns may reveal the actual secrets that are processed. Consider the following simple example of a modular exponentiation algorithm by Kocher [9]. This algorithm evaluates $R = y^k \bmod n$, with k being the equivalent of a private key.

```
modexp(in k, in y, out R_{w-1}) {
    s_0 = 1
    for j = 0 to w - 1
        if (bit j of k) is 1 then
            R_j = (s_j . y) mod n
        else
            R_j = s_j
        S_{j+1} = (R_j)^2 mod n
    end for
```

When this algorithm executes in software, the value of k decides, bit by bit, which branches of the if-then-else statement execute. These branches are easy to distinguish since the multiplication $(s_j.y) \bmod n$ requires more computations on the processor than the simple assignment of s_j. An attacker could use the execution timing or the power profile of the processor to obtain the secret value k.

There is no shortage of attack ideas and proposals for countermeasures of each type of attack. Zhou counts no less than 200 papers on power analysis attacks and relevant countermeasures [10]. Interestingly, there are next to no efforts that investigate the role of all these countermeasures in the design flow of a complete embedded system. Yet this is crucial, as a secure embedded system is only as good as the weakest side-channel link contained within it.

Table 1 presents a classification of side-channel attacks based on the particular abstraction level chosen by the attacker. At the most abstract level, the attacker is only able to access the underlying machine (also called the interpreter) that implements a cryptographic algorithm. At the next level, the attacker takes the execution time of crypto-operations into account. This can be done be measuring the execution time between externally observable events such as input-output, program start and

Table 1: A hierarchy of attacks on secure embedded systems.

Abstraction Level	Attacker Proximity	Countermeasure	Example Attack
Abstract	Off-line	Algorithm	Cryptanalysis
Interpreter	Connected	Protected Partition	Software Tampering [11], Scan-Chain readout, Fault Attack
Time	Connected	Constant-Time Design [12]	Software timing, Cache Misses [13], Branch predictions [14]
Power	Close-range	Constant-Power Design	Differential and Higher -order Differential Analysis [15]
Radiation	Close-range	Electromagnetic Shielding	Remote Power Analysis
Implementation	Physical	Physical Shielding	Circuit Tampering, Semi-Invasive attacks [16]

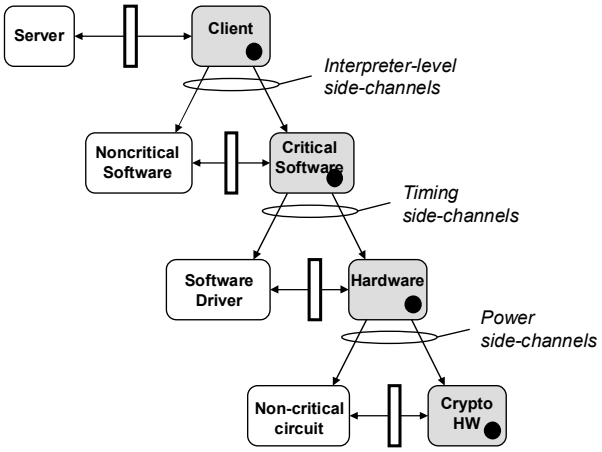

Figure 2: The tree-of-trust supports systematic implementation of the root-of-trust.

completion, and so forth. One level further the attacker also takes into account the energy consumption. This can be either electrical energy obtained through power measurements, or electromagnetic energy obtained through radiation measurements.

Researchers have demonstrated attacks at each of these levels, and they can be far more efficient than cryptanalytic attacks. For example, a brute-force attack on the 128-bit key of an Advanced Encryption Standard (AES) is infeasible with current technologies. However, a side-channel attack may still work when a brute-force attack fails. The cache-timing attack presented by Osvik in [13] extracts the 128-bit key out of a software AES cipher in 3 seconds. Similarly, the branch-prediction attack presented by Aciicmez reveals an RSA key in fractions of a second [14]. The only good response to these risks is a side-channel-aware integration of cryptography with embedded system design.

4 The Tree of Trust

Side-channel attacks may jeopardize a system's security policy at multiple levels of abstraction. It is impossible to protect a system against all these side-channel attacks with a single countermeasure. Instead, we need a *systematic* deployment of countermeasures that will protect the root-of-trust at different levels of abstraction. Indeed, the root-of-trust can have multiple embodiments, depending on the system under consideration. For example, the hardware registers used to store a private key are roots-of-trust. A key agreement protocol that manipulates this private key will also contain a root-of-trust. Figure 2 illustrates the steps taken by a designer to implement (or refine the implementation of) a root-of-trust. These steps are required for each abstraction level under consideration.

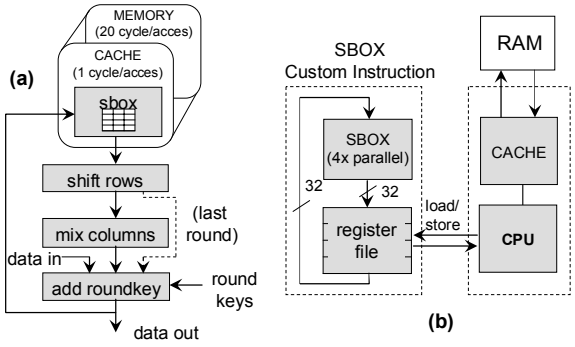

Figure 3: (a) AES data flow illustrating the timing leak (b) A security-partitioned implementation on ASIP.

- *Secure partitioning* is based on analysis of the side-channels in the design. For example, an encryption algorithm in software may be sensitive to timing analysis attacks. A designer can isolate the part of the program that is a potential timing-analysis target, and partition the software program into a secure and a non-secure process. This allows the designer to concentrate on the timing-analysis-sensitive part and further harden that part, for example by translating part of that program into hardware with constant- execution-time.
- *Secure integration* combines the non-secure part and the secure part of a design together in a manner that is consistent with the security policy and that guards the root-of-trust within the security-critical part. A secure interface resides in between the two parts and ensures that the root-of-trust remains confined within the secure part. This way, only the secure part of a design is sensitive to further side-channel attacks.

The Tree-of-Trust is a recursive application of secure partitioning and secure integration over multiple levels of abstraction. The abstraction levels of interest are the protocol level, the software-intensive architecture level, the hardware-intensive micro-architecture level and finally the circuit level. Each level roughly corresponds to a particular class of side-channel attacks, starting with interpreter-level attacks and working down to power attacks.

We provide an example of security partitioning for a software implementation of the AES (Advanced Encryption Standard) cipher. Osvik [13] and several other authors have pointed out that the lookup tables used in AES (S-boxes) are a timing side-channel into the roundkey as illustrated in Figure 3a. Depending on the presence of an S-box entry into the processor's cache, the execution time of the AES algorithm shows small variations. From the dataflow in Figure 3a, it follows that the index of an S-box is directly correlated to the AES roundkey. As a result, the execution time variations of an AES algorithm are

Table 2: Comparison of original design and security-partitioned ASIP design.

	Base processor	+ HW S-box
Avg Cycles / Round	3,515 cycles	1,520 cycles
D-cache misses	162 (variable)	2 (constant)
Stall cycles	602	342
Constant-time	No	Yes
Relative performance	100%	231%

correlated to the key, and this is used as the basis for so-called cache-attacks. Bernstein argues that high-performance, constant-time software is a challenge to write [17]. We therefore suggest a solution as in Figure 3b: the S-box is implemented with dedicated hardware, attached to the processor through a dedicated interface. This can be implemented by means of a custom-instruction-set processor. Table 2 shows the result of S-box custom instructions on a 32-bit ASIP. Because a hardware S-box eliminates all data-cache misses during execution of AES, the AES algorithm achieves a constant execution time. Moreover, since the custom S-box instruction implements 4 S-box accesses in parallel, the performance of AES encryption improves with a factor of 2.31. With this transformation, the root-of-trust is protected against timing attacks. Based on the Tree of Trust, the design can now be further partitioned. The next step would be to consider power-based side-channels in the CPU and in the hardware S-boxes. This can result in further partitioning and the introduction of constant-power digital logic.

Simple and effective side-channel countermeasures are essential to support the partitioning process in the tree of trust. The next section will review several side-channel-attack countermeasures at different levels of abstraction.

5 Side-channel resistant design and design methods

Systematic deployment of countermeasures will protect the root-of-trust at different levels of abstraction. This is not an easy task, as designers and engineers are trained to work towards an optimization goal, with the help of design methods. Design for security on the other hand attempts to minimize risk (i.e. prevent something to happen). In this section we will focus on countermeasures which should be of interest to the digital designer and which could be supported by EDA tools.

5.1. Physical level tamper resistance

At the physical level, manufacturers have developed tamper proof techniques such as putting the security parts into special casings with light, temperature, tampering and/or motion sensors depending the application. A top layer sensor meshes is an extra layer of metal on top of the integrated circuit which is continuously monitored for interruptions and shorts [18]. It prevents laser cutting or selective etching to access the internals of the circuit.

Extremely important is the link to test and testability: scan chains and JTAG ports give are an effective side entry into the internals of the integrated circuit. Secure circuits therefore have limited testability, such as certain forms of BIST.

5.2. Circuit level techniques to resist side channel leakage

Countermeasures against tampering don't work against the side channel leakage through power, timing or electromagnetic radiation of the devices. The fundamental reason is that integrated circuits are made of CMOS technology. Standard cell based design is very successful because the standard CMOS circuit style is very robust against clock variations or fluctuations in the power supply. And it is really low power, because it only consumes power when the circuits operate at least to a first degree when neglecting the subthreshold leakage issues associated with deep submicron technologies.

On top, it is supported by a very successful design flow: a design is written at register transfer level in VHDL or Verilog and an 'automatic' tool flow (optimistically) translates this to a GDS file ready for tape-out.

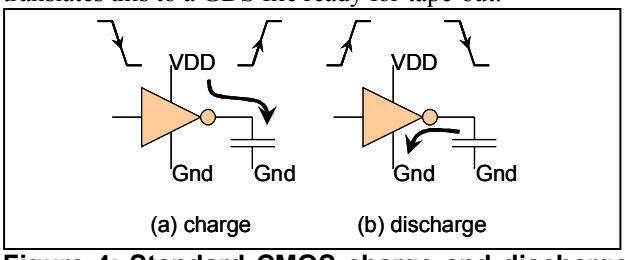

Figure 4: Standard CMOS charge and discharge event.

As shown in Figure 4, a CMOS circuit really tells what it is doing by looking at the current supplied to the circuit. An attacker can detect a charging, discharging action or no action. A solution at the circuit level is to use logic styles that have a power consumption pattern that is independent from the data being processed. When logic values are measured by charging and discharging capacitances, we need to use a fixed amount of energy for every transition. This can be obtained by full-custom dynamic, differential logic styles with balanced load capacitances. But more practical from a design perspective is an approach that can be integrated with a regular design flow. It has been the focus of European projects such as ASYNC and SCARD [19]. It is illustrated with the wave dynamic differential logic style WDDL and its associated design flow. A WDDL gate will show exactly one transition every clock cycle. And a WDDL design can be integrated in a regular standard CMOS design flow as shown in Figure 5. A few scripts are added to the flow.

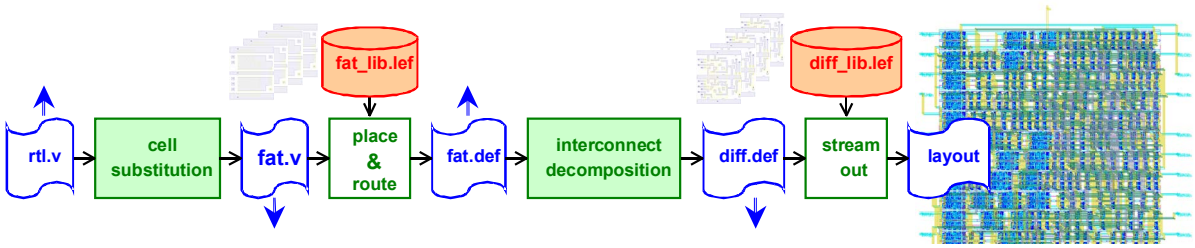

Figure 5: Design flow for side-channel resistant circuits based on power.

A first script translates a regular VHDL net list to a so-called fat net list. This net list contains differential WDDL type gates. In this library each cell has differential inputs and differential outputs. To the place and route tool, these inputs and outputs are presented as single 'fat' (i.e. double width) inputs and outputs. A regular place & route tool is used to route the fat wires. Afterwards a second script is run that decomposes the fat wires into two differential wires with an identical wiring pattern.

A WDDL based circuit has an area that is 3 times larger and the power consumption is 4 times higher. The benefit is that a differential power analysis attack on an AES implementation is not successful even after 1.5M encryptions. A functionally identical AES discloses the key after an average of 2000 encryptions [20].

In literature, other approaches and circuit styles are proposed: e.g. based on asynchronous logic or masked dual-rail precharge logic (MDPL) [21]. The area and power consumption of MDPL is even higher: a factor 10 compared to regular standard cells, but it has the advantage that it does not depend on differential routing.

5.3. Simulation for side channel attacks

Accurate simulation is a crucial tool in the development of effective countermeasures. We briefly describe the issues for logic-level simulation, but the remarks are also valid at other abstraction levels [22].

The efficiency of logic simulators is obtained by abstracting lower-level technology effects during simulation. For example, cycle-based design ignores races, glitches, and other sub-cycle timing effects. It also abstracts electrical signals into discrete logic values, thereby loosing the ability to evaluate the power consumption. Circuits that have perfect resistance at the simulated logic level are prone to side-channel attacks when implemented afterwards due to glitches, imbalances in capacitive loads or other lower level effects.

The ideal simulator to evaluate power-based side-channels at logic level has the following characteristics. First, it should provide a power-trace over time, rather then a single overall power value. Second, the power-trace should be calibrated. Calibrated simulation is very hard to obtain if only approximate circuit loading capacitances are known. Yet the calibration is required to evaluate the magnitude of the side-channel signal, and derive the amount of *real* measurements required for a successful side-channel attack. Third, the simulator should cover not only a single instance of the circuit, but should also take technology-dependent variations into account. A simulator that covers all these features remains an open research problem.

5.4. Secure micro-architectures

Figure 3 shows one example of a side channel secure SBOX implementation. It is a data path example. But the control sequences need also to be balanced. The if-then-else operation in the modular exponentiation algorithm given in section 3 should be balanced. Section 3 shows a traditional right-to-left binary multiply-and-square algorithm. Similarly a left-to-right square-and-multiply exists. The Montgomery ladder balances this as follows: the if-branch and the else-branch both execute a square and a multiply. The difference is in the assignments to registers.

```
exp(in k, in y, out R0) {
    R0 = 1; R1 = y;
    for j = w-1 to 0
        if (bit j of k is 0) then
            R1 = R0.R1; R0 = R0.R0;
        else [bit j = 0]
            R0 = R0.R1; R1 = R1.R1;
    end for
    return R0;
```

For ease of notation, the mod n operation is not added to the equations. At first sight, the Montgomery ladder seems inferior because it requires w multiplications instead of ½ w on average. But it does provide side channel resistance to simple attacks and it has other benefits [23].

5.5. Secure micro-processors

High-end embedded systems contain large amounts of software. In a secure embedded system, a microprocessor needs to provide multilevel security (different software processes have different privileges) as well as multilateral security (different software processes are kept apart) [5]. Typical modern processors provide privileged-user modes to support multi-level security. However, this is a coarse

approach, unsuited for most applications. New intiatives like ARM Trustzone [25] take a fine-granular approach.

The processor's micro-architecture is a rich source of side-channels. They can be found in the cache [13], in the branch prediction [14], and the memory organization [12]. They can be addressed by improving (modifying) the microprocessor architecture, such as for example is done in the SecureCore project [24].

Finally, the programmable nature of processors introduces yet challenge: how to ensure the integrity of the link between application software and the processor that executes that software? This is one of the concerns of trusted computing [26]. AEGIS is an example of a trusted micro-architecture that implements software integrity by means of platform-unique functions [27].

5.6. Secure algorithm techniques

There is also plenty of research at the algorithm level to protect implementations from side channel leakage. A very typical example is "key blinding" also called "exponent blinding" [28]. Advanced side channel attacks (differential and higher) require the measurements of multiple encryptions to attack a device. These multiple encryptions are needed to filter the signal out of the very noisy signals e.g. by correlation. However, if for every exponentiation (in RSA) or for every point multiplication (in Elliptic Curve Cryptography) a different exponent (for RSA) or different secret scalar (for ECC) is used, then it is not possible to collect sufficient measurements. This is in essence the goal of key blinding. Instead of calculating

$$Q = k.\ P$$

the ECC system will calculate:

$$Q = k'.\ P \text{ with } k' = k + r.N$$

with N =the number of points on the elliptic curve and r a random number. If for every calculation a new random number is chosen, then multiple measurements with the same secret key are not possible.

6 Conclusions and Outlook

If we want trustworthy ubiquitous embedded devices, we need to take security into account during the design of these devices. In this paper we advocate a root-of-trust model together with a security policy. This model is hierarchically refined over all design abstraction layers. It is illustrated with several practical examples. Optimizing for security and trust is different from optimizing for power, area or speed. And there remain many challenges for a more secure digital design supported by secure design methodologies.

Acknowledgement

This work was partially supported by NSF awards 0644070 and 0541472, FWO projects G.0475.05 and G.0300.07 and funds from the K.U.Leuven.

References

[1] R,. Merritt, "Cellphone could crack RFID tags, says cryptographer," Electronic Engineering Times, 14/2/06.
[2] J. Schwartz, "Graduate Cryptographers Unlock Code of 'Thiefproof' Car Key," New York Times, Section A, p. 14, 1/29/05.
[3] J. Markoff, "Study Says Chips in ID Tags Are Vulnerable to Viruses," New York Times, Section C, p. 3, 3/15/06.
[4] H. van Tilborg, "Encyclopedia of Cryptography and Security," Springer, 2005.
[5] R. Anderson, "Security Engineering: A guide to building dependable and distributed systems," Wiley Computer Publishing, 2001.
[6] K. Vedder, "GSM: Security, Services and the SIM," LNCS 1528, Springer, 1998.
[7] 3G Security: Security Architecture, Universal Mobile Telecomm. System (UMTS), tech. specification 3GPP TS 33.102, 1999.
[8] P. Tuyls, L. Batina, "RFID-Tags for Anti-counterfeiting," RSA 2006 Cryptographers' Track, LNCS 3680, p. 115-131.
[9] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems," Proc. CRYPTO '96, Lecture Notes on Computer Science, 1109:104:113, Springer-Verlag, 1996.
[10] Y. Zhou, D. Feng, "Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing," Cryptology ePrint Archive, Report 2005/388.
[11] P. vanOorschot, et al., "Hardware-Assisted Circumvention of Self-Hashing Software Tamper Resistance," IEEE Transactions on Secure and Dependable Systems, 2(2):82—92, April-June 2005.
[12] B. Chevallier-Mames, et al., "Side-channel atomicity," IEEE Trans. on Computers, 53(6):760—68, June 2004.
[13] D. Osvik, A. Shamir, E. Tromer, "Cache Attacks and Countermeasures: the Case of AES," Proc CT-RSA, LNCS 3860, 1—20, Springer, 2006.
[14] O. Aciicmez, C. Koc, JP. Seifert, "On the Power of Simple Branch Prediction Analysis," Cryptology ePrint Archive, Report 2006/351.
[15] S. Mangard, E. Oswald, T. Popp, "Power Analysis Attacks: Revealing the secrets of smart cards," Springer, March 2007.
[16] S. P. Skorobogatov, "Semi-invasive attacks: a new approach to hardware security analysis," University of Cambridge, Technical Report UCAM-CL-TR-630, April 2005.
[17] D.J. Bernstein, "Cache-timing attacks on AES," preprint, 2005, online at http://cr.yp.to/papers.html
[18] O. Kömmerling, M. Kuhn, "Design principles for Tamper-Resistant Smartcard Processors," Proc. of the USENIX workshop on Smartcard technology, pp. 9- 20, May 1999.
[19] Aigner M., et al., "Side Channel Analysis Resistant Design Flow", Proc. ISCAS 2006, pp. 2909-2912, May 2006.
[20] D. Hwang et al., "AES-Based Security Coprocessor IC in 0.18-um CMOS with Resistance to Differential Power Analysis Side-Channel Attacks," IEEE JSSC 41(4), pp. 781-792, 2006.
[21] T. Popp, and S. Mangard, "Masked Dual-Rail Pre-Charge Logic: DPA-Resistance without Routing Constraints", Proc CHES 2005, LNCS , Springer Verlag, 2005.
[22] K. Tiri, I. Verbauwhede, "Simulation Models for Side-Channel Information Leaks," Proc. 2005 Design Automation Conference (DAC 2005), pp.228-233, June 2005.
[23] M. Joye, S.-M. Yen, "The Montgomery powering ladder," Proc. CHES 2002, LNCS 2523, pp. 291-302, 2002.
[24] Z. Wang and R. B. Lee, "Covert and Side Channels due to Processor Architecture," Proc. ACSAC'06, pp.473-482, December 2006.
[25] T. Alves, D. Felton, "Trustzone: Integrated Hardware and Software Security," ARM white paper, July 2004.
[26] E. Gallery, "An overview of trusted computing technology," in Trusted Computing, eds. C. Mitchell, IEE press, 2005.
[27] E. Suh, C. O'Donnell, I. Sachdev, S. Devadas, "Design and Implementation of the AEGIS single-chip secure processor using physical random functions," Proc. ISCA 05, p. 25-36, June 2005.
[28] J.-S. Coron, "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems", CHES 99, LNCS 1717, pp. 292-302.