

HW/SW Co-design of TA/SPA-resistant Public-key Cryptosystems

K. Sakiyama*, L. Batina*, P. Schaumont[†] and I. Verbauwhede*

*ESAT-COSIC, K.U. Leuven, Belgium.

[†]ECE Dept., Virginia Tech, USA.

*{Kazuo.Sakiyama, Lejla.Batina, Ingrid.Verbauwhede}@esat.kuleuven.be

[†]schaum@vt.edu

Abstract. This paper proposes a hardware/software (HW/SW) co-design methodology for secure Public-Key Cryptosystems. Our design flow allows to assess the risk for simple side-channel attacks including Timing Analysis (TA) and Simple Power Analysis (SPA) at an early design stage. It also allows to evaluate the quality of countermeasures against these attacks. The HW/SW co-design is illustrated with an Elliptic Curve Cryptosystem (ECC) over $GF(p)$. By balancing the execution time in SW and HW, a first order side-channel-resistant design can be obtained with a 15% increase in latency for a point multiplication.

Keywords: HW/SW co-design, TA-attacks, SPA-attacks, ECC implementation

1 Introduction

The implementation of Public-Key Cryptography (PKC) is a challenge in embedded systems such as smartcards, RF-ID tags, and mobile phones. They have limited silicon resources and a limited power budget. For this reason, the short key-lengths of ECC (proposed by Koblitz [1] and Miller [2]) are often preferred over the more traditional RSA-based systems. From a Side-Channel Attack (SCA) point of view, both ECC and RSA need to be resistant to protect private information (*e.g.* a secret key in a memory).

HW/SW co-designs are attractive especially for PKCs because they can take advantage of the flexibility of SW and at the same time use the performance offered by HW. In this paper, we are concerned with a code-design flow that will take SCA into account in the HW/SW partitioning and the design of the interface.

1.1 Side-Channel Attacks

Several types of SCAs have been introduced. Anderson and Kuhn summarized a number of attacks [3]. Among the SCAs, the most straightforward attack is a physical attack directly to the silicon. This is a powerful method if a probing point is available. For instance, it is easy to retrieve secret information by probing the data on a bus. The fault induction attack [4] is a well-known technique that works by disturbing the device to induce errors in the computation. These attacks are named active attacks after the technique.

Possibly more dangerous type of attack, undetectable for the embedded system, is the passive attack. This type is based on measuring physical characteristics leaking from side-channels of the embedded systems. Timing Analysis (TA) attacks check the computation time. If the execution time varies with the data or the key used in the computations, this can be detected by the attacker [5]. Simple Power Analysis (SPA) attacks measure the power consumption during cryptographic operations and guess the actual types of computations. In [6], Kocher, Jaffe, and Jun introduced differential power analysis (DPA) that also considers effects correlated to data values. Electromagnetic analysis (EMA) attacks [7] [8] and Acoustic analysis (AA) attacks were also introduced as effective passive SCA examples [9].

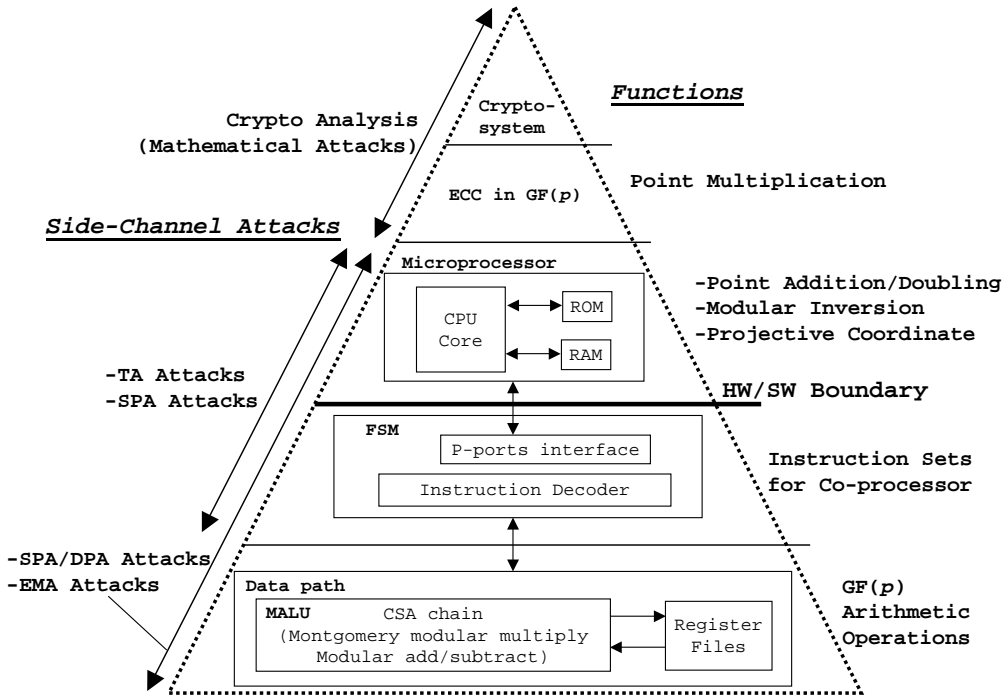


Fig. 1. Security pyramid : possible SCAs and required functions for each abstraction level of ECC over $GF(p)$.

1.2 Our design approach

Fig.1 shows how SCA and HW/SW co-design are related, illustrated for the case of PKCs. A PKC is decomposed into multiple layers of operation (point multiplication, addition/doubling, $GF(p)$ operations) which each map into HW or SW. Possible SCAs are shown on the left side of the pyramid. Starting from mathematical attacks, each abstraction level can potentially be attacked corresponding to its functionality. Especially, the HW/SW boundary can be a serious threat to ECC since the boundary is easy to identify, and an obvious interface point in the system.

The HW/SW boundary situates at the cycle-true, Register Transfer (RT) level of design. At that level, simple SCAs can be detected and countermeasures can be implemented. Of course, once a design is made resistant to the simple SCAs, the differential ones and higher order ones need to be addressed. This requires however solutions at a lower level of abstractions, *e.g.* at the gate or transistor level [10].

1.3 Paper Overview

The remainder of this paper is as follows. Section 2 explains the co-design flow proposed in the paper. Section 3 presents system overview for ECC over $GF(p)$. The HW co-processor has a Modular Arithmetic Logic Unit (MALU) used for accelerating operations in the finite field $GF(p)$. The SW relies on a co-processor attached to the 8051. Section 4 shows a case study of simple SCA and the corresponding defense against it on an 160-bit ECC over $GF(p)$ (ECC-160p) and Section 5 concludes the paper.

2 A Secure Co-Design Flow

In this section, we look at a HW/SW co-design approach for system design with consistent resistance against SCAs. In designs that include both SW and HW components, the natural fit is for SW to support flexibility and for HW to support performance and energy/efficiency. Frequently, flexibility and performance constraints are in competition, leaving the designer to decide on an optimal partitioning between HW and SW. Under consideration of SCAs, also the risk of attack needs to be included besides flexibility and performance constraints. The time-multiplexed nature of SW makes it more susceptible to TA attacks or SPA attacks (here we call it simple SCAs). HW on the other hand can easily be made to perform with constant-timing behavior and thus will show better resistance against simple SCAs. In our design flow, we make use of this knowledge to decide on a proper boundary between HW and SW.

2.1 The Co-simulation Tool

We use a co-simulation environment, called GEZEL [11], that allows us to estimate immediate dynamic power consumption. We use toggle count per clock cycle (TCPC) as an approximation for this power. The toggle count is obtained directly out of the RT-level model and does not require synthesis of the model. Despite this approximation, our experiments have shown that it is sufficient to build cryptosystems that are safe to simple SCAs. As mentioned before, higher order attacks need to be addressed at lower levels of abstraction. The co-design approach gives the designer an environment to get a quick and correct evaluation of first order attacks.

Cryptographical algorithms, including public-key ciphers, are typically first developed as C/C++ programs. The first step is to partition these C/C++ programs into a HW part, described as RT-level, and a SW part. The partitioning is based on SW performance analysis on the one hand, and on insight into the cryptographical algorithm on the other. The PKC illustrated in Fig.1 for example is layered in logical abstraction levels that each fit nicely into either HW or SW. The GEZEL design environment allows us to try out different alternatives, and to co-simulate for each partition the HW and the SW as a cycle-true model. An instruction-set simulator (ISS) is used to obtain cycle-accurate SW performance with accurate modeling of the HW/SW interface. At this level, we also obtain toggle counts of the HW model as a function of clock cycle. These toggle counts allow us to analyze the risk for simple SCAs and, if detected, to rewrite the SW and/or the HW. After cycle-accurate co-simulation, the HW model is converted into VHDL, and synthesized by a secure back-end. In the next subsection, we briefly discuss the mechanism for toggle counting at the RT-level.

2.2 RT-level toggle counting

The HW descriptions in GEZEL are expressions of cycle-true register transfers, containing operations and assignments on signals and registers. We obtain toggle count estimates directly on these expressions, by means of a simple set of rules:

- The TCPC for a register or a signal is the Hamming distance between the value during the previous clock cycle and the value during the current clock cycle. In the case of a register, this toggle count is measured at the register input.
- The TCPC of a simple operation is given by the Hamming distance of the previous operation result and the current result. The toggle count is measured at the operation output. Simple operations includes additions, subtractions, shifts, multiplications, and so on.
- The TCPC of an expression composed of simple operations is given by the sum of TCPC of individual operations.

For example, assume an RT-level expression as follows:

```

signal  a;
register b;
a = 3;
b = b + a;

```

This piece of code contains three operations: two assignments and an addition. In the first clock cycle, signal **a** changes from 0 to 3. The addition operation output will be 3 as well, and this value will be assigned to register **b**. The total toggle count for the first clock cycle thus equals 6. In the second clock cycle, signal **a** does not change value. The output of the addition operator will now change from 3 to 6 however (Hamming distance '110' - '011' = 2), and register **b** will change from 3 to 6 as well. The toggle count for the second clock cycle thus equals 4. We can continue in this way to obtain an approximation of the immediate dynamic power consumption. This methodology is very simple, and for example does not take glitching nor the implementation complexity of operators into account. For the purpose of simple SCAs on the other hand, it is adequate.

3 System architecture for ECC operations over $GF(p)$

First, the introduced MALU and the implemented software of ECC over $GF(p)$ is shortly given. Second, potential vulnerability in the ECC algorithm is explained. Third, possible countermeasures for simple SCAs are introduced.

3.1 Modular Arithmetic Logic Unit (MALU)

The proposed MALU is a Carry-Save Adder (CSA) based Montgomery modular multiplier. It is composed of regularly allocated cells as illustrated in Fig.2. The 5-3 CSAs are used to sum up five-bit inputs that are $xy, mn, s, c0$, and $c1$ and outputs three bits for the redundant CS-form that has a value of $2(c0_{next} + c1_{next}) + s_{next}$ where s and $c0/1$ are the virtual sum and carries respectively. The bit multiplication xy and mn are main inputs for computing bit level of Montgomery multiplication, *i.e.* $(xy + mn)/2$ [14]. The behavior of the `cell(i, j)` is explained by the following equation where d is the number of CSAs concatenated in i direction.

$$\begin{aligned}
2^{d+1}c1_{i+1,j+1} + 2^d c0_{i+1,j} + s_{i,j-1} &= x_i y_j + m_i n_j + c0_{i,j} + c1_{i,j} + s_{i,j} \\
m_i[0] &= c1_{i,j} \oplus c0_{i,j} \oplus x_i[0] y_j \oplus n_j \\
m_i[l] &= c1_{next} \oplus c0_{next} \oplus s_{i,j}[l] \oplus x_i[l] y_j \oplus n_j
\end{aligned} \tag{1}$$

Input/output vectors for such bit-level variables are described as $X, Y, N, S, C0$, and $C1$, where X and Y are the multiplicands, N is the modulus, and $S, C0$, and $C1$ are intermediate variables representing a redundant CS-form. The MALU has two independent stages. One is the Carry-Save(CS)-stage that operates Montgomery algorithm in a redundant CS-form. Another stage converts the CS-form integer into a normal integer by propagating carries, namely the Carry-Propagate(CP)-stage. The combination of the $MALU_N$ and CP_N allows all basic modular computation necessary for Elliptic Curve (EC) point multiplication over $GF(p)$ as shown in Eq.2. This unit will be implemented in a HW co-processor and separate care needs to be taken that it has a constant execution time.

$$\begin{aligned}
MALU_N(XR, YR, SR) &= (XY + S)R \bmod N \\
CP_N(AR, BR) &= (A + B)R \bmod N
\end{aligned} \tag{2}$$

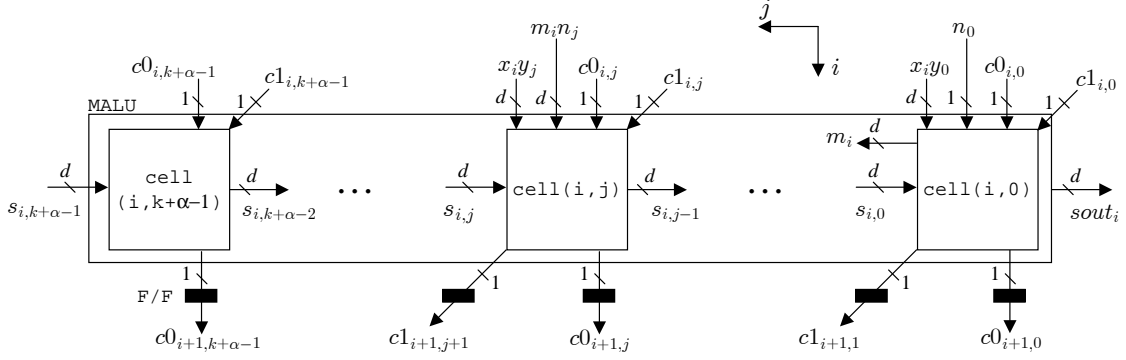


Fig. 2. Schematic of the MALU in the co-processor.

3.2 Point Addition/Doubling

Point addition and doubling can be performed according to the algorithm given in [12]. Here we assume that the two points that will be added, *i.e.* $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ are already transformed to the weighted projective coordinates (Jacobian representation) and Montgomery representation, where (X, Y, Z) corresponds to the affine coordinates $(X/Z^2, Y/Z^3)$. The result point is stored as Q , *i.e.* $Q \leftarrow P + Q$.

We now create a schedule for point addition using a single MALU. The following scheduling as shown in Table 1 can be used. Point doubling is considered as a special case of point addition, *i.e.* $Q \leftarrow 2Q = Q + Q$. In Table 2 a possible schedule for point doubling is given. For efficient computing of point addition and doubling, four additional registers storing intermediate variables are provided. These are stored in the co-processor RAM.

Table 1. Scheduling of point addition.

POINT ADDITION	COST
$t_1 = \text{MALU}_N(Z_1, Z_1, 0)$	1M
$t_2 = \text{MALU}_N(X_2, t_1, 0)$	1M
$t_3 = \text{MALU}_N(Z_2, Z_2, 0)$	1M
$t_4 = \text{MALU}_N(X_1, t_3, t_2)$	1M
$t_2 = \text{CP}_N(2N + 1, t_2, 0)$	1A
$t_2 = \text{MALU}_N(X_1, t_3, t_2)$	1M
$t_1 = \text{MALU}_N(t_1, Z_1, 0)$	1M
$t_1 = \text{MALU}_N(t_1, Y_2, 0)$	1M
$Y_2 = \text{MALU}_N(t_3, Z_2, 0)$	1M
$t_3 = \text{MALU}_N(Y_2, Y_1, t_1)$	1M
$t_1 = \text{CP}_N(2N + 1, t_1, 0)$	1A
$Y_2 = \text{MALU}_N(Y_2, Y_1, t_1)$	1M
$t_1 = \text{MALU}_N(t_2, t_2, 0)$	1M
$t_1 = \text{MALU}_N(t_4, t_1, 0)$	1M
$t_4 = \text{CP}_N(2N + 1, t_1, 0)$	1A
$X_2 = \text{MALU}_N(Y_2, Y_2, t_4)$	1M
$t_4 = \text{MALU}_N(2, X_2, 0)$	1M
$t_4 = \text{CP}_N(2N + 1, t_4, 0)$	1A
$t_4 = \text{MALU}_N(1, t_1, t_4)$	1M
$t_1 = \text{MALU}_N(t_2, t_2, 0)$	1M
$t_1 = \text{MALU}_N(t_1, t_2, 0)$	1M
$t_1 = \text{MALU}_N(t_3, t_1, 0)$	1M
$t_1 = \text{CP}_N(2N + 1, t_1, 0)$	1A
$Y_2 = \text{MALU}_N(t_4/2, Y_2, t_1/2)$	1M
$t_1 = \text{MALU}_N(Z_1, Z_2, 0)$	1M
$Z_2 = \text{MALU}_N(t_2, t_1, 0)$	1M
TOTAL	21M + 5A

Table 2. Scheduling of point doubling.

POINT DOUBLING	COST
$t_1 = \text{MALU}_N(X_2, X_2, 0)$	1M
$t_1 = \text{MALU}_N(2X_2, X_2, t_1)$	1M
$t_2 = \text{MALU}_N(Z_2, Z_2, 0)$	1M
$t_2 = \text{MALU}_N(t_2, t_2, 0)$	1M
$t_2 = \text{MALU}_N(a, t_2, 0)$	1M
$t_1 = \text{MALU}_N(1, t_1, t_2)$	1M
$t_2 = \text{MALU}_N(2Y_2, Y_2, 0)$	1M
$t_3 = \text{MALU}_N(2t_2, t_2, 0)$	1M
$t_2 = \text{MALU}_N(2X_2, t_2, 0)$	1M
$t_4 = \text{MALU}_N(2, t_2, 0)$	1M
$t_4 = \text{CP}_N(2N + 1, t_4, 0)$	1A
$X_2 = \text{MALU}_N(t_1, t_1, t_4)$	1M
$t_4 = \text{MALU}_N(1, X_2, 0)$	1M
$t_4 = \text{CP}_N(2N + 1, t_4, 0)$	1A
$t_2 = \text{MALU}_N(1, t_2, t_4)$	1M
$Z_2 = \text{MALU}_N(2Z_2, Y_2, 0)$	1M
$t_3 = \text{CP}_N(2N + 1, t_3, 0)$	1A
$Y_2 = \text{MALU}_N(t_1, t_2, t_3)$	1M
TOTAL	15M + 3A

3.3 Point Multiplication

Point multiplication can be implemented as a repeated combination of point addition and point doubling. We used the simplest algorithm, *binary method*. It is well-known that *binary method* is vulnerable for SCAs because of a different computation time in the for-loop. One of the countermeasures is add-and-always-double method proposed by Coron [13]. The idea is that SCA resistance is ensured by computing both point addition and point doubling in a for-loop. This method increases the HW cost and decreases the performance because of the additional dummy point additions.

Our proposed countermeasure is described as follows. As shown previously, point addition has more computational steps than point doubling. We create a countermeasure for SCA attacks by letting them have the same number of steps. To do this, dummy MALUs and CPs are added to point doubling shown in Table 2. In addition, the co-processor is implemented to have a constant execution time for each MALU or CP operation. This is the so-called balancing of point operations. Moreover, imbalance of *binary method* can be solved by inserting dummy instructions such that intervals of point operations can be constant and toggle counts can be the same within the intervals. Namely, this is for balancing of the conditional branches in SW. It is commonly accepted that increasing the resistance against SCAs is at the cost of system performance and/or resources. The trade-off among cost, performance, and security can easily be explored by our proposed HW/SW co-design environment.

4 A case study of simple SCAs and corresponding countermeasure

Here, the result for a case study of ECC-160p implementation is discussed. Three types of metrics, cost, performance, and security for ECC-160p are estimated using the proposed design flow of GEZEL system-level co-simulation. First, HW and SW are implemented based on a strategy to find the best trade-off between performance and cost. The performance/cost-optimized result tends to be vulnerable to simple SCAs. To confirm the fact, power estimation for point multiplication is simulated with GEZEL. More precisely, the toggle counts of the co-processor is collected through the ECC operations and traced in smoothing format (average of 2000 cycles). The result is shown in Fig.3a and Fig.3b. AtoP, PtoA represent coordinate conversion from/to affine to/from projective. Representation conversions are done with NtoM and MtoN that means conversion from/to normal form to/from Montgomery form. The trace in Fig.3a clearly says which type of the ECC operations is executed. The highest risk exists in its enlarged figure (Fig.3b) for EC point multiplication. Carefully observing the trace, different shapes of peaks and valleys can be found. As mentioned in Section III, the difference is caused by the imbalanced SW/HW depending on point operations and conditional branches. Thus, the secret key is easily cracked (the cracked secret key is written up in Fig.3b).

In the second place, HW and SW are modified based on our proposed simple SCA-resistant implementation. The result is shown in Fig.3c. From the trace, it is hard to distinguish the point operation type. One of the side-effects of the countermeasures is a longer execution time because dummy MALUs and CPs are inserted in point doubling operations. Hence, this resistance comes at the price of lower performance. The more detail observation is given in the following.

The C/C++ codes are compiled with μ Vision2 by Keil Software, Inc with a target device of Intel 8051AH. The co-processor block was synthesized with Project Navigator by Xilinx and implemented on a Virtex-II PRO (XC2VP30). Two different configurations, performance/area-oriented design and simple SCA-resistant design are summarized in Table.3. The case of using non-balanced *binary method* (no simple SCA-resistance) shows on the left side of the table. The right side of the table describes the simple SCA-resistant result with the proposed countermeasure. In the countermeasure implementation, the latency of point multiplication increases

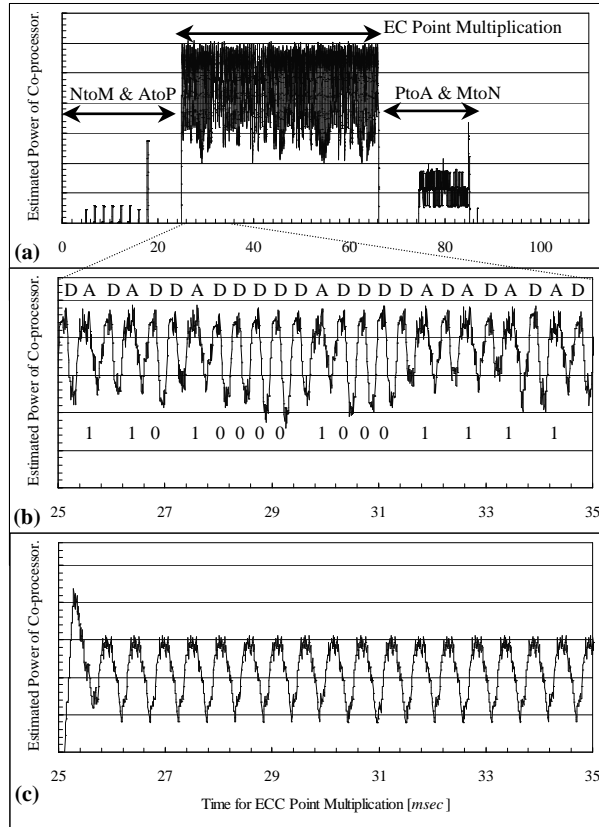


Fig. 3. Result of power simulation with GEZEL for ECC-160p. (a): Estimated power consumption for the whole ECC-160p operation. (b): The enlarged trace of (a). "D" and "A" means point doubling and point addition respectively. The cracked secret key is written below the wave. (c) simple SCA-resistant result. Point doubling and point addition are indistinguishable.

by 15% and the program size becomes slightly large (1%). However, the size of XRAM and the co-processor do not change. In this sense, it is said that our proposed countermeasure explores a trade-off between performance and security.

5 Conclusions

We have presented a secure HW/SW co-design flow for PKCs. By using the design flow, it is possible to verify the simple SCA-resistance in an early stage of a cryptographic design. ECC-160p is shown as an example of tamper-proof implementations with the proposed design flow. The result showed a trade-off among cost, performance and security by the HW/SW co-design exploration.

6 Acknowledgements

This research has been supported by a research grant of K.U. Leuven and by the FWO projects (G.0141.03) and (G.0450.04).

Table 3. Result of point multiplication of ECC-160p for different security configurations on the same co-processor.

Simple SCA-resistance	None	Yes
Total Latency[ms] @12MHz operation	91.7	105.6
8051 Resource[Bytes]		
XRAM	211	←
PROM	2,561	2,589
FPGA Mapping of Co-processor		
Number of 4-input LUTs	6,666	←
Number of Slice F/Fs	1,195	←
Number of Block RAM	6	←
Critical Path[ns]	18.8	←

References

1. N. Koblitz, "Elliptic Curve Cryptosystems," *Math. Computation*, vol. 48, pp. 203-209, 1987.
2. V. S. Miller, "Use of Elliptic Curve in Cryptography," *Advances in Cryptology: Proceedings of CRYPTO'85. Lecture Note in Computer Science, Springer-Verlag* vol. 218, pp. 417-426, 1985.
3. R. Anderson and M. Kuhn, "Tamper-resistance - a cautionary note," *Proceedings of the 2nd USENIX Workshop on Electronic Commerce. USENIX Association*, pp. 1-11, 1996.
4. D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," *Advances in Cryptology: Proceedings of EUROCRYPTO'97. Lecture Note in Computer Science, Springer-Verlag*, vol. 1233, pp. 37-51, 1997.
5. P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Advances in Cryptology: Proceedings of CRYPTO'96. Lecture Note in Computer Science, Springer-Verlag*, vol. 1109, pp. 104-113, 1996.
6. P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Advances in Cryptology: Proceedings of CRYPTO'99. Lecture Note in Computer Science, Springer-Verlag*, vol. 1666, pp. 388-397, 1999.
7. K. Gandolfi and C. Moutrel and F. Olivier, "Electromagnetic Analysis: Concrete Results," *Cryptographic Hardware and Embedded Systems: Proceedings of CHES'01. Lecture Note in Computer Science, Springer-Verlag*, vol. 2162, pp. 255-265, 2001.
8. J.-J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards," *Smart Card Programming and Security (E-smart 2001), Springer-Verlag*, vol. 2140, pp. 200-210, 2001.
9. A. Shamir and E. Tromer, "Acoustic cryptanalysis On nosy people and noisy machines," *Preliminary proof-of-concept presentation*, <http://www.wisdom.weizmann.ac.il/tromer/acoustic/>.
10. K. Tiri and I. Verbauwhede, "Simulation Models for Side-Channel Information Leaks," *42nd Design Automation Conference (DAC 2005)*, pp. 228-233, 2005.
11. P. Schaumont, "GEZEL Language Reference," ver1.7, 2004.
12. I. Blake, G. Seroussi, and N. Smart, "Elliptic Curves in Cryptography," *Cambridge University Press, London Mathematical Society Lecture Note Series* 265, 1999.
13. J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," *Cryptographic Hardware and Embedded Systems: Proceedings of CHES'99. Lecture Note in Computer Science, Springer-Verlag*, vol. 1717, pp. 292-302, 1999.
14. P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, pp. 519-21, 1985.