Secure AES Hardware Module for Resource Constrained Devices

Elena Trichina[1] and Tymur Korkishko[2]

[1] Department of Computer Science, University of Kuopio,
P.O.B. 1627, FIN-70211, Kuopio, Finland
etrichin@cs.uku.fi
[2] Information security TG, i-Networking Lab, Information Security Group,
Samsung Advanced Institute of Technology, Korea
k.tymur@samsung.com

Abstract. Low power consumption, low gate count, and high throughput are standard design criteria for cryptographic coprocessors designated for resource constrained devices such as smart cards. With the advent of side channel attacks, devices' resistance to such attacks became another major requirement. This paper describes a cryptographic hardware module for an AES algorithm that provides complete protection against first order differential power analysis by embedding a data masking countermeasure at a hardware level. We concentrate on inversion in $GF(2^8)$ since this is the only non-linear operation that requires complex transformations on masked data and on bits of the masks. The simulation and synthesis results confirm that the proposed solution is suitable for applications in GSM and ad-hoc networks in terms of performance, gate count and power consumption. To our knowledge, this is the first implementation of a side channel-resistant AES hardware module suitable for smart- and SIM-cards.

1 Introduction

In applications such as smart cards and related embedded devices, hardware complexity and tamper resistance are very important issues that directly affect the cost and consumer acceptance of such devices. When invasive attacks on smart cards had been reported [2], an industry reacted by incorporating in a chip features such as glue logic, metal shields, and sensors of abnormal behavior [15]. But it was not long before a new class of *side channel* attacks emerged as a powerful thread. These attacks enable breaking cryptographic algorithms by measuring timing characteristics [13], power consumption [12, 20] or electromagnetic radiation [9, 25] of a smart card microprocessor when it runs cryptographic applications.

Until recently, most of these attacks exploited some specific features of software implementations of cryptographic algorithms. Fittingly, most countermeasures were also designed at software level. For many applications, however, it is necessary that cryptographic algorithms should be realized in hardware. Hence,

C. Castelluccia et al. (Eds.): ESAS 2004, LNCS 3313, pp. 216–230, 2005.

research into vulnerability of cryptographic hardware is just as important. Although not many results had been published yet, it is prudent to suggest that cryptographic hardware also leaks side channel information, and that alongside with general tamper-resistant features, cryptographic coprocessors should include countermeasures specifically targeted to protect them against side channel attacks.

One of the most powerful techniques to counteract side channel attacks is to *mask* all input and intermediate data with some random values in order to de-correlate any information leaked through the side channels from actual secret data being processed [5]. The idea is simple: the message and the key are masked with some random masks at the beginning of computations, and thereafter everything is almost as usual. Of course, the value of the mask at the end of some fixed step must be known in order to re-establish the expected value at the end of the execution; we call this *mask correction*.

In this paper we propose an architecture for an AES [7] coprocessor that is immune to simple and first order differential power analysis attacks. The main idea is to implement directly in hardware a novel method of computing all field operations directly on masked data. A price to pay for such built-in security is increased gate counts and power consumption. The synthesis results for an architecture with data masking countermeasure implementing a 16 clock/round version of the AES with a flexible key size, show that with 0.18 μm technology the performance of 4Mbps can be achieved with the circuit comprising 20,506 gates clocked at 5MHz and requiring 1.7 μA. This is better than countermeasures such as dynamic and differential CMOS logic [29] or asynchronous circuits with dual rail logic [23] can offer. Also, our solution has an advantage of using standard technologies, standard gate libraries and well-established design tools.

The general design flow for a secure AES module is depicted in Fig. 1. First, a Verilog model had been created and tested, after which Cadence Design System Verilog-XL simulator was used to generate timing diagrams and to verify the correctness of the design. When RTL code had been verified, the digital circuit was synthesized with Synopsys Design Analyzer tool using Samsung 0.18 μ libraries. Power compiler simulation data at 5MHz were obtained with the in-house simulation tool CubicWare.

The rest of the paper is organized as follows. After a brief description of the AES algorithm and basic hardware architecture, we outline in Chapter 3 how to reduce inversion in the field $GF(2^8)$ to inversion in the composite field $GF((2^4)^2)$, and how the latter can be realized in combinational logic only. Chapter 4 discusses a general countermeasure against side channel attacks comprising masking all manipulated data with random values, and points out the difficulties of implementing inversion on masked data. We suggest our solution to this problem by showing how computations on masked data and corresponding mask corrections can be carried out at a gate level. The resulting secure AES hardware architecture is described in details in Chapter 5, where we also estimate the cost in terms of the gate count and power consumption. The paper is concluded with simulation results and an outline of a future work.

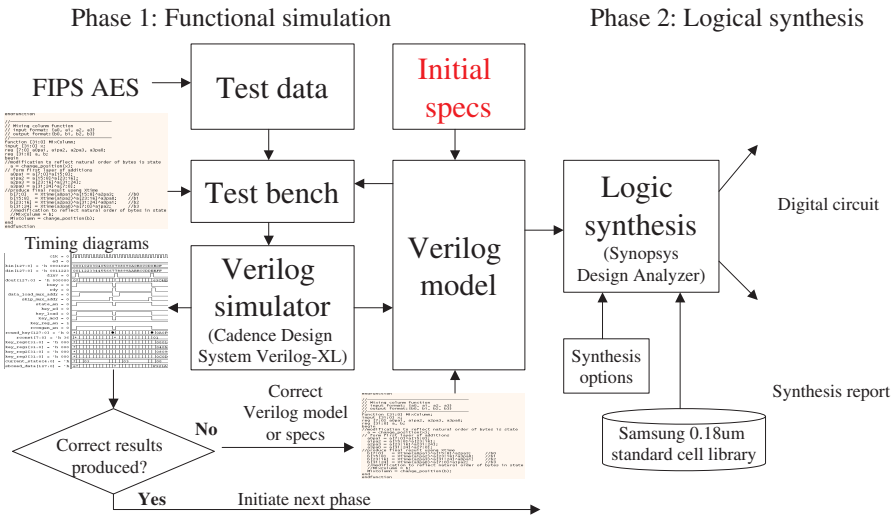

Fig. 1. AES hardware design flow

2 AES Reminder

The Advanced Encryption Standard [7] is a round-based symmetric block cipher. The round consists of four different operations namely, *SubBytes*, *ShiftRows*, *MixColumn*, and *AddRoundKey* that are performed repeatedly in a certain sequence; each operation maps a 128-bit input state into a 128-bit output state. The state is represented as 4×4 matrix of bytes. The number of rounds, depending on the key size, is 10, 12 or 14, with the MixColumn operation being omitted in the last round. Prior to the main loop, AddRoundKey is executed for initialization. The standard key size is 128 bits; but for some applications 192 and 256-bit keys must be supported as well. In the decryption process, the inverse operations of each basic function are executed in a slightly different order.

ShiftRows is a cyclic shift operation on each of four row in a 4×4-byte state using $0 \sim 3$ offsets. *MixColumn* treats 4-byte data blocks in each column of a state as coefficients of a 4-term polynomial, and multiplies them modulo $x^4 + 1$ with the fixed polynomial $c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. *AddRoundKey* is a bit-wise XOR operation on 128-bit round keys and the data. These three operations are *linear*.

SubBytes is the main building block of the AES algorithm. It replaces each byte in a state by its substitute in an *Sbox* that comprises a composition of two transformations:

– First, each byte in a state is replaced with its reciprocal in the finite field $GF(2^8)$, except that 0, which has no reciprocal, is replaced by itself. This is the only *non-linear* function in the AES algorithm.

– Then an affine transformation f is applied. It consists of a bitwise matrix multiply with a fixed 8×8 binary matrix followed by XOR with the hexadecimal number {63}.

The *round key* is computed in parallel to the round operation. It is derived from the cipher key by means of key expansion and round key selection operations, which are similar to those of the round operation and use *Sbox*-es.

Fig. 2 shows an implementation of a fully "unfolded" 128 key bits AES algorithm that executes one round of encryption/decryption per cycle.

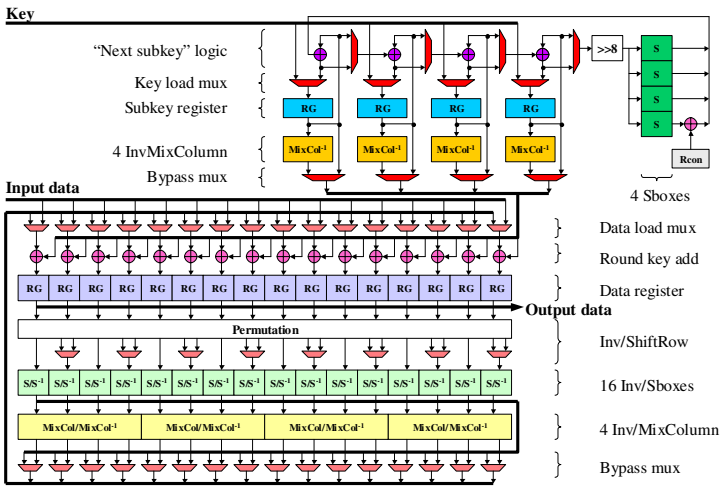

Fig. 2. Structure of the datapath and the key scheduler for a fully unfolded AES architecture

A sequence of round operations is implemented as a combinational circuit and its input and output are connected to 128-bit data registers. The multiplexors are used to skip some operations in the first and the last rounds and to choose between encryption or decryption modes. While *AddRoundKey* and *ShiftRows* have rather trivial hardware realization, there are many possible choices how to optimize silicon area for the other two operations. Important design parameters are suitability for both, encryption and decryption, and the number of *Sbox* and *MixColumn* blocks used in the design. An efficient implementation of these two blocks plays a major role in reducing area [16, 18] and power consumption [22].

3 Sbox Architecture

There are many design trade-offs to be considered when implementing an *Sbox* in ASIC since the size, the speed, and the power consumption of an AES coprocessor depend largely on the number and the style of implementation of

Sbox-es [22]. It also turned out that this operation is the most difficult to protect against side channel attacks.

A number of flexible ASIC solutions that use similarities between encryption and decryption to share silicon were proposed [16, 22], where the *SubBytes* operation was implemented in two steps, as a combination of inversion in the field and an affine transformation. While the affine transformations used for encryption and decryption are slightly different, the silicon implementing inversion in $GF(2^8)$ can be used for both, as shown in Fig. 3. Therefore, an area- and power-efficient and secure implementation of inversion may have a big impact on an overall design.

The most obvious solution is to use a look-up table for this operation [16]. It is fast and inexpensive in terms of power consumption [22]. There is a major drawback, however. Namely, the size of silicon, which is about 1,700 gate equivalents per table in 0.18μm technology. Considering that up to 20 such tables (including 4 tables for key scheduling) may be required, this solution is hardly feasible for co-processors intended for smart cards and other embedded systems.

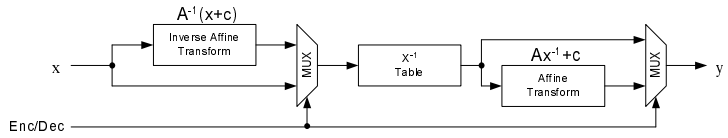

Fig. 3. S-box implementation suitable for encryption and decryption

Among various alternative approaches, the one based on inversion in the composite field produces the most compact AES implementations [26, 27, 32] that can also be optimised with respect to power consumption [22]. As a basis for our design, we use fully combinational logic implementation of inversion in composite fields described in [32].

3.1 Composite Fields

Usually, the field $GF(2^8)$ is seen as an extension of $GF(2)$ and therefore its elements can be represented as bytes. However, $GF(2^8)$ can also be seen as a quadratic extension of $GF(2^4)$; in this case an element $a \in GF(2^8)$ is represented as a linear polynomial $a_H x + a_L$ with coefficients in $GF(2^4)$. We denote it $[a_H, a_L]$. This isomorphic representation had been found to be far better suited for hardware implementation [24, 26, 22, 32].

The bijection from $a = (a_0, ..., a_7)$ to a two-term polynomial $[a_H = (a_{h0}, ..., a_{h3}), a_L = (a_{l0}, ..., a_{l3})]$ is given by a linear function *map* computed by means of the XOR operation on bits of a (see [32]):

$a_L = (a_{l0} = a_C \oplus a_0 \oplus a_5,\ a_{l1} = a_1 \oplus a_2,\ a_{l2} = a_A,\ a_{l3} = a_2 \oplus a_4)$, and
$a_H = (a_{h0} = a_C \oplus a_5,\ a_{h1} = a_A \oplus a_C,\ a_{h2} = a_B \oplus a_2 \oplus a_3,\ a_{h3} = a_B)$,

where $a_A = a_1 \oplus a_7, a_B = a_5 \oplus a_7, a_C = a_4 \oplus a_6$. The inverse transformation map^{-1} converts a two-term polynomial back to element $a \in GF(2^8)$, and is defined in a similar way. For more details see [32].

All arithmetic operations applied to elements in $GF(2^8)$ can also be computed in the new representation. Two-term polynomials are added by addition of corresponding coefficients: $(a_H x + a_L) \oplus (b_H x + b_L) = (a_H \oplus b_H)x + (a_L \oplus b_L)$.

Multiplication and inversion of two term-polynomials require modular reduction to ensure that the result is a two-term polynomial as well. For this purpose, one can use irreducible polynomial $n(x) = x^2 + \{1\}x + \{e\}$ whose coefficients have been chosen to optimize finite field computations.

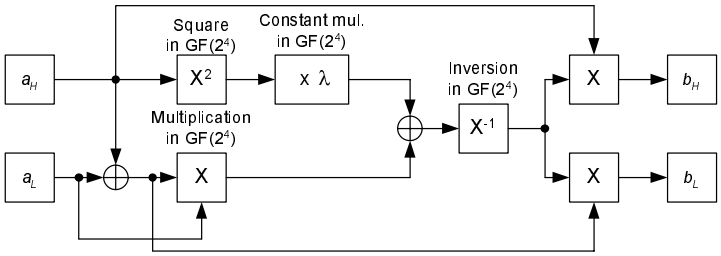

Fig. 4. Inversion in $GF((2^4)^2)$

Inversion of a two-term polynomial is defined as $(a_H x + a_L) \otimes (a_H x + a_L)^{-1} = \{0\}c + \{1\}$. From this definition the formulae for inversion can be derived:

$$(a_H x + a_L)^{-1} = (a_H \otimes d)x + (a_H \oplus a_L) \otimes d, \quad where$$
$$d = ((a_H^2 \otimes \{e\}) \oplus (a_H \otimes a_L) \oplus a_L^2)^{-1}.$$

Fig. 4 depicts a block diagram of inversion in the field $GF((2^4)^2)$. As one can see, one addition, one squaring, one constant multiplication, three general multiplications and one inversion in $GF(2^4)$ are necessary for its implementation. These operations can be realized in combinational logic as described in [32], and reminded below.

3.2 Arithmetic Operations in $GF(2^4)$

Addition in $GF(2^n)$ is a simple bitwise XOR. Multiplication and inversion in $GF(2^4)$ require an irreducible polynomial of degree four, e.g., $m(x) = x^4 + x + 1$.

Multiplication $q(x) = a(x) \otimes b(x) = a(x) \cdot b(x) \mod m(x)$ can be realized in combinational logic as follows:

$$\begin{aligned} q_0 &= a_0 b_0 \oplus a_3 b_1 \oplus (a_2 \oplus a_3) b_2 \oplus (a_1 \oplus a_2) b_3 \\ q_1 &= a_1 b_0 \oplus (a_0 \oplus a_3) b_1 \oplus a_2 b_2 \oplus a_1 b_3 \\ q_2 &= a_2 b_0 \oplus a_1 b_1 \oplus (a_0 \oplus a_3) b_2 \oplus (a_2 \oplus a_3) b_3 \\ q_3 &= a_3 b_0 \oplus a_2 b_1 \oplus a_1 b_2 \oplus (a_0 \oplus a_3) b_3. \end{aligned} \tag{1}$$

Here concatenation of two bits $a_i b_j$ represents binary multiplication, i.e., logical AND.

Squaring $q(x) = a(x)^2 \mod m(x)$ is a special case of multiplication, and can be computed with XOR operations only: $q_0 = a_0 \oplus a_2$, $q_1 = a_2$, $q_2 = a_1 \oplus a_3$ and $q_3 = a_3$.

An inverse $q(x) = a(x)^{-1} \mod m(x)$ of an element $a \in GF(2^4)$ is derived by solving the equation $a \otimes a^{-1} \mod m(x) = 1$, and is implemented in combinational logic as described below:

$$\begin{aligned} q_0 &= a_A \oplus a_0 \oplus a_0a_2 \oplus a_1a_2 \oplus a_0a_1a_2 \\ q_1 &= a_1(a_0 \oplus a_2 \oplus a_3 \oplus a_0a_3) \oplus a_0a_2 \oplus a_3 \\ q_2 &= a_0(a_1 \oplus a_2 \oplus a_3 \oplus a_2a_3) \oplus a_2 \oplus a_3 \oplus a_0 \\ q_3 &= a_A \oplus a_0a_3 \oplus a_1a_3 \oplus a_2a_3, \end{aligned} \tag{2}$$

where $a_A = a_1 \oplus a_2 \oplus a_3 \oplus a_1a_2a_3$.

As one can see, all arithmetic operations in the extension field are eventually reduced to the bit-wise logical XOR and AND functions.

4 Side Channel Attacks and Computations on Masked Data

Basically, side-channel attacks work because there is a correlation between the physical measurements taken during computations, such as power consumption [12, 20], EMF radiation [9, 25], time of computations [13], and the internal state of the processing device, which itself is related to a secret key.

Among attacks, the Differential Power Analysis (DPA) is the most dangerous [20]. It uses statistical analysis to extract information from a collection of power consumption curves obtained by running an algorithm many times with different inputs. Then an analysis of a probability distribution of certain points on the curves is carried on. It uses correlations between power consumption and specific key-dependent bits which appear at known steps of cryptographic computations.

For example, a selected bit b at the output of one *Sbox* of the first round of the AES algorithm will depend on the known input message and 8 unknown bits of the key. The correlation between power consumption and b can be computed for all 256 values of 8 unknown bits of the key. The correlation is likely to be maximal for the correct guess of the 8 bits of the key. Then an attack can be repeated for the remaining S-boxes.

There are many strategies to combat side-channel attacks. On the hardware level, the countermeasures usually include clock randomization [28, 14], power consumption randomization or compensation [8], randomisation of instruction set execution and/or register usage [17]. However, the effect of these countermeasures can be reduced by various signal processing techniques [6]. Software-based countermeasures include introducing dummy instructions, randomization of the instruction execution sequence, balancing Hamming weights of the internal data, and bit splitting. When each data bit is split into two shares, we get what is called *data masking*.

Data masking is one of the most powerful software countermeasures against side channel attacks [5, 12]. The idea is simple: the message and the key are

masked with some random masks at the beginning of computations, and thereafter everything is almost as usual. Of course, the value of the mask at the end of some fixed step (e.g., at the end of the round or at the end of a linear part of computations) must be known in order to re-establish the expected data value at the end of the execution; we call this *mask correction*.

A traditional XOR operation is used for data masking; however, a mask is arithmetic in $GF(2^8)$. The operation is compatible with the AES structure except for inversion in *SubBytes*, which is the only non-linear transformation. In other words, to compute mask corrections for all linear transformations in a round, we simply have to apply these transformations separately to masked bytes and to masks. Numerous efforts to find an efficient secure implementation of the *Sbox* on masked data had only a limited success.

The first attempt to transform masked data between Boolean and arithmetic operations in a secure way [19] was shown to be insufficient against DPA attacks; a more sound method for mask switching [11] involves too much computational overhead.

To overcome this difficulty, Akkar and Giraud [1] proposed so-called transformed masking, where first an additive mask is replaced by a multiplicative mask in a series of multiply and add operations in the field, after which usual inversion takes place, and finally, the transformation of multiplicative mask into additive mask is carried out again as a series of multiply and add operations. However, it was pointed out in [10, 30] that a multiplicative mask does not blind zero, which becomes an Achilles hill of the implementation.

In what follows we show how to overcome this problem by carrying on computations on masked data at the gate level.

4.1 XOR and AND on Masked Data

As one have seen, all arithmetic operations in the extension field are eventually reduced to bit-wide XOR and AND operations. Since XOR is a linear function, one can compute it directly on masked data.

Hence, the problem of "masked inversion" can be effectively reduced to computing binary AND on masked bits and on bits of the mask without ever revealing actual data bits in the process.

Our solution is based on algebraic properties of logic operations. We derive the solution by manipulating algebraic formulae. Denote masked bits via $\tilde{a} = (a \oplus r_a)$ and $\tilde{b} = (b \oplus r_b)$. Then

$$\tilde{a} \cdot \tilde{b} = (a \oplus r_a) \cdot (b \oplus r_b) = (a \cdot b) \oplus (a \cdot r_b) \oplus (r_a \cdot b) \oplus (r_a \cdot r_b). \tag{3}$$

In order to compute the mask correction $(a \cdot r_b) \oplus (r_a \cdot b) \oplus (r_a \cdot r_b)$ in a secure way, we want to express the terms $(a \oplus r_b)$ and $(b \oplus r_a)$ using only masked data $\tilde{a}$, $\tilde{b}$, and the bits of the mask r_a and r_b. From the equation $r_a \cdot \tilde{b} = (r_a \cdot b) \oplus (r_a \cdot r_b)$ we derive the method to compute $r_a \cdot b$ securely:

$$(r_a \cdot b) = r_a \cdot \tilde{b} \oplus (r_a \cdot r_b). \tag{4}$$

Similarly, from equation $r_b \cdot \tilde{a} = (r_b \cdot a) \oplus (r_b \cdot r_a)$ we compute:

$$(a \cdot r_b) = r_b \cdot \tilde{a} \oplus (r_a \cdot r_b). \tag{5}$$

Substituting corresponding terms in equation 3 for their values obtained in 4 and 5, and simplifying the result, we have:

$$(a \cdot b) = \tilde{a} \cdot \tilde{b} \oplus (r_a \cdot \tilde{b}) \oplus (r_b \cdot \tilde{a}) \oplus (r_a \cdot r_b. \tag{6}$$

Hence, the computations of the "mask correction" can be carried out without compromising the bits of actual data.

The question is if recycling previously used mask bits is good enough to make the whole construction robust. In other words, we have to prove that the joint random variable $(\tilde{a}, \tilde{b})$ is balanced and independent of the joint random variable (a, b) and all intermediate terms in the equation 6 are balanced and independent from a, b, $a \cdot b$, and $a \oplus b$.

However, this analysis is made redundant by the fact that all modern secure devices such as smart cards must have random number generator on board. Such generators are implemented in hardware and have a high output rate. Thus, generating random bits on-line is possible.

To achieve balanced and independent intermediate results, we use a freshly generated random bit z as a new mask, computing masked AND as follows:

$$a \cdot b \oplus z = ((((\tilde{a} \cdot \tilde{b}) \oplus (r_a \cdot \tilde{b})) \oplus ((r_b \cdot \tilde{a}) \oplus z)) \oplus (r_a \cdot r_b)).$$

An actual implementation of the masked AND operation is realized as a cascade of layers of logic gates. The first layer generates the necessary elementary AND-terms. All other layers produce variables in the form $(\alpha \oplus z)$, where z is independent on all the bits that have been used so far. Alternative constructions can be used as well stemming from the basic algebraic formulae relating the operations AND, OR and XOR in $GF(2)$.

4.2 Masked Inversion in $GF(2^4)$.

The final architecture for a masked inverter in $GF((2^4)^2)$ is depicted in Fig. 5. Here SMul and X^{-1} boxes represent blocks of combinational logic implementing a multiplier 1 and an inverter 2, where each "$\oplus$" is replaced with masked XOR, and each "$\cdot$" is replaced with masked AND, as discussed previously. M is a random mask, *(P xor M)* represents a masked input, and *Z, W, F* are new masks used to "refresh" the masked data at the end of each operation in $GF(2^4)$ and at the end of inversion in $GF((2^4)^2)$.

Notice, that the inverse field isomorphism map^{-1} and the affine transformation f are both linear operations, and they can be merged to optimize the gate count in the *Sbox*; the same holds for *map* and f^{-1} in decryption [16, 22]. This is how we implemented it. The final structure of the masked *Sbox* is depicted in Fig. 6. The duplicate datapath is used to compute the mask correction. Altogether, 1.2K gate equivalents is required to implement one *Sbox*, which is 25% better than the table lookup implementation.

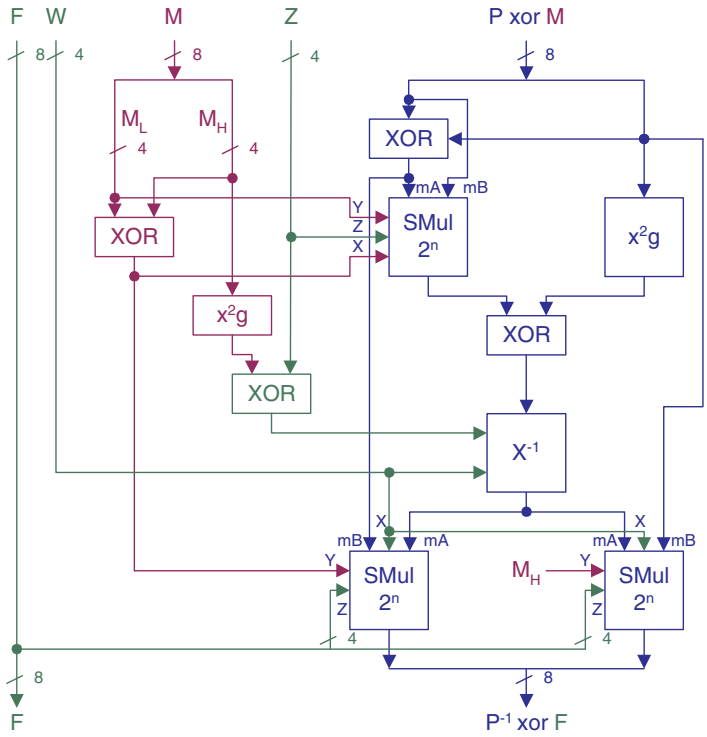

Fig. 5. Masked inversion in $GF(2^4)$

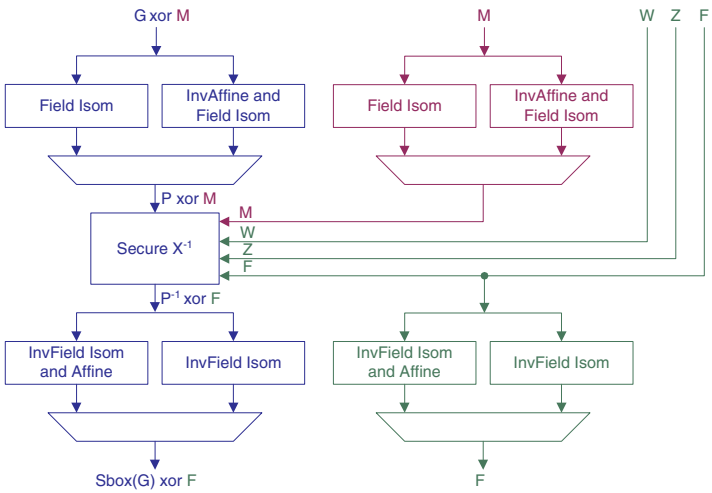

Fig. 6. Masked S-box

5 Secure AES Coprocessor

When manipulating masked data, all stages in a round, apart from inversion, require simple mask corrections in a form of analogous computations on masks that are carried out in parallel with the main computation flow. This can be achieved by duplicating hardware. The final architecture for a secure AES coprocessor is depicted in Fig. 7.

Another solution is to pipeline computations on masked data and on masks, which halves the throughput, and for which we need additional 128-bit registers. We had chosen the first approach because it is simpler conceptually, and it does not use more silicon than a pipelined architecture.

The hardware module based on the described scheme has been fully implemented in $0.18\mu m$ technology. The table in Fig. 8 gives the detailed gate count for a "minimalist" architecture where only one *Sbox* had been used for the main datapath. As one can see, a masked *Sbox*, where computations on bits of the masks are interleaved with computations on masked data, consumes relatively small amount of gates in comparison with the total amount of gates required for a whole datapath duplication. Thus, as a balance between the security, the throughput, and the gate count, four *Sbox*-es had been used for the main datapath in a real-life implementation,

The summary of the synthesis results for an original, i.e., unmasked, AES coprocessor and its masked counterparts is given in the table below. The last

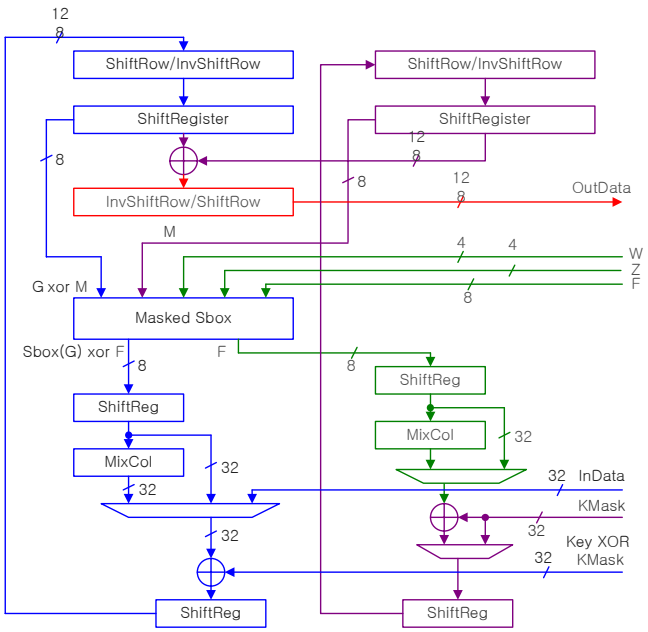

Fig. 7. Datapath for secure AES

Components	Subcomponents	Scalable key size	128 bit key size
Masked Sbox	Inverter in GF($(2^4)^2$)	1206	1206
	Input-output processing	350 (180)	350 (180)
Subtotal for masked Sbox (for key scheduler)		**1556 (1386)**	**1556 (1386)**
Masked datapath	Masked data	4900	4900
	Mask	2732	2732
Subtotal for masked datapath		**7632**	**7632**
Masked key scheduler	Masked key	6523	4835
	Mask	4913	3225
Subtotal for masked key scheduler		**11436**	**8060**
Control	Data control	300	300
	Key control	1140	400
TOTAL for masked AES		**20506**	**16390**

Fig. 8. Gate count for secure AES module

column gives the comparison with the minimalist architecture with only one *Sbox* in a datapath.

Parameter	Unmasked 4 Sboxes	Masked 4 Sboxes	Minimalist Masked 1 Sbox
Clk/Round	4	4	16
Gate count (128 bit key)	11.8K	21.7K	16.39K
Performance at 5MHz	16 Mbps	16 Mbps	4Mbps
Maximal performance	80 Mbps	74 Mbps	–
Power consumption	1.1 mA	2.1 mA	1.6 mA
Critical path	40 ns	43.18 ns	60 ns

The total increase in the gate count for a secure AES module with a fixed key size is 183%. However, it is a power consumption that is a real bottleneck in applications such as mobile communication. For a modern SIM-card with an AES coprocessor on board to be compliant with the GSM standard, the power consumption of the coprocessor itself must be around 1 mA. The power consumption even for a minimalist masked architecture is 1.6 mA in 0.18 μm technolog. However, with 0.13 μm technology the power consumption can be reduced to 1.07 mA, which allows us to use this module in applications such as GSM and ad-hoc networks.

6 Conclusion

We propose a new hardware implementation of the AES module secure against first order DPA attacks. Namely, we designed a combinational logic block to compute inversion directly on masked values without ever revealing actualdata in a process. Moreover, our approach solves the troubling problem of a zero attack.

The described approach can be used in other cryptographic coprocessors that use non-linear operations. It provides comparable protection as dynamic and

differential logic [29] and asynchronous dual rail circuits [23] at the similar (or slightly less) price in terms of the gate count and power consumption. Taking into account that the latter techniques require new logic libraries, careful "balancing" of place and routing, new development tools and longer time-to-market, our design offers a competitive alternative to hardware protection.

There are many interesting research and practical issues that remain unanswered. First of all, it would be good to understand exactly the amount of randomness that is actually needed to provide total protection against various DPA attacks. The first step in this direction was made in [3].

Secondly, while the data masking technique protects well against first-order DPA attacks, where an attacker observes correlations on only one intermediate value, it still may be vulnerable to higher-order attacks where up to N points on the power consumption curves (or, in other words, N intermediate values) can be observed at the same time [19]. In theory, a relevant countermeasure in this case should split each data bit into N shares, and carry on independent computations on each of the shares. Practically, it would amount to N-replication of hardware if we apply a similar technique.

Of course, in designing efficient and secure coprocessors for smart cards, other considerations, notably, production costs, have to be considered as well. Often it is not the best feature that counts, but overall design trade-offs. Investigation of such trade-offs is a challenging experimental problem. It is already known, for example, that clock randomization alone does not offer a sufficient protection even against first-order DPA because its effect can be relatively easy eliminated by clever processing of power curves using signal processing techniques [6]. However, a combination of clock randomization and data masking may make it very difficult to mount a higher-order DPA because an attacker would have to realighn simultaneously N points on power curves.

As a future work, we want to compare modules that work on masked data with other ASIC implementations of the AES algorithm for smart cards, where resistance to attacks is ensured by general-purpose countermeasures, such as variable clocks [28, 14], random current generators, balancing/duplicate logic [8, 4], random wait states, and their combinations [6].

References

1. Akkar, M., Giraud, C.: An implementation of DES and AES, secure against some attacks. Proc. Cryptographic Hardware and Embedded Systems: CHES 2001. Lecture Notes in Computer Science **2162** (2001) 309-318
2. Anderson, R., Kuhn, M.: Low cost attacks on tamper resistant devices. Proc.Security Protocols: IWSP 1997. Lecture Notes in Computer Science **1361** (1997) 125-136
3. Blmmer, J., Merchan J. G., Krummel, V.: Provably secure masking of AES. IACR Cryptology ePrint Archive Report 2004/101 (2004)
4. M. Bucci, L.Germani, M. Guglielmo, R. Luzzi, A. Trifiletti: A simulation methodology for DPA resistance testing of cryptographic processors (manuscript) 2003

5. Chari, S., Jutla, C., Rao, J., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. Proc. Advances in Cryptology – Crypto'99. Lecture Notes in Computer Science **1666** (1999) 398-412,
6. Clavier, C., Coron, J-S., Dabbous, N.: Differential power analysis in the presence of hardware countermeasures. Proc. Cryptographic Hardware and Embedded Systems: CHES 2000. Lecture Notes in Computer Science **1965** (2000) 252-263
7. Daemen, J., Rijmen, V.: *The design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag Berlin Heidelberg (2002)
8. Fruhauf, S., Sourge, L.: *Safety device against the unauthorized detection of protected data*. U.S. patent 5,404,402 (1995)
9. Gandolfi, K., Mourtel, C., Oliver, F.: Electromagnetic analysis: concrete results. Proc. Cryptographic Hardware and Embedded Systems: CHES 2001. Lecture Notes in Computer Science **2162** (2001) 251-261
10. Goliç, J., Tymen,Ch.: Multiplicative masking and power analysis of AES. Proc. Cryptographic Hardware and Embedded Systems: CHES 2002. Lecture Notes in Computer Science **2523** 198-212
11. Goubin, L.: A sound method for switching between boolean and arithmetic masking. Proc. Cryptographic Hardware and Embedded Systems: CHES'01. Lecture Notes in Computer Science **2162** (2001) 3-15
12. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. Proc. Advances in Cryptology – CRYPTO'99. K Lecture Notes in Computer Science **1666** (1999) 388-397
13. Kocher, P.: Timing attacks on implementations of Diffie-Hellmann, RSA, DSS, and other systems. Proc. Advances in Cryptology – Crypto'96. Lecture Notes in Computer Science **1109** (1996) 104-113
14. Kocher, P., Jaffe J., Jun, B.: *Using unpredictable information to minimize leakage from smartcards and other cryptosystems*, USA patent, International Publication number WO 99/63696 (1999)
15. Kommerling, O., Kuhn, M.: Design principles for tamper-resistant smartcard processors. Proc. USENIX Workshop on Smartcard Technology (Smartcard 99) (1998) 9-20
16. Lu, C. C., Tseng, S-Y.: Integrated design of AES (Advanced Encryption Srandard) encryptor and decryptor. Proc. IEEE conf. on Application-Specific Systems, Architectures, and Processors (ASAP'02) (2002) 277-285
17. May, D., Muller, H. L.,Smart, N. P.: Random register renaming to foil DPA. Proc. Cryptographic Hardware and Embedded Systems – CHES'01. Lecture Notes in Computer Science **2162** (2001)
18. Mangard, S., Aigner, M., Dominikus, S.: A highly regular and scalable AES hardware architecture. IEEE Transactions on Computers **52** no. 4 (2003) 483-491
19. Messerges, T.: Securing the AES finalists against power analysis attacks. Proc. Fast Software Encryption Workshop 2000. Lecture Notes in Computer Science **1978** (2000) 150-165
20. Messerges, T. S., Dabbish, E. A., Sloan, R. H.: Examining smart-card security under the thread of power analysis. IEEE Trans. Computers. **51** no. 5 (2002) 541-522
21. Messerges, T. S.: Using second-order power analysis to attack DPA resistant software. Proc. Cryptographic Hardware and Embedded Systems – CHES 2000. Lecture Notes in Computer Science **1965** (2000) 238-251
22. Morioka, S., Satoh, A.: An optimized S-Box circuit architecture for low power AES design. Proc. Cryptographic Hardware and Embedded Systems: CHES 2002. Lecture Notes in Computer Science **2523** (2003) 272-186

23. Moore, S., Anderson, R., Cunningham, P., Mullins, R., Taylor, G.,: Improving smart card security using self-timed circuits. Proc. Proceeding 8th IEEE International Symposium on Asynchronous Circuits and Systems – ASYNC'02. IEEE (2002) 23-58
24. Paar, C.: *Efficient VLSI architectures for bit parallel computations in Galois fields*. PhD Thesis, University of Essen, Germany (1994)
25. Quisquater, J. J., Samide, D.: Electromagnetic analysis (ema): measures and counter-measures for smart cards. Proc. Smartcard Programming and Security. Lecture Notes in Computer Science **2140** (2001) 200-210
26. Rudra, A., Dubey, P., Julta, C., Kumar, V., Rao, J., Rohatgi, P.: Efficient Rijndael implementation with composite field arithmetic. Proc. Cryptographic Hardware and Embedded Systems – CHES'01. Lecture Notes in Computer Science **2162** (2001) 175-188
27. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A compact Rijndael hardware architecture with S-Box optimization. Proc. Advances in Cryptology – ASIACRYPT 2001. Lecture Notes in Computer Science **2248** (2001) 239-254
28. E. Sprunk, *Clock frequency modulation for secure microprocessors*, USA patent number WO 99/63696 (1999)
29. Tiri, K., Akmal, M., Verbauwhede, I.: A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. Proc. IEEE 28th Europen Solid-State Circuit Conf. – ESSCIRC'02 (2002)
30. E. Trichina, E., De Seta, D., Germani, L.: Simplified Adaptive Multiplicative Masking for AES and its secure implementation. Proc. Cryptographic Hardware and Embedded Systems: CHES 2002. 2523 of Lecture Notes in Computer Science **2523** (2002) 277-285
31. Wolkerstorfer, J.,: An ASIC implementation of the AES MixColumn operation, In Proceedings Austrochip 2001 (2001)
32. Wolkerstorfer, J., Oswald, E., Lamberger, M.: An ASIC implementation of the AES S-Boxes. Proc. Topic in Cryptography – CT-RSA 2002. 2271 of Lecture Notes in Computer Science **2271** (2002) 67-78