Secure Implementation of the Stern Authentication and Signature Schemes for Low-Resource Devices

Pierre-Louis Cayrel[1], Philippe Gaborit[1] and Emmanuel Prouff[2]

[1] Université de Limoges, XLIM-DMI,
123, Av. Albert Thomas 87060 Limoges Cedex France
{pierre-louis.cayrel,philippe.gaborit}@xlim.fr
[2] Oberthur Technologies
71-73, rue des hautes pâtures 92726 Nanterre Cedex France
e.prouff@oberthurcs.com

Abstract. In this paper we describe the first implementation on smart-card of the code-based authentication protocol proposed by Stern at Crypto'93 and we give a securization of the scheme against side channel attacks. On the whole, this provides a secure implementation of a very practical authentication (and possibly signature) scheme which is mostly attractive for light-weight cryptography.

1 Introduction

While Cryptography aims at preventing persons from cheating, Coding Theory has been originally designed to prevent accidental errors coming from the imperfections of the transmission systems (*e.g.* phone lines, microwaves, satellite communications, CDs, etc.). Nowadays, it studies more generally how to protect information transiting over unperfect channels from alterations. The core idea is to send over the channel more data than the initial amount of information to convey. The added information, usually called *redundancy*, is structured in such a way that it is possible to detect and (eventually) to correct almost all the errors that could occur during the data transmission.

After a first scheme proposed by McEliece in 1978 using error-correcting codes for encryption, the idea of using error-correcting codes for authentication purposes was due to Harari, followed by Stern (first protocol) and Girault. The protocols of Harari and Girault were subsequently broken, while Stern's one was five-pass and unpractical. Eventually, the first practical and secure protocol based on error-correcting codes was proposed by Stern at Crypto'93 [15]. This zero-knowledge authentication protocol is based on an error-correcting codes problem usually referred as the *Syndrome Decoding* (*SD* in short) Problem. Stern's protocol is a Fiat-Shamir-like protocol but with a cheating probability of 2/3 rather than 1/2 for Fiat-Shamir. It is hence considered as a good alternative to the numerous authentication schemes whose security relies on classical number theory problems such as the factorization or the discrete logarithm problems.

Although the Stern Scheme was proposed almost 15 years ago, it has (as far as we know) never been implemented on smart card until now. This is merely due to the usual drawback of code-based systems: the size of the public data is large. Indeed, since the prover and the verifier have to know a large random matrix with at least 100-kbits, it is hard to use the scheme on devices with low resources such as smart cards or RFID tags. This drawback has been recently solved by Gaborit and Girault in [2] where they propose to use the parity check matrix of a random quasi-cyclic code rather than a pure random matrix. This solution permits to preserve the security of the scheme and decreases the description of the random matrix to only a few hundred bits. This new advance opens the door to the use of the Stern protocol in devices with low resources.

Contribution In this paper we give for the first time a precise description of the implementation of Stern's Protocol (which is very different from the classical number-theory based protocols) and we show how to protect the main steps of the algorithm against side channels attacks. Eventually, we obtain for our implementation an authentication in 6 seconds and a signature in 24 seconds, both without any crypto-processor. This is a promising result when compared to an RSA implementation which would take more than 30 seconds in a similar context without crypto-processor. Stern's Protocol may have a natural application in contexts where the time constraints are not tight such as: authentication for pay-TV or authentication for counterfeiting of expensive goods (*e.g.* ink cartridges of copy machines or expensive clothes). Besides this, the protocol has also the 4 following advantages :
1) it can be an alternative to the number-theory based protocols in case a putative quantum computer may exist;
2) since it essentially involves linear operations, the protocol seems easier to protect against side channel attacks than the number-theory based protocols;
3) the linear operations (scalar products or bit-permutations) are easy to implement in hardware and are very efficient in this context;
4) the secret key is smaller than the one of the other protocols (a few hundred bits) for the same security level.

Organisation of the paper The paper is organized as follows. In Section 2, we describe Stern's Authentication and Signature schemes and we precise the four main steps of the implementation. In Section 3, we present the side channel attacks in the coding theory context and in Section 4 we propose a secure version of our implementation against side channel attacks. In Section 5, we comment the implementation and eventually we conclude.

2 Stern Authentication Scheme

2.1 Basic Scheme

Stern's Scheme (see [15] for more details) is an interactive zero-knowledge protocol which aims at enabling any user (usually called *the prover* P) to identify himself to another one (usually called *the verifier* V). Let n and k be two integers such that $n \geq k$. Stern's Scheme assumes the existence of a public $(n-k) \times n$

matrix $\widetilde{H}$ defined over the field $\mathbb{F}_2$ and the choice of an integer $t \leq n$. The matrix $\widetilde{H}$ and the weight t are protocol parameters and may be used by several (even numerous) different provers.

Each prover P receives a n-bit secret key s_P (also denoted by s if there is no ambiguity about the prover) of Hamming weight t and computes a *public identifier* i_V such that $i_V = \widetilde{H} s_P^T$. This identifier is calculated once in the lifetime of $\widetilde{H}$ and can thus be used for several authentications. A user P can prove to V that he is the person associated to the public identifier i_V, by performing the following protocol, (for h a standard hash function):

1. [Commitment Step] P randomly chooses $y \in \mathbb{F}^n$ and a permutation σ defined over $\mathbb{F}_2^n$. Then P sends to V the commitments c_1, c_2 and c_3 such that :

$$c_1 = h(\sigma | \widetilde{H} y^T);\ c_2 = h(\sigma(y));\ c_3 = h(\sigma(y \oplus s)),$$

   where $h(a|b)$ denotes the hash of the concatenation of the sequences a and b.
2. [Challenge Step] V sends $b \in \{0, 1, 2\}$ to P.
3. [Answer Step] Three possibilities:
   - if $b = 0 : P$ reveals y and σ.
   - if $b = 1 : P$ reveals $(y \oplus s)$ and σ.
   - if $b = 2 : P$ reveals $\sigma(y)$ and $\sigma(s)$.
4. [Verification Step] Three possibilities:
   - if $b = 0 : V$ verifies that c_1, c_2 have been honestly calculated.
   - if $b = 1 : V$ verifies that c_1, c_3 have been honestly calculated.
   - if $b = 2 : V$ verifies that c_2, c_3 have been honestly calculated, and that the weight of $\sigma(s)$ is t.
5. Iterate the steps 1,2,3,4 until the expected security level is reached.

Fig. 1. Stern's Protocol

Based on the difficulty of the SD problem, it is proven that the protocol is zero-knowledge with a probability of cheating of (2/3) for one round. An appropriate confidence level is reached by repetition of the protocol.

Remark 1. By using the so-called Fiat-Shamir Paradigm [1], it is theoretically possible to convert Stern's Protocol into a signature scheme, but then the signature is very long: about 140-kbit long for 2^{80} security.

Despite the advantages of the protocol (it can be an alternative to number theory based protocol, it is fast and it uses simple linear operations), Stern's Scheme has rarely been used since its publication in 1993. Indeed, the scheme presents the two following drawbacks, which together makes it unpracticable in many applications: 1) many rounds are required (typically 28 if we want the cheater success probability to be less than 2^{-16}), 2) the public key element $\widetilde{H}$ is very large (typically 150-kbit long).

The first point is inherent to interactive protocols and in some situations, it does not really constitute a drawback. For instance, if the prover and the

verifier entities can be connected during a long period, then authentication can be achieved gradually. In this case the entire authentication process is performed by executing, time to time during a prescribed period (*e.g.* one hour), an iteration of Algorithm 2.1 until the expected level of security is reached. Such kind of *gradual authentication* may be of practical interest in pay TV or in systems where a machine (*e.g.* a copy machine or a coffee dispenser) wants to authenticate a physical resource (e.g. an ink or a coffee cartridge).

The second drawback has been recently considered by Gaborit and Girault in [2]. We recall the outlines of their approach in the next section.

2.2 Alternative Scheme Based on Quasi-cyclic Codes

The idea of [2] is to replace the random matrix $\widetilde{H}$ by the parity matrix of a particular type of codes whose representation is very compact: the *quasi-cyclic* codes. Let l be an integer value, the parity matrix H of a $[2l, l, \cdot]$ quasi-cyclic code takes the form $H = (I|A)$, where I denotes the $l \times l$ identity matrix and A is a *circulant matrix*, that is a matrix defined for every $(a_1, a_2, a_3, \cdots, a_l) \in \mathbb{F}_2^l$ by:

$$A = \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_l \\ a_l & a_1 & a_2 & \cdots & a_{l-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_2 & a_3 & a_4 & \cdots & a_1 \end{pmatrix} .$$

As it can be easily checked, representing H does not require to store all the coefficients of the matrix (as it is the case in the original Stern's Scheme) but requires only the l-bit vector $(a_1, a_2, a_3, \cdots, a_l)$ (which is the first row of A). Let n equal $2l$, when replacing the random matrix by a random double-circulant one, the parameter sizes of Stern's Scheme become:

Private data: the secret s of bit-length n.

Public data: the public syndrome i_V of size $\frac{n}{2}$ and the first row of A of size $\frac{n}{2}$, which results in n bits.

It is explained in [2] that for this kind of matrices it is enough to take $l = 347$ and $t = 74$ (and hence $n = 694$). As the new version of Stern's Scheme involves parameters with small sizes and continues to use only elementary logical operations, it becomes highly attractive for light-weight implementations. This is especially true for environments where memory (RAM, PROM, etc.) is a rare resource. The version of Stern Scheme discussed in the rest of the paper involves a $\frac{n}{2} \times n$ *double circulant matrices*.

2.3 Main Operators

A quick analysis of Stern's Protocol shows that the different steps are merely composed of the four following main operators:

Matrix-vector product: the multiplication of a vector by a random double circulant matrix;

Hash function: the action of a hash function;
Permutation: the generation and the action of a random permutation on words;
PRNG: a pseudo-random generator used to generate random permutations and random vectors.

By using quasi-cyclic codes, it becomes possible to implement the prover application of Stern's Scheme in an embedded device (as *e.g.* a smartcard). However, being implemented in a low resource device, the prover application becomes vulnerable to side-channel attacks and appropriate countermeasures must therefore be added. In the two following sections, we discuss about the problematic of side-channel attacks in our context and then, we precise for each of the above operators a way to implement it and how to protect it against side-channel attacks.

3 Side-channel Attacks

Side-channel attacks aim at recovering information about *sensitive variables* appearing in the description of the algorithm under attack. We shall say that a variable is sensitive if it is a function of both public data and of a secret (resp. private) parameter of the algorithm.

In the protocol we described in Fig. 1, we only want to implement Steps 1 and 3 (the ones that rely on the prover) in an embedded device. The other Steps 2 and 4 (that rely on the verifier) may indeed be performed on a PC. For Steps 1 and 3, we have the following list of sensitive variables that can be potentially targeted by SCA:

Threat A. the random vector y in the computations of c_1 (when performing $\tilde{H}y^T$) and of c_2 (when performing $\sigma(y)$): if an attacker is able to retrieve y during one of these steps, then with a probability $1/3$ he is able to recover s when A answers $y \oplus s$ to the $(b = 1)$-challenge.

Threat B. the private vector s during the computation $\sigma(s)$: in this case the attacker recovers A's private parameter.

Threat C. the private vector s during the computation $\sigma(y \oplus s)$: in this case the attacker recovers A's private parameter.

Threat D. the bit-permutation σ during the computation of $\sigma(s)$ or $\sigma(y \oplus s)$: if an attacker is able to retrieve σ during one of these steps, then with a probability $1/3$ he is able to recover s when A answers $\sigma(s)$ to the $(b = 2)$-challenge.

Threat E. the bit-permutation σ during the computation of the hash value $h(\sigma|\tilde{H}y^T)$: if an attacker is able to retrieve σ, then with a probability $1/3$ he is able to recover s when A answers $\sigma(s)$ to the $(b = 2)$-challenge.

Remark 2. When looking for sensitive variables in Fig. 1, we have assumed that the analysis of the device behavior during the storage or the loading of a data does not bring useful information about it. In other terms, we made the classical assumption that information about a data only leaks from calculus involving it

(and eventually other public information) and that data manipulations themselves do not leak enough information on current devices.

To retrieve information about the sensitive data listed above, we assume in the rest of the paper that the SCA adversary can only perform an attack belonging to one of the following three categories:

1. The so-called *timing* attacks consist in analyzing the time taken to execute cryptographic algorithms.
2. The so-called *simple analysis* attacks (SPA in short) are on-line attacks that consist in directly interpreting power consumption measurements and in identifying the execution sequence.
3. The so-called *correlation* attacks (DPA in short) work as *greedy* algorithms: the side-channel information is analyzed by statistical means until the secrets are extracted.

It may be noticed that other categories of SCA attacks exist as for instance the templates or the Higher Order SCA ones. We chose to not consider these attacks for our implementation since they rely on a much stronger (and hence less realistic) adversary than the ones involved in the attacks listed above. For more details about SCA, the reader is referred to [7].

Let us now present the outlines of the defense strategy we shall apply to protect the implementation of the algorithm described in Fig. 1.

Defense Strategy To deal with timings attacks issue, both hardware and software countermeasures are usually involved simultaneously. At the software level, a classical defense strategy consists in implementing all the operations involving sensitive data in a way that does not depend on the data value (for instance methods based on conditional branches are precluded). We chose to follow this strategy for all the operations that are susceptible to manipulate sensitive data.

The most common way of thwarting SPA and DPA involves random values (called *masks*) to de-correlate the leakage signal from the sensitive data which are manipulated. This protection method is usually called *first order masking*. It has been argued in several recent papers (*e.g.* [3, 9, 14]) that this method is sound (when combined with usual hardware protections) to protect an algorithm against SPA and all kinds of first order DPA.

In a first order masking of an algorithm, every sensitive variable y appearing in the algorithm is never directly manipulated by the device and is represented by 2 values $\widetilde{y}$ (the *masked data*) and M (the *mask*). To ensure the DPA-resistance, the mask M takes random values and to ensure completeness, $\widetilde{y}$ satisfies

$$\widetilde{y} = y \oplus M \quad . \tag{1}$$

Since y is sensitive, every function S of y is also sensitive as long as S is known by the attacker. Let z denote this new sensitive value $S(y)$. To mask the processing of z without revealing information on y, two new values $\tilde{z}$ and N must

be computed from $(\widetilde{y}, M)$ (which represents y in the implementation) in such a way that

$$\tilde{z} \oplus N = z = S(y) \ . \tag{2}$$

The critical point of such a method is to deduce the new pair of (masked value)/mask $(\tilde{z}, N)$ from the previous pair (y, M) without compromising the security of the scheme with respect to first order DPA. This problem is usually referred as the *mask correction Problem*.

When S is linear, it can be resolved very efficiently since we have:

$$z = S(y) = S(y \oplus M \oplus M) = S(\widetilde{y}) \oplus S(M) \ , \tag{3}$$

Hence, we simply have to define $\tilde{z}$ and N such that $\tilde{z} = S(\widetilde{y})$ and $N = S(M)$.

Dealing with the mask correction Problem when S is non-linear is much more difficult. Numerous papers have been published which aim to tackle this issue (an overview of the existing methods is proposed in [14]). As argued in [14], when the input and output dimensions n and m of the function S are small, then the so-called *Re-computation* method (REC in short) is the most appropriate one since it only requires one memory transfer and the pre-processing of a RAM table of 2^n elements (one time per algorithm execution):

Re-computation method. Let M and N be two random variables and let us assumed that the RAM look-up table S^* associated to the function $y \mapsto S(y \oplus M) \oplus N$ has be pre-processed. Then, to compute $S(y) \oplus N$ from $\tilde{y} = y \oplus M$, the REC method performs a single operation: the table look-up $S^*[\tilde{y}]$.

As we will see in the next sections, applying first order masking to Stern's Protocol induces only a very small timing overhead and an acceptable memory overhead, since almost all the performed operations are linear (and thus Relation (3) applies most of the time). Moreover, for the few non-linear operations that must be protected (in particular when the bit-permutation σ is computed), we can apply efficiently the REC method since the dimensions of the involved sub-functions are small.

4 Algorithm Specification

In this following, we focus on the four operators defined in Section 2. For each of them, we exhibit an efficient implementation and we discuss about how to protect it effectively against SPA and DPA.

4.1 Matrix-vector Product

Algorithm Description For a Quasi-cyclic Matrix When double-circulant matrices are involved, very efficient algorithms exist to compute the matrix-vector product. In the following, we detail the computation of the product $\tilde{H} \times v$ between the $\frac{n}{2} \times n$ double-circulant matrix $\tilde{H} = (I|A)$ and the n-bit vector v. In our description, we shall denote by *matrix* the $\frac{n}{2}-$bit row vector of A and by *result* an $\frac{n}{2}$-bit temporary vector. Also, we shall denote by $|reg|$ the number of

bits contained in a register of the processor and by *nblocs* the number of blocs in *matrix*: $nblocs = \lceil \frac{n}{2\times|reg|} \rceil$. Additionally, we will denote by v_L (resp. by v_R) the least significant half part of v (resp. the most significant half part of v): namely, we have $v_L = (v_1, \cdots, v_{n/2})$ and $v_R = (v_{n/2+1}, \cdots, v_n)$.

Algorithm 1 Quasi cyclic matrix vector product
INPUT: $matrix = \tilde{H}$,v,$|reg|$
OUTPUT: $result = \tilde{H}v^T$

1. **for** i **from** 1 to $\frac{n}{2}$ **do** $result[i] = v_L[i]$; // initialisation with the first half of the vector
2. **for** i **from** 1 to $|reg|$ **do**
3. **if** $i > 1$; v_R is rotated of one bit to the left;
4. **for** j **from** 1 to $|nblocs|$ **do**
5. **if** the i-th bit of $matrix[j] == 1$; // add v_R to the result beginning with the jth bloc
6. $br = 1$
7. **for** jj **from** j to $|nblocs|$ **do**
8. $result[br] = result[br] \oplus v_R[jj]$;$br = br + 1$;
9. **for** jj **from** 1 to $j - 1$
10. $result[br] = result[br] \oplus v_R[jj]$; $br = br + 1$;
11. **return** $result$

SCA-Security Discussion As argued in Section 3, information about the sensitive data y may leak during the matrix-vector product $\tilde{H}y^T$ (Threat A) and a first order masking must thus be applied. As this product is linear for the bitwise addition and due to (3), masking the calculus is straightforward and implies an acceptable timing/memory overhead.

Before computing $result = \tilde{H}y^t$, the vector y is masked with a n-bit mask M (randomly generated). Then Algorithm 1 is input with *matrix* (*i.e.* the first row of A) and $\tilde{y} = y \oplus M$. The corresponding output is $\tilde{H}\tilde{y}^t = result \oplus N$, where we denoted by N the value $\tilde{H}M^t$. As all the coordinate-bits of y are masked with a uniformly distributed random value, the SPA or the first order DPA analysis of the matrix-vector product does not bring information about y. To make the future unmasking of $\tilde{y}$ possible, a second matrix-vector product $N = \tilde{H}M^T$ is performed and stored in memory together with $\tilde{y}$.

Complexity Discussion : Secure Version For a quasi-cyclic matrix of size $\frac{n}{2} \times n$ whose first row is of weight $p+1$, the following steps have to be undertaken:

- masking the matrix $\tilde{y} = y \oplus M$.
- computing $\tilde{H}\tilde{y}^t = result \oplus N$, where $N = \tilde{H}M^t$.
- a second matrix-vector product $N = \tilde{H}M^T$ is performed and stored in memory together with $\tilde{y}$.
- extract and test the $n/2$ bits of the matrix first row for the product $\tilde{H}\tilde{y}^t$
- extract and test the $n/2$ bits of the matrix first row for the product $\tilde{H}M^t$
- $\lceil \frac{n}{2\times|register|} \rceil$ binary-shifts of the vector $\tilde{y}$

– $\lceil \frac{n}{2\times|register|} \rceil$ binary-shifts of the vector M
– $2p \times \lceil \frac{n}{2\times|register|} \rceil$ registers to be added to the two results

The secure version requires two products matrix vector one for the mask and one for the product to determine. The cost is therefore around the double of the one of the non-secure version.

4.2 Hash Function

To counteract Threat E, the Stern Protocol Implementation must involve a hash function implementation that is secure against first order DPA. Until now, the securing of hash function implementations against SCA has been rarely focused, essentially because these functions usually operate on non-sensitive (often public) data. However, Lemke *et al.* [4] or McEvoy *et al.* [8] have shown that, in some applications like HMAC authentication, mounting DPA attacks against hash functions makes sense when secret (or private) data have to be hashed together with public data. In [8], the authors exhibit a way to protect an hardware implementation of the hash function SHA-256 against first order DPA. In the rest of this section, we will assume that the device on which is implemented the Stern Protocol possesses such a secure hardware implementation of SHA-256. It may be noticed that the masking method used by McEvoy *et al.* for hardware implementations may also be followed to design a masked software implementation of SHA-256. However, in this case, the timing and memory overheads become too large. Actually, if the device does not have a secure SHA-256 implementation, it may be pertinent to use a hash function based on block cipher constructions (the State of the Art of hash functions published by Preneel in [13] give several examples of such functions). Indeed, in such a case the hash function can inherit the DPA-security from the involved block cipher algorithm and the nowadays embedded devices possess almost always a DES Hardware and sometimes an AES Hardware that include anti-DPA mechanisms. In the case where neither secure hash function nor secure block cipher algorithms are implemented in the device, then it is always possible to use one of the numerous DES or AES DPA-secure software implementations proposed in the Literature (see for instance [9, 14]) and to involve them in a hash function based on block ciphers (like for instance MDC-4).

Remark 3. For a hash function to provide a satisfying security, the bit-length of the hash values it produces must be at least 160. When using hash functions based on block ciphers, it may be difficult to get hash values of such a bit-length. In this case, a solution may be to concatenate several output blocks until the bit-length 160 is achieved or exceeded and, if necessary, to truncate in order to get a length of exactly 160 bits (for instance two AES ciphering will result in 256 bits which can be truncated to 160 bits).

Algorithm Description : Secure Hardware Implementation of SHA-256
In [8], McEvoy et al. describe a first order masked hardware implementation of

the HMAC algorithm based on SHA-256. Using an implementation on a commercial FPGA board, they present a masked hardware implementation of the algorithm, which is designed to counteract first-order DPA attacks.

SCA-Security Discussion In [8], the resistance of the SHA-256 implementation is formally analyzed and demonstrated.

Complexity Discussion It is shown in [8] that the processor and the interface circuitry corresponding to the masked SHA-256 utilize 1734 slices (37% of the FPGA resources) and that the critical path in the design (i.e. the longest combinational path) is 18.6. As argued in [8], the area has almost doubled compared with the unprotected implementation but the speed has not been overly affected.

4.3 Permutation Method

Defining a vectorial permutation σ over $\mathbb{F}_2^n$ (like the one used in Figure 1) amounts to define an *index permutation* ψ over $\{0, \cdots, n-1\}$ such that for every $y = (y[0], \cdots, y[n-1]) \in \mathbb{F}_2^n$ we have $\sigma(y) = (y[\psi(0)], \cdots, y[\psi(n-1)])$. In this paper, we chose to design the permutation ψ by following the approach suggested by Luby and Rackoff in [5, 6] and improved in [10, 12]. The core idea of this approach is to involve a few pseudo random functions in a Feistel Scheme. As argued by the authors in [10, 12], such a method makes it possible to design random permutations very efficiently since only a few Feistel rounds are needed and since the input/output dimensions of the involved functions are more or less logarithmic in the size of the words on which the permutation operates.

Let us first recall some basic facts about the so-called Luby-Rackoff schemes.

Luby-Rackoff's Scheme For every function f defined from $\mathbb{F}_2^l$ into $\mathbb{F}_2^m$, the Feistel round involving f, denoted by $\psi(f)$, is defined for every pair $(L, R) \in \mathbb{F}_2^m \times \mathbb{F}_2^l$ by $\psi(f)[L, R] = [R, L \oplus f(R)]$. The composition of k Feistel rounds, that is the function $\psi(f_k) \circ \ldots \circ \psi(f_1)$, is denoted by $\psi(f_1, \ldots, f_k)$ or by ψ in short if there is no ambiguity about the involved functions.

If f and g are two randomly generated independent functions defined from $\mathbb{F}_2^m$ into itself, then it has been argued in [10, 12] that the function $\psi(g, f, g, f)$ is indistinguishable from an uniform distribution by an observer, even if the latter has access to the inverse permutation. As a consequence, to design an index-permutation ψ over $\{0, \cdots, 2^{2m} - 1\}$ (and thus a vectorial bit-permutation σ over $\mathbb{F}_2^n$ with $\log_2(n) = 2m$), we simply need to generate the two independent random functions f and g.

Once the Luby-Rackoff scheme has been designed for two functions f and g, there are merely two strategies to compute the bit-permutations $\sigma(y)$, $\sigma(y \oplus s)$ and eventually $\sigma(s)$ in Stern's Protocol. The fist one consists in pre-computing $\psi(i)$ for every $i \leq n$ and then to store the sequence $(\psi(i))_{i \leq n}$ as a representation of σ. In this case, each time σ must be applied to a vector, then its table

representation is accessed n times (one time for each bit-index). This strategy requires the RAM allocation of $n \times \lceil \log_2(n) \rceil$ bits, which is quite expensive in a low resource context. The second strategy consists in computing $(\psi(i))_{i \leq n}$ each time one needs to determine the bit index corresponding to i in σ. This strategy, which has been chosen for our implementation, is more time consuming than the previous one but it does not require any RAM allocation.

By construction, Luby-Rackoff Scheme only permits to construct index permutations ψ such that n is a power of 2. Since the size parameter n we consider for Stern's Scheme is 347 (see Section 2), we couldn't use Luby-Rackoff Scheme straightforwardly, but a slightly modified version of it.

Algorithm Description In Section 2 we argued that the parameters size (of s and y) should be at least $n = 694 = 2 \times 347$. Let m denote the value $\lceil \log_2(n) \rceil / 2$. To implement a permutation on vectors of any bit-length n such that $\lceil \log_2(n) \rceil$ is even, we suggest hereafter to randomly generate two functions f and g defined from $\mathbb{F}_2^m$ into itself and to use the Luby-Rackoff Scheme in the following way:

Algorithm 2 Bit-permutation for any n such that $\lceil \log_2(n) \rceil$ is even

INPUT: the vector v to permute, the bit-length n of v, the value $n' = 2^{\lceil \log_2(n) \rceil}$, a Luby-Rackoff Scheme $\psi(g, f, g, f)$ with f and g defined from $\mathbb{F}_2^{\lceil \log_2(n) \rceil / 2}$ into itself.
OUTPUT: the vector $result = \sigma(v)$

1. **for** i **from** 0 **to** $n' - 1$ **do** $T[i] \leftarrow 0$
2. **for** i **from** 0 **to** $n' - 1$ **do**
3. $new\_index \leftarrow \psi(i)$; $result[new\_index] \leftarrow v[i]$; $T[new\_index] \leftarrow 1$;
4. $j \leftarrow n$;
5. **for** i **from** 0 **to** $n - 1$ **do**
6. **if** $(T[i] = 0)$
7. **while** $(T[j] = 0)$ **do** $j \leftarrow j + 1$;
8. $result[i] \leftarrow result[j]$; $j \leftarrow j + 1$
9. **return** $result$

Remark 4. Table T needs to be computed only one time per each permutation σ. Once computed, it can be used for all the permutation involving σ.

In Algorithm 2, each iteration of the second loop computes the bit-index new_index in $result$ where to store the bit-value $v[i]$. During this processing, Table T keeps trace of the $result$ bit-coordinates that are updated during this process. When the loop is ended, a third loop is iterated to fill the bit-coordinate of index $i < n$ that has not been initialized by the second loop (which are the ones such that $T[i] = 0$), with the bit-coordinates of index $n \leq j < n'$ that has been initialized (which are such that $T[j] = 1$).

SCA-Security Discussion In order to thwart Threats C and D (see Section 3), we chose to mask the computations of σ and ψ. Since σ is at most used 3 times before being re-generated, it may be targeted by SPA attack but does not suffer from DPA. The linearity of σ makes it easy to mask its processing: we mask the input y with a random mask M (which results in a masked input $\tilde{y} = y \oplus M$) and we unmask the output $\sigma(\tilde{y})$ by simply x-oring it with $\sigma(M)$.

In Algorithm 2, the same function ψ is applied $2^{n'}$ times on known input before being re-generated. It can thus be targeted by DPA attacks. To counteract them, we chose to mask the intermediate variables that appear during the processing of ψ and to apply the REC method to deal with the mask correction problem when the functions f and g are used. Each time new functions f and g are generated (thus defining a new function ψ), we generate two random masks $r, s \in \mathbb{F}_2^m$ and we define two new functions f^* and g^* such that $f^*(x) = f(x \oplus r) \oplus s$ and $g^*(x) = g(x \oplus s) \oplus r$. We describe the normal and secure processing of ψ in Figure 2.

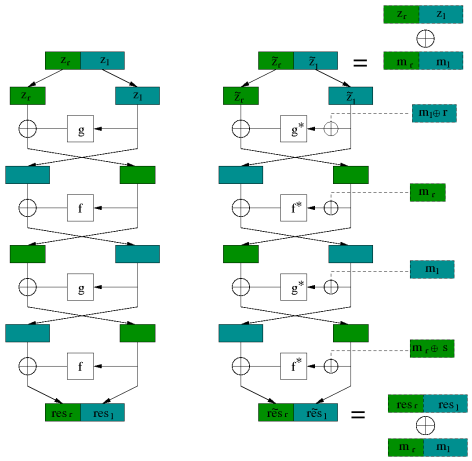

Fig. 2. Luby Rackoff permutation - unsecure and secure versions.

Complexity Discussion The function ψ must be re-generated at each execution of Algorithm 2. This requires the generation of $2 \times m \times 2^m$ random bits to define the functions f and g. The un-secure processing of ψ involves 4 look-up tables and 4 bitwise m-bit additions. Its secure processing requires the pre-computation of the new RAM lookup-tables f^* and g^* (with complexity $O(2^m)$) and 8 additional bitwise m-bit additions (4 for the mask correction and 2×2 for the masking/unmasking of the input/output) compared to the un-secure calculus.

Algorithm 2 involves two n'-bit local variables T and *result*. It processes n' times the function ψ and executes two loops involving around respectively $2 \times n'$ and $4 \times n'$ elementary operations. This results, for the σ processing, in $14n' = 8 \times n' + 2 \times n' + 4 \times n'$ elementary operations and in the generation of $2 \times m \times 2^m$ random bits for the processing of ψ without any SCA countermeasure. In the

SCA-secure mode, the processing of σ requires $22n' = (8+8) \times n' + 2 \times n' + 4 \times n'$ elementary operations and the generation of $2 \times m \times 2^m$ random bits for f and g, one n-bit vectors M to mask the input y of σ and one $2m$-bit vector (m_r, m_l) to mask all the input of ψ. To make the final unmasking of $\sigma(y \oplus M)$ possible, the vector $\sigma(M)$ must also be computed, which adds $22n'$ elementary operations (note that the mask (m_r, m_l) does not need to be re-generated to protect the processing of ψ for $\sigma(M)$). Finally, we get for the secure processing in the secure mode, around $44n'$ (*i.e.* $44 \times 2^{\lceil \log_2(n) \rceil}$) elementary operations and the generation of $n + 2 \times m \times 2^m + 2m$ random bits (*i.e.* $n + \lceil \log_2(n) \rceil \times 2^{\lceil \log_2(n) \rceil / 2} + \lceil \log_2(n) \rceil$).

Example 1. For the choice of parameter size done in Section 2 (*i.e.* $n = 694$), we have $m = \lceil \log_2(n) \rceil / 2 = \lceil \log_2(694) \rceil / 2 = 5$ and $n' = 2^{\lceil \log_2(n) \rceil} = 2^{10} = 1024$. In such a case, the processing of σ without any security requires around $14 \times 10^3 \approx 14 * 1024$ elementary operations and the generation of $320 = 10 \times 32$ random bits. In the SCA-secure mode, it requires around $45 \times 10^3 \approx 44 * 1024$ elementary operations and the generation of $1024 = 694 + 10 \times 32 + 10$ random bits.

Example 2. For $n = 512$ (which is the choice of parameter size done in Section 5), the processing of σ without any security requires around $7 \times 10^3 \approx 14 * 512$ elementary operations and the generation of $224 = 4 \times 2^4 + 5 \times 2^5$ random bits (note that in this case, since $\log_2(512) = 9$ is odd, the functions f and g cannot have the same dimensions and we chose f being from $\mathbb{F}_2^4$ into $\mathbb{F}_2^5$ and g being from $\mathbb{F}_2^5$ into $\mathbb{F}_2^4$). In the SCA-secure mode, it requires around $22 \times 10^3 \approx 44 * 512$ elementary operations and the generation of around 450 random bits.

4.4 Pseudorandom Generator

We need a pseudorandom generator to construct the seed of the code and the permutation. Nowadays, most of the pseudo-random generators used in commercial applications are either based on stream cipher or on block cipher algorithms. Hardware implementations of stream cipher are often faster than the ones of block ciphers. However, there are only available in some specific devices (and are for instance not available in most of the smart cards), whereas block ciphers algorithms such as DES or AES are almost systematically implemented in hardware. We consider that the Pseudo Random Number Generator (PRNG) that we will use to generate the seed is not biased and secure against SPA and DPA attacks. In [11], the author present a block cipher-based PRNG secure against side-channel key recovery.

5 Implementation

5.1 Experimental Results

We have realized the implementation of Stern Authentication with double circulant matrices for $l = 256$ (*i.e.* $n = 512$) on a 8051-architecture **without crypto-processor** nor hardware SHA-256, and with a CPU running at 12 MHz.

Time for 1 round	Time (ms)
PRNG (vector y, function f and g)	16.7
Matrix-vector product	22.0
Permutations (and an xor)	22.6
Hash function (SHA-256)	107.6
Total for one round	168.9
Authentication (35 rounds)	5 911.5

Table 1. Performances of the implementation

Remark 5. Timing performances given in Table 1 do not take the communication cost into account. This choice has been made because the transmission rate highly depends on the application type. For instance, the today VISA norm imposes 9600 bauds (which is quite low), whereas the nowadays technologies make it possible to have 110000 bauds for transmission rate.

We obtain an authentication in ≈ 6 seconds and a signature in ≈ 24 seconds for a security of 2^{85}. The communication cost is around 40-kbits in the authentication scheme and around 140-kbits for the signature. It must be noticed that the timing performances would be highly improved by using a hardware SHA-256 instead of a software implementation.

The implementation detailed above doesn't include SCA countermeasures. According to the study conducted in Section 4, the timing/memory overhead expected after securization is around ($\times 3$). This value is really small compared for instance to a secure software version of the AES where the overhead is around $\times 10$.

6 Conclusion and Future Work

We have described in this paper the first implementation of Stern protocol on smart card (in fact it is also more generally the first code-based system implemented on smart-card with usual resources). For a satisfying security level, the size of the public key is only 694 bits using a quasi cyclic representation of the matrix considered. The double-circulant matrices are a good trade-off between random and strongly structured matrices. In this case the operations are indeed really simple to perform and can be implemented easily in hardware. Moreover, the fact that the protocol essentially performs linear operations makes the algorithm easy to protect against side channel attacks. We thus think that the protocol is a new option to carry out fast strong authentication on smart cards. Additionally, we think that the use of a dedicated linear-algebra co-processor should significantly improve the timing performances of our implementation.

Future work besides this one includes considering Fault injection attacks (this was beyond the scope of this paper) and implementation of other variations of Stern protocol which can have other small advantages (see [2]) for protocol variations.

References

1. A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In A. Odyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.
2. P. Gaborit and M. Girault. Lightweight code-based identification and signature. *IEEE Transactions on Information Theory (ISIT)*, pages 191–195, 2007.
3. L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
4. K. Lemke, K. Schramm, and C. Paar. DPA on n-bit sized boolean and arithmetic operations and its applications to IDEA. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 205–219. Springer, 2004.
5. M. Luby and C. Rackoff. Pseudo-random permutation generators and cryptographic composition. *Symposium on Theory of Computing*, vol. 18, pages 353–363, 1986.
6. M. Luby and C. Rackoff. How to construct pseudorandom permutation and pseudorandom functions. *SIAM J. Comput.*, vol. 17, pages 373–386, 1988.
7. S. Mangard, E. Oswald, and T. Popp. Power Analysis Attacks – Revealing the Secrets of Smartcards. Springer, 2007.
8. R. McEvoy, M. Tunstall, C. Murphy, and W. P. Marnane. Differential power analysis of HMAC based on SHA-2, and countermeasures. In S. Kim, M. Yung, and H.-W. Lee, editors, *WISA*, volume 4867 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2008.
9. E. Oswald and K. Schramm. An Efficient Masking Scheme for AES Software Implementations. In J. Song, T. Kwon, and M. Yung, editors, *WISA 2005*, volume 3786 of *Lecture Notes in Computer Science*, pages 292–305. Springer, 2006.
10. J. Patarin. How to construct pseudorandom and super pseudorandom permutation from one single pseudorandom function. In R. Rueppel, editor, *Advances in Cryptology – EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1992.
11. C. Petit, F.-X. Standaert, O. Pereira, T. G. Malkin, and M. Yung. A Block Cipher based PRNG Secure Against Side-Channel Key Recovery. Available at `http://eprint.iacr.org/2007/356.pdf`.
12. J. Pieprzyk. How to construct pseudorandom permutations from single pseudorandom functions advances. In I. Damgård, editor, *Advances in Cryptology – EUROCRYPT '90,Lecture Notes in Computer Science*, vol. 473 , pages 140–150. Springer, 1990.
13. B. Preneel. Hash functions - present state of art. *ECRYPT Report*, 2005.
14. E. Prouff and M. Rivain. A Generic Method for Secure SBox Implementation. In S. Kim, M. Yung, and H.-W. Lee, editors, *WISA,Lecture Notes in Computer Science*, vol. 4867, pages 227–244. Springer, 2008.
15. J. Stern. A new identification scheme based on syndrome decoding. In D. Stinson, editor, *Advances in Cryptology – CRYPTO '93*, *Lecture Notes in Computer Science*, vol. 773, pages 13–21. Springer, 1993.