Secure and Efficient RNS Approach for Elliptic Curve Cryptography

Apostolos P. Fournaris
Electrical and Computer
Engineering Dpt.
University of Patras, Greece
Email: apofour@ieee.org

Louiza Papachristodoulou, Lejla Batina
Digital Security Group,
Radboud University Nijmegen,
The Netherlands
Email: {louizap, lejla@cs.ru.nl}

Nicolas Sklavos
Computer Engineering
& Informatics Dpt.
University of Patras, Greece
Email: nsklavos@ceid.upatras.gr

***Abstract*—Scalar multiplication, the main operation in elliptic curve cryptographic protocols, is vulnerable to side-channel (SCA) and fault injection (FA) attacks. An efficient countermeasure for scalar multiplication can be provided by using alternative number systems like the Residue Number System (RNS). In RNS, a number is represented as a set of smaller numbers, where each one is the result of the modular reduction with a given moduli basis. Under certain requirements, a number can be uniquely transformed from the integers to the RNS domain (and vice versa) and all arithmetic operations can be performed in RNS. This representation provides an inherent SCA and FA resistance to many attacks and can be further enhanced by RNS arithmetic manipulation or more traditional algorithmic countermeasures. In this paper, extending our previous work, we explore the potentials of RNS as an SCA and FA countermeasure and provide an description of RNS based SCA and FA resistance means. We propose a secure and efficient Montgomery Power Ladder based scalar multiplication algorithm on RNS and discuss its SCA-FA resistance. The proposed algorithm is implemented on an ARM Cortex A7 processor and its SCA-FA resistance is evaluated by collecting preliminary leakage trace results that validate our initial assumptions.**

I. Introduction

RNS is an arithmetic representation of integer numbers that is advantageous when it comes to parallel arithmetic calculations. The system can also be used to represent elements of cyclic groups or finite fields. In RNS, a number is represented by a given moduli base (RNS base) consisting of several base elements. The fact that each modulo of a number can be processed in parallel (for any type of operation) as well as the fact that a single bit fault in a moduli can lead to ”difficult to trace” changes in an overall number, hints that there is fertile ground for introducing RNS related Side Channel Attack (SCA) and Fault injection Attack (FA) countermeasures in cryptosystems [2] [4] [14] [18] [9] [10].

Scalar multiplication, the key operation behind Elliptic Curve Cryptography (ECC), relies heavily on finite field to perform all arithmetic operations. Thus, the introduction of RNS as the number system for $GF(p)$ elements and their operations can be a step towards increasing SCA/FA resistance. However, this does not constitute enough protection against SCAs/FAs nor does it guarantee efficient implementations.

Several researchers have made observations about the potentials of RNS as a side-channel attack countermeasure as well as a fault injection attack countermeasure. Bajard et al. in [2] proposes, originally for modular exponentiation, a random permutation of the moduli bases. The periodic change using base permutation during the modular exponentiation (and consecutively scalar multiplication) computation flow can introduce enough randomness to thwart SCAs. This approach leads to a leak resistant arithmetic (LRA) technique that can be applied to modular exponentiation designs (used for RSA) in two ways, either by choosing a new base permutation once at the beginning of each modular exponentiation or by changing a permutation in each modular multiplication operation of the exponentiation process [10].

In this paper, we explore the RNS potentials in elliptic curve scalar multiplication, extending the work done in [10], taking into account both aspects of security (SCA/FA resistance) and efficiency. The various ways of transformation from RNS arithmetic to binary are explored as well as the way of performing RNS modular multiplication. The RNS version of the Montgomery modular multiplication algorithm is described and its importance as the basis of both efficient and secure RNS scalar multiplication implementation is described. We present the current approaches on designing RNS Montgomery multiplication and argue that the algorithm can be used as a basis for SCA/FA resistance in scalar multiplication. To justify the above argument, a variation of a scalar multiplication algorithm is described that offers SCA/FA resistance through the combination of RNS characteristics and traditional scalar multiplication algorithmic countermeasures like the Montgomery Power Ladder. The RNS based resistance of our approach adopts the random permutation of the RNS bases in each scalar multiplication algorithmic round, thus adapting the LRA technique for scalar multiplication and modifying it in order to achieve efficiency yet retain the disassociation of secret information from physical leakage. Furthermore, in our approach we take advantage of the base extension operation that is performed during an RNS Montgomery modular multiplication in order to enhance fault detection. To verify the correctness of our approach and to test its security and efficiency, we implemented the described scalar multiplication algorithm in the ARM Cortex A7 processor of a Raspberry Pi 2 using GMP C library as a basis for all arithmetic operations.

The rest of the paper is organized as follows. In section II the RNS arithmetic for ECC is presented. Section III presents the employed algorithm and argues over its SCA/FA resistance. Details of our implementation are included in section IV with some preliminary results of our SCA/FA analysis. Finally, section V concludes the paper.

II. RNS for EC Point operations

A number x can be represented in RNS as a set of n moduli x_i ($x \overset{RNS}{\rightarrow} X : (x_1, x_2, ...x_n)$) of a given RNS basis $B : (m_1, m_2, ...m_n)$ as long as $0 \leq x < M$ where $M = \prod_{i=1}^{n} m_i$ is the RNS dynamic range and all m_i are pairwise relatively prime. Each x_i can be derived from x by calculating $x_i = \langle x \rangle_{m_i} = x \ mod \ m_i$. Assuming that we have two numbers a and d represented in RNS as $A : (a_1, a_2, ...a_n)$ and $D : (d_1, d_2, ...d_n)$ we can obtain addition, subtraction and multiplication in RNS as $A \oslash D = (\langle a_1 \oslash d_1 \rangle_{m_1}, ... \langle a_n \oslash d_n \rangle_{m_n})$ where $\oslash : (+, -, \times)$. Exact division by D coprime with M is equivalent to multiplying by the inverse $\langle D^{-1} \rangle_M$. Since RNS is a non-positional representation, comparisons, divisions and modular reductions are complex operations, which are performed either by converting the number from RNS to binary representation or by using base extension algorithms.

Binary reconstruction from RNS representation can be done using the Chinese Remainder Theorem (CRT) $x = \left\langle \sum_{i=1}^{n} \langle x_i \cdot M_i^{-1} \rangle_{m_i} \cdot M_i \right\rangle_M$ where $M_i = \frac{M}{m_i}$ and M_i^{-1} is the multiplicative inverse of M_i. The required M modulo reduction, due to the high bit length of M , is not efficiently realized and is usually performed by introducing a correction factor w, where $x = \sum_{i=1}^{n} \langle x_i \cdot M_i^{-1} \rangle_{m_i} \cdot M_i - w \cdot M$.
To avoid the above process, x's Mixed Radix System (MRS) representation $\widetilde{X} : (u_1, u_2, ...u_n)$ can be used for RNS to binary conversion. The MRS number $\widetilde{X}$ can be obtained from $X : (x_1, x_2, x_3, ...x_n)$ by executing the Mixed Radix Conversion (MRC) algorithm of (1).

$$\begin{aligned} & u_1 = x_1 \quad u_2 = \langle (x_2 - u_1) \cdot m_{1,2}^{-1} \rangle_{m_2} \\ & u_3 = \langle ((x_3 - u_1) \cdot m_{1,3}^{-1} - u_2) \cdot m_{2,3}^{-1} \rangle_{m_3} \\ & \dots \\ & u_n = \langle ((x_n - u_1) \cdot m_{1,n}^{-1} - u_2) \cdot m_{2,n}^{-1} - \dots \\ & - u_{n-1}) \cdot m_{n-1,n}^{-1} \rangle_{m_n} \end{aligned} \tag{1}$$

where $m_{i,j}^{-1}$ is the multiplicative inverse of m_i modulo m_j i.e. $m_i \cdot m_{i,j}^{-1} \equiv 1 \ mod \ m_j$. From the MRS number representation, an integer x can be recovered by performing $x = u_1 + g_2 u_2 + g_3 u_3 + \ldots + g_n u_n$ where $g_i = \prod_{j=1}^{i} m_j$.

For ECC approved ECs defined over $GF(p)$ (EC on $GF(2^k)$ are not discussed in this paper), all $GF(p)$ operations (addition, subtraction, multiplication) are modular operations. Performing RNS $GF(p)$ addition or subtraction can be easily realized by expressing p in RNS format i.e. $P : (p_1, p_2, p_3, ...p_n)$ and calculating:

$$\begin{aligned} & (A \oslash D) mod \ P = (\left\langle \langle a_1 \oslash d_1 \rangle_{m_1} \right\rangle_{p_1}, \left\langle \langle a_2 \oslash d_2 \rangle_{m_2} \right\rangle_{p_2}, .. \\ & \ldots \left\langle \langle a_n \oslash d_n \rangle_{m_n} \right\rangle_{p_n}) \quad where \quad \oslash : (+, -) \end{aligned} \tag{2}$$

However, RNS modular multiplication over $GF(p)$ is a computationally difficult operation. It is usually realized through the RNS Montgomery multiplication algorithm that avoids modular inversions, but includes base extension operations [3] [10].

A. RNS Base Extension

Assuming that we introduce two RNS bases $B_n = (m_1, m_2, \ldots, m_n)$ and $\acute{B_n} = (m_{n+1}, m_{n+2}, \ldots, m_{2n})$ such that $gcd(m_i, m_j) = 1$ for all $i \in \{1, n\}$ and $j \in \{n+1, 2n\}$, we express a GF(p) number x in base B_n or $\acute{B_n}$ as X_B and $X_{\acute{B}}$ respectively while in both RNS bases as $X_{B \cup \acute{B}}$. We define $M_{B \cup \acute{B}}$ as $M_{B \cup \acute{B}} \prod_{i=1}^{2n} m_i$, also, $M_B = \prod_{i=1}^{n} m_i$ and M_B^{-1} as the multiplicative inverse of M_B in base B_n as well as $M_{\acute{B}} = \prod_{i=n+1}^{2n} m_i$ and $M_{\acute{B}}^{-1}$ as the multiplicative inverse of $M_{\acute{B}}$ in base $\acute{B_n}$. The RNS Montgomery multiplication (RNSMM) is presented as Algorithm 1 and as an outcome calculates $S_B = A \cdot B \cdot M_B^{-1} \ modp$ and $S_{\acute{B}} = A \cdot B \cdot M_B^{-1} \ modp$. Base extension from one base to the other in Algorithm 1 is needed since M_B^{-1} does not exist in base B_n and therefore computations must be migrated to the $\acute{B_n}$ base to come up with S_B.

$$\begin{aligned} & x_j = \langle \acute{x_1} + \cdots + m_{n-3} (\acute{x_{n-2}} + (m_{n-2} (\acute{x_{n-1}} + (m_{n-1} \acute{x_n})))) \rangle_{m_j} \\ & for \quad all \quad j \in \{n+1, n+2, ...2n\} \ of \acute{B_n} \end{aligned} \tag{3}$$

```
Algorithm 1. RNS Montgomery Modular Multiplication
$RNSMM(A, D, P, B_n, \acute{B_n})$
Input: $B_n = (m_1, \ldots, m_n), \acute{B_n} = (m_{n+1}, \ldots, m_{2n})$,
$P_{B \cup \acute{B}} = P_B \cup P_{\acute{B}} : (p_1, p_2, ...p_n, p_{n+1}, ...p_{2n})$,
$M_B = \prod_{i=1}^{n} m_i$, $M_{\acute{B}} = \prod_{i=n}^{2n} m_i$,
$A_{B \cup \acute{B}} = A_B \cup A_{\acute{B}} : \{a_1, \ldots, a_n, a_{n+1}, \ldots a_{2n}\}$,
$D_{B \cup \acute{B}} = D_B \cup D_{\acute{B}} : \{d_1, \ldots, d_n, d_{n+1}, \ldots, d_{2n}\}, M_{\acute{B}}^{-1}, -P_B^{-1}$
Output: $S_B = A_B \cdot D_B \cdot M_B^{-1} \ modP_B$ and
        $S_{\acute{B}} = A_{\acute{B}} \cdot D_{\acute{B}} \cdot M_B^{-1} \ modP_{\acute{B}}$
1. $G_{B \cup \acute{B}} = A_{B \cup \acute{B}} \times D_{B \cup \acute{B}}$
i.e. ($g_i = \langle a_i \times d_i \rangle_{m_i}$ in base $B_n$ and $\acute{g_i} = \left\langle \acute{a_i} \times \acute{d_i} \right\rangle_{\acute{m_i}}$ in base $\acute{B_n}$)
2. $Q_B = G_B \times (-P_B^{-1})$ i.e. $\left( \left\langle g_i \times \langle -p^{-1} \rangle_{m_i} \right\rangle_{m_i} \right)$ in $B_n$
3. $Q_B \rightarrow Q_{\acute{B}}$ Base extension $B_n \rightarrow \acute{B_n}$
4. $R_{\acute{B}} = G_{\acute{B}} + Q_{\acute{B}} \times P_{\acute{B}}$
5. $S_{\acute{B}} = R_{\acute{B}} \times M_{\acute{B}}^{-1}$
6. $S_{\acute{B}} \rightarrow S_B$ Base extension $\acute{B_n} \rightarrow B_n$
Return $S_B$ and $S_{\acute{B}}$
```

Two main approaches to base extension are used in practice for RNS arithmetic: the MRS system and the Cox-Rower architecture introduced in [16]. The Cox-Rower architecture consists of parallel arithmetic units, the Rowers, which perform the independant computations for each base concurrently, and the Cox unit dedicated to the computation of an approximation of the factor w. Therefore, it can be efficiently implemented in hardware. An interesting work in protecting the Cox-Rower architecture against multi-fault attacks is presented in [1].

The MRS system is often used for RNSMM base extension, despite the fact that it is sequential and therefore slower compared to Cox-Rower. As a first step of MRS, the base B_n RNS number is converted into a base B_n MRS number following (1). In the second step, the base B_n MRS number is converted into a base $\acute{B_n}$ RNS number according to (3). A similar two step procedure is followed for base extension from $\acute{B_n}$ to B_n respectively.

It must be noted that each RNS number A used in Montgomery multiplication must be represented in the Montgomery format, meaning in the form $A_B \cdot M_B \ modP_B$ or $A_{\acute{B}} \cdot M_{\acute{B}} \ modP_{\acute{B}}$. To transform a number in the Montgomery normalized form, an RNSMM must be performed between A and $M_{B\cup\acute{B}} \ modP$ using the bases B_n and $\acute{B_n}$ in reverse order (i.e. $RNSMM(A, M_{B\cup\acute{B}} \ modP, P, \acute{B_n}, B_n)$). To leave the Montgomery domain we must perform an RNSMM of the Montgomery formatted RNS number A with 1 (i.e. $RNSMM(A, 1, P, B_n, \acute{B_n})$).

Efficient base extension operation heavily relies on the choice of B_n and $\acute{B_n}$. To increase computation efficiency, the bases' moduli must be chosen so that their multiplicative inverses are small numbers. Most studies on optimal base moduli [7] [5] agree that moduli of the form $2^k \pm c_i$, $2^k - 2^{t_i} \pm 1$ or 2^k, $2^k - 1$, $2^{k-1} - 1$ $2^{k+1} - 1$ (Mersenne numbers) for various i values provide good performance results. Each base's moduli (n) number must also be optimally determined as well as each moduli's k value (defining all involved values bit length). Usually, such numbers are specified according to the GF(p) defining the EC. The RNS bases B_n and $\acute{B_n}$ dynamic range must be close to p ($4p < M$). Recent results from Bigou and Tisserand in [6] show how to perform RNS modular multiplication with a single base bit width instead of a double one, which results in two times faster implementation for the same area.

B. Using RNS for SCA and FA resistance

Several researchers have pointed out the potentials of RNS as a side-channel and fault injection attack countermeasures. Bajard et al. in [2] proposes, originally for modular exponentiation, a random permutation of the base B_n and $\acute{B_n}$ moduli thus creating $\binom{2n}{n}$ random permutations of B_n and $\acute{B_n}$. We denote each such RNS Base γ permutation as $B_{n,\gamma}$ and $\acute{B_{n,\gamma}}$. The periodic change of a base permutation during the modular exponentiation (and consecutively scalar multiplication)computation flow , as presented in Figure 1, can introduce enough randomness to thwart SCAs. This approach leads to a leak resistant arithmetic (LRA) technique that can be applied to modular exponentiation designs (used for RSA) in two ways, either by choosing a new base permutation once at the beginning of each modular exponentiation or by changing a permutation in each RNSMM operation of the exponentiation process. The base transition of an RNS number A represented in a base permutation γ to a new permutation $\acute{\gamma}$ can be done by performing two consecutive RNSMMs. Initially $A_1 = RNSMM(A, M_{B\cup\acute{B}} \ mod \ P, P, \acute{B}_{n,\acute{\gamma}}, B_{n,\acute{\gamma}})$ [1] is performed and it is followed by $RNSMM(A_1, 1, P, \acute{B_{n,\gamma}}, B_{n,\gamma})$

[1]Note that A has the form $A \cdot M_{B_{n,\gamma}} \ mod \ P$ (Montgomery form) since it is an output of some previous RNSMM

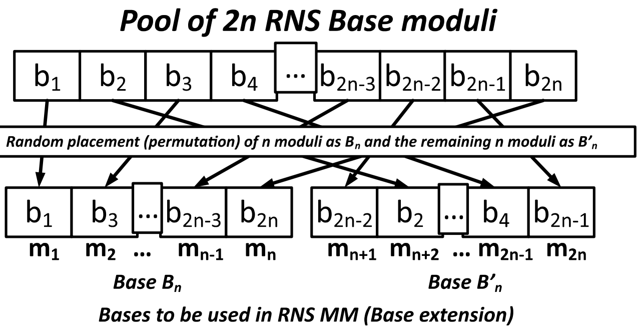
Pool of 2n RNS Base moduli
b1 b2 b3 b4 ... b2n-3 b2n-2 b2n-1 b2n
Random placement (permutation) of n moduli as Bn and the remaining n moduli as B'n
b1 b3 ... b2n-3 b2n
b2n-2 b2 ... b4 b2n-1
m1 m2 ... mn-1 mn
mn+1 mn+2 ... m2n-1 m2n
Base Bn
Base B'n
Bases to be used in RNS MM (Base extension)

Fig. 1. Leak Resistant Arithmetic approach on Base Randomization

Applying the LRA technique in scalar multiplication follows a similar approach to modular exponentiation. Some attempts to introduce LRA in scalar multiplication have been made in [14] [13], however, they are applicable only to the CRT type of base extension using the Cow-Rower method when pseudo-Mersenne numbers are used for base moduli. In scalar multiplication, a permutation transition can be done only once (per scalar multiplication), in every round of the scalar multiplication process or before every $GF(p)$ RNSMM operation of every point operation of every round. Taking into account that the transition from one permutation to another costs 2 RNSMM, the third approach is not affordable in terms of speed. The first approach, providing a single randomization per scalar multiplication may be vulnerable to horizontal SCA attacks (depending on the employed implementation methodology) so of special interest is the second approach were the RNS bases are permuted once per scalar multiplication round. This approach offers a promising balance between performance and SCA resistance strength.

RNS has a long history as fault tolerance and detection tool and thus can be used for identifying possible FAs. Fault detection through RNS is achieved by introducing redundancy during RNSMM as described in [4] [17]. In the existing two RNS bases moduli B_n and $\acute{B_n}$ used in RNSMM, a redundant moduli m_r is added. Thus the RNSMM algorithm is executed using redundant bases $B_n \cup m_r$ and $\acute{B_n} \cup m_r$. Key point in the detection process is base extension of the RNS values during RNSMM from base B_n to $\acute{B_n} \cup m_r$ instead of $\acute{B_n}$ (in step 3 of Algorithm 1) as well as base extension from $\acute{B_n}$ to $B \cup m_R$ (in step 6 of Algorithm 1). The redundant RNSMM algorithm results $S_{B\cup m_r}$ and $S_{\acute{B}\cup m_r}$ include moduli related to base element m_r $\left(\text{i.e.} \left\langle S_{B\cup m_r}\right\rangle_{m_r} \text{ and } \left\langle S_{\acute{B}\cup m_r}\right\rangle_{m_r}\right)$. If no fault is injected during an RNSMM then the 2 moduli must be the same. This approach is capable of detecting a single fault during a RNSMM and its main additional performance cost (compared to the original RNSMM) is associated with the RNS Base extension operations. A similar fault detection technique was proposed in [14] but is applicable only to Cox Rower RNSMM designs (that use the CRT base extension method) while the technique described here and proposed in

[4] [10] is generic and can be applied to any base extension methodology.

III. FA and PA Resistant Scalar Multiplication

Given the description of RNS PA and FA countermeasures, we adopt the inclusion of LRA as an add-on countermeasure in an PA resistant SM algorithm in order to provide horizontal and vertical attacks resistance. In the described algorithm (Algorithm 2), LRA is combined with the base point blinding technique (additive randomization of the EC base point V) in the Montgomery Power Ladder (MPL) algorithm expanding the work of [11] [9] and [10]. MPL is considered secure against most vertical and horizontal attacks.

In Algorithm 2, we introduce LRA RNS base randomization once in each SM round (steps 4c and 4d) and in that way manage to include a different randomization element in every round. The input point V is initially blinded by adding to it a random element R, thus preventing sophisticated, comparative simple PAs [8]. MPL is a highly regular SM algorithm since it always performs two point operations per round, regardless of the scalar bit e_i. It also provides an intrinsic fault detection mechanism based on the mathematical coherence of R_0 and R_1. As observed in [15] and by Giraud in [12], the R_0 and R_1 points in an MPL round always satisfy the equation $R_0 = V + R_1$. Injecting a fault during computation in an R_1 or R_0 variable will ruin this coherence and by introducing an MPL coherence detection mechanism in the end of the MPL algorithm, this fault will always be detected. This technique is adopted in step 6 of Algorithm 1 where $R_0 + V \neq R_1$ if a fault is injected. Note that the correct result is unblinded only after the fault detection mechanism, in order to provide protection against possible bypassing (by injecting a second fault) of the fault detection countermeasure.

```
Algorithm 2. LRA PA-FA Blinded MPL algorithm
Input: EC base point $V$, random point $R \in EC(GF(p))$, $e = (e_{t-1}, e_{t-2}, ...e_0)$
1. Choose random initial base permutation $\gamma_t$. Transform V, R to RNS format using $\gamma_t$ permutation
2. $R_0 = R$, $R_1 = R + V$, $R_2 = -R$
3. $CMF(R_0, R_1, R_2, B\acute{}_{n,\gamma_t}, B_{n,\gamma_t})$
4. For $i = t-1$ to $0$
   (a) $R_2 = 2R_2$, always performed in initial permutation $\gamma_t$
   (b) choose a random base permutation $\gamma_i$
   (c) $RBP(R_0, B_{n,\gamma_{i+1}}, \acute{B}_{n,\gamma_{i+1}}, B_{n,\gamma_i}, \acute{B}_{n,\gamma_i})$
   (d) $RBP(R_1, B_{n,\gamma_{i+1}}, \acute{B}_{n,\gamma_{i+1}}, B_{n,\gamma_i}, \acute{B}_{n,\gamma_i})$
   (e) if $e_i = 1$
         $R_0 = R_0 + R_1$ and $R_1 = 2R_1$ in permutation $\gamma_i$
       else
         $R_1 = R_0 + R_1$ and $R_0 = 2R_0$ in permutation $\gamma_i$
       end if
5. $RBP(V, B_{n,\gamma_t}, \acute{B}_{n,\gamma_t}, B_{n,\gamma_0}, \acute{B}_{n,\gamma_0})$
6. If ($i$ and $e$ are not modified and $R_0 + V = R_1$)
   then
     (a) $RBP(R_0, B_{n,\gamma_0}, \acute{B}_{n,\gamma_0}, B_{n,\gamma_t}, \acute{B}_{n,\gamma_t})$
     (b) return $R_0 + R_2$
   else return error
```

All EC points in Algorithm 2 are represented in projective coordinates. Conversion to Montgomery Format (CMF) operation is used for transforming all EC point coordinates into the Montgomery format, so that RNSMM can be performed correctly. This conversion will require 6 RNSMMs. The RBP function performs base transformation from base permutation γ to permutation $\acute{\gamma}$ and requires 6 RNSMMs. The RBP function is executed in each MPL round once for point R_0 and once for R_1. As it can be observed from Algorithm 2, we do not perform RBP for the R_2 point doubling since this operation already includes computations only of a random point (it remains random during the whole scalar multiplication without any interference). Note that R_2 computations retain the same base permutation γ_t in all MPL rounds since R_2 point doubling involves only the random EC point R (no need to re-randomize it through RBP). However, since V is used in the fault detection mechanism and R_2 is needed for unblinding the correct result, after the last MPL round, there is a base transformation from the initial permutation γ_t to the last round's permutation γ_0 for V and there is also a base transformation from the last round's permutation γ_0 to the initial permutation γ_t that is done after passing successful fault detection.

IV. Implementation

In order to implement the above algorithms, a consistent realization process was followed, based on two steps. Taking into account that a considerable number of parameters are constant for all scalar multiplication operations on a specific EC (they are related to the Bases' moduli m_i, the moduli number n and the p value of the GF(p) field defining the EC), these values can be precomputed and stored in memory units so as to be used repeatedly for all scalar multiplications. Therefore, as a first step of realizing the proposed approach, an appropriate design methodology needs to be conceived in order to precompute and store the above mentioned values in memory space with efficiency. This step needs to be executed only once for all EC computations, so it can be considered as an initialization step. The second step in the previous sections' algorithm realization is the actual scalar multiplication design that needs to use the precomputation structure realized in the first step. To provide precomputations for all possible bases moduli combinations (realizing the base permutation γ), a numeric index (denoted as permutation index) is assigned to each such combination and a structure is associated to this index. This permutation structure includes the following information:

- The permutation index γ
- The n moduli that constitute base $B_{n,\gamma}$
- The n moduli that constitute base $B\acute{}_{n,\gamma}$
- The current Base $B_{n,\gamma}$ dynamic range M_B

There exist $\binom{2n}{n}$ different permutation structures that are stored in array form.

As an outcome of the first step, an entry is created as a memory structure for each moduli of the 2 RNS Bases (needed in the RNSMM algorithm). Such information (for a single entry i) are the following:

- the moduli value m_i

TABLE I

EDWARDS CURVE ($p = 2^{192} - 2^{64} - 1$) CASE STUDY EMPLOYED 8 BASE MODULI

$\mathbf{m_1}$	$2^{50}-2^{20}-1$	$\mathbf{m_5}$	2^{50}
$\mathbf{m_2}$	$2^{50}-2^{22}-1$	$\mathbf{m_6}$	$2^{50}-1$
$\mathbf{m_3}$	$2^{50}-2^{18}-1$	$\mathbf{m_7}$	$2^{51}-1$
$\mathbf{m_4}$	$2^{50}-2^{10}-1$	$\mathbf{m_8}$	$2^{49}-1$

- the $p\ mod m_i$ value
- the $\langle -p^{-1} \rangle_{m_i}$ value
- A matrix of $2n$ elements calculating $\langle m_j^{-1} \rangle_{m_i}$ for all $j \in \{0, 1, 2n-1\}$
- A matrix of $\binom{2n}{n}$ elements calculating $\left\langle M_{B_{n,\gamma}}^{-1} \right\rangle_{m_i}$ for all $\gamma \in \{0, 1, \frac{2n!}{n!n!} - 1\}$

As a proof-of-concept implementation of the above described two step design process, appropriate software code was written for ARM cortex A class processors (having a Raspberry Pi 2 as a reference design) using the GMP library for all $GF(p)$ operations. As a case study, $n = 4$ was used for an Edwards based Elliptic Curve (with $c = 1$ and $d = 2$) defined over GF(p) where $p = 2^{192} - 2^{64} - 1$ (NIST prime field). The employed case study moduli are presented in Table I.

Since $n = 4$ there exist 70 different base permutations which as an individual SCA countermeasure, introduce small randomization. However, since this randomization is combined with the base point blinding technique, the overall randomization of the implementation leakage trace is adequate to provide side-channel attack resistance. This can be observed from the collected traces using Electromagnetic Emission probes from the Raspberry Pi's processor during the implemented scalar multiplication execution. More precisely, our experimental setup is the following:

- Raspberry Pi 2 Model 8 with a 900MHz quad-core Cortex A7 processor.
- EMV Langer probe RF-U 5-2.
- Lecroy Waverunner 610Zi samping at 25MS/sec.

The traces are presented in Figure 2. As a preliminary analysis, we obtained traces with different random points and we observed the same patterns. High regularity in the leakage traces prohibits simple side-channel attacks. A more detailed analysis of the leakage traces containing several statistical tests and profiling of the device will be performed, in order to evaluate completely our implementation.

V. CONCLUSIONS

This paper presented a highly regular implementation of RNS scalar multiplication algorithm. Our combined countermeasure consisting of randomization of the EC base point and random permutation of the base moduli should provide resistance against the most common PA and FA attacks. Indeed, a preliminary analysis of our traces shows high regularity in the leakage traces. As future work, we plan to evaluate our algorithm in more sophisticated attack scenarios, in order to prove in practice our theoretical security analysis.

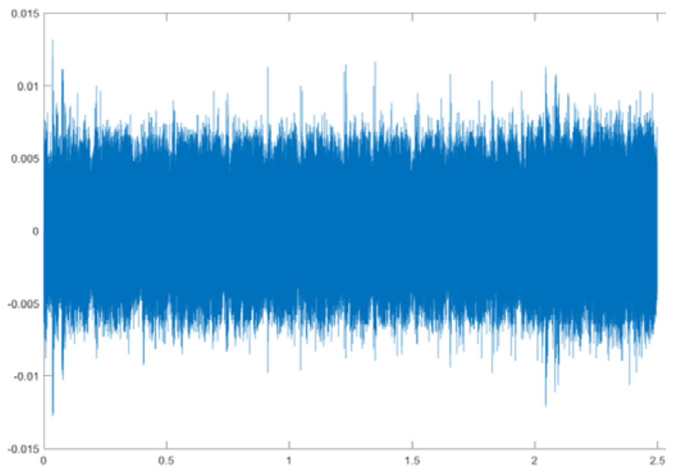

Fig. 2. proposed Scalar Multiplication EM leakage trace on Raspberry Pi 2 ARM processor

ACKNOWLEDGMENT

This work is supported by EU COST IC1204 TRUDEVICE, through the Short Term Scientific Mission of L. Papachristodoulou to the Computer Engineering & Informatics Dept, University of Patras, Greece, Winter 2016.

REFERENCES

[1] J. Bajard, J. Eynard, and N. Merkiche. Multi-fault attack detection for RNS cryptographic architecture. In *23nd IEEE Symposium on Computer Arithmetic, ARITH 2016, Silicon Valley, CA, USA, July 10-13, 2016*, pages 16–23, 2016.

[2] J. Bajard, L. Imbert, P. Liardet, and Y. Teglia. Leak resistant arithmetic. *CHES*, 3156, 2004.

[3] J.-C. Bajard, L.-S. Didier, and P. Kornerup. An RNS Montgomery modular multiplication algorithm. In *Proc.13th IEEE Symp. on Comp. Arithmetic*, pages 234–239. IEEE Comput. Soc, 1997.

[4] J.-C. Bajard, J. Eynard, and F. Gandino. Fault Detection in RNS Montgomery Modular Multiplication. In *IEEE 21st Symp. on Comp. Arithmetic*, pages 119–126. IEEE, Apr. 2013.

[5] J. C. Bajard, M. Kaihara, and T. Plantard. Selected RNS Bases for Modular Multiplication. In *2009 19th IEEE Symp. on Comp. Arithmetic*, pages 25–32. IEEE, June 2009.

[6] K. Bigou and A. Tisserand. Single base modular multiplication for efficient hardware RNS implementations of ECC. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 123–140, 2015.

[7] M. Esmaeildoust, D. Schinianakis, H. Javashi, T. Stouraitis, and K. Navi. Efficient RNS Implementation of Elliptic Curve Point Multiplication Over GF(p). *IEEE Trans. on VLSI Systems*, 21(8):1545–1549, Aug. 2013.

[8] J. Fan and I. Verbauwhede. An updated survey on secure ECC implementations: Attacks, countermeasures and cost. *Cryptography and Security: From Theory to Applications*, 6805:265–282, Jan. 2012.

[9] A. P. Fournaris, N. Klaoudatos, N. Sklavos, and C. Koulamas. Fault and power analysis attack resistant RNS based edwards curve point multiplication. In *Proceedings of the 2nd Workshop on Cryptography and Security in Computing Systems, CS2 at HiPEAC 2015, Amsterdam, Netherlands, January 19-21, 2015*, pages 43–46, 2015.

[10] A. P. Fournaris, L. Papachristodoulou, L. Batina, and N. Sklavos. Residue number system as a side channel and fault injection attack countermeasure in elliptic curve cryptography. In *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–4, April 2016.

[11] G. Fumaroli and D. Vigilant. Blinded fault resistant exponentiation. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *FDTC*, volume 4236 of *LNCS*, pages 62–70. Springer, 2006.

[12] C. Giraud. An rsa implementation resistant to fault attacks and to simple power analysis. *IEEE Trans. on Computers*, 55(9):1116–1120, 2006.

[13] N. Guillermin. A high speed coprocessor for elliptic curve scalar multiplications over Fp. *Lecture Notes in Computer Science Advances in Cryptology Cryptographic Hardware and Embedded Systems CHES 2010*, pages 48–64, 2010.

[14] N. Guillermin. A coprocessor for secure and high speed modular arithmetic. *IACR Cryptology ePrint Archive*, 2011.

[15] M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In *4th CHES 2003*, pages 291–302, UK, 2003. Springer-Verlag.

[16] S. Kawamura, M. Koike, F. Sano, and A. Shimbo. Cox-rower architecture for fast parallel montgomery multiplication. In *Advances in Cryptology EUROCRYPT*, 2000.

[17] G. Perin and L. Imbert. Electromagnetic analysis on RSA algorithm based on RNS. In *2013 Euromicro Conference on Digital System Design (DSD)*, number 1, pages 345–352, 2013.

[18] D. Schinianakis and T. Stouraitis. Hardware-fault attack handling in RNS-based Montgomery multipliers. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pages 3042–3045. IEEE, May 2013.