Secure and Efficient RNS software implementation for Elliptic Curve Cryptography

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

Louiza Papachristodoulou
Digital Security Group,
Radboud University Nijmegen,
The Netherlands
Email: louizap@cs.ru.nl

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

***Abstract*—Elliptic Curve Cryptography operations rely heavily on the strong security of scalar multiplication. However, this operation is vulnerable to side channel (SCA) and fault injection (FA) attacks. The use of alternative arithmetic systems like Residue Number System (RNS) for all scalar multiplication underline operations has been proposed as an efficient countermeasure approach for the above attacks. 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 additional RNS arithmetic manipulations or more traditional algorithmic countermeasures. In this paper, extending our previous work, we explore the potentials of RNS as an SCA and FA countermeasure. A description of RNS based SCA and FA resistance means is provided through appropriate scalar multiplication algorithmic variations, traces of the proposed algorithm are collected and the results are analyzed regarding the RNS countermeasure strength. More specifically, in this paper, a secure RNS based Montgomery Power Ladder based scalar multiplication algorithm is provided and is implemented on an ARM Cortex A7 processor. The implementation SCA-FA resistance is evaluated by collecting preliminary leakage trace results that validate our initial assumptions.**

I. Introduction

In the RNS arithmetic representation, a number is represented by a given moduli base (RNS base) consisting of several base elements that can be processed in parallel (parallel arithmetic calculations advantage). The system can also be used to represent elements of cyclic groups or finite fields and thus can be applied to Elliptic Curves (that use finite fields). The parallelism that RNS arithmetic provides, constitutes a good cause for adopting this system for finite field operations (and consecutively Elliptic Curve operations). The parallel processing of any type of finite field operation outcome apart from speed may also increase the power consumption trace complexity, thus offering a basic Side Channel Attack (SCA) resistance. Taking also into account that a single bit fault in a moduli can lead to "difficult to trace" changes in an overall finite field element, supports the argument that the RNS can be introduced as SCA and Fault injection Attack (FA) countermeasure in cryptosystems [2] [4] [15] [21] [10] [11].

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 SCA countermeasure as well as a FA 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 [11].

In this paper, we explore the RNS potentials in elliptic curve scalar multiplication, extending the work done in [11], taking into account the arithmetic system's benefits and combining it with more traditional SCA/FA countermeasures. 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 presented and its importance as the basis of a secure RNS scalar multiplication implementation is described. Current approaches on designing RNS Montgomery multiplication are also presented in an effort to support our argument that the algorithm can be used as a basis for SCA/FA resistance in scalar multiplication. The approach described in [11], in this paper, is expanded and analyzed in detail so as to provided further justification to the above described cause, thus detailing the description of a scalar multiplication algorithm variant capable of offering 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 a good trade-off between efficiency and SCA/FA resistance yet fully retaining 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. In this paper, as opposed to [11] where only a theoretical security analysis is present, the correctness of the proposed solution is verified and its security is tested, by implementing 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. Electromagnetic leakage traces were also collected while a scalar multiplication was running with and without our proposed SCA/FA countermeasure approach and after analysis, several interesting conclusions were extracted proving the proposed approach quality.

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 and some preliminary measurements, results of our SCA/FA analysis are presented in section V. Finally, section VI 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 over moduli 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).

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

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:

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

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] [11].

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 $S_B = A \cdot B \cdot M_B^{-1} \ mod \ p$ and $S_{\acute{B}} = A \cdot B \cdot M_B^{-1} \ mod \ p$. Base extension from one base to the other in Algorithm 1 is needed since M_B^{-1} cannot be computed directly for base B_n (M is multiple of the base elements) and therefore computations must be migrated to the $\acute{B_n}$ base to come up with S_B.

Two main approaches to base extension are used in practice for RNS arithmetic: the MRS system and the Cox-Rower architecture introduced in [17]. The Cox-Rower architecture consists of parallel arithmetic units, the Rowers, which perform the independent computations for each base concurrently, and the Cox unit dedicated to the computation of an approximation of the correction factor w. Therefore, it can be efficiently implemented in hardware. An interesting work

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 $S_{\acute{B}}$

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.

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

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 $M_{B\cup\acute{B}}\ mod\ P$ 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 [8] [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 [7] 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 countermeasure. 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 as long as the number of moduli is high. 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 $RNSMM(A_1, 1, P, \acute{B_{n,\gamma}}, B_{n,\gamma})$

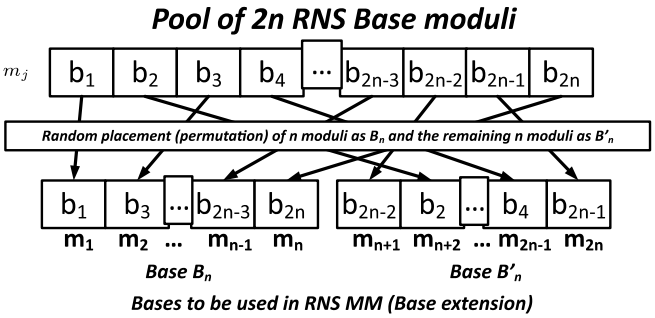

Fig. 1. Leak Resistant Arithmetic approach on Base Randomization

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 [15] [14], 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

[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

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] [20]. 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 $B_n \cup m_R$ (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 $\left(\text{i.e. } \langle S_{B \cup m_r} \rangle_{m_r} \text{ and } \left\langle S_{\acute{B} \cup m_r} \right\rangle_{m_r}\right)$. 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 [15] 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] [11] is generic and can be applied to any base extension methodology.

III. FA and SCA Resistant Scalar Multiplication

Given the description of RNS SCA and FA countermeasures, we adopt the inclusion of LRA as an add-on countermeasure in an SCA 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 [12] [10] and [11]. 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 [9]. 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 [16] and by Giraud in [13], 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. The added randomness before the beginning of scalar multiplication helps in this second fault protection approach since if an error occurs during scalar multiplication this will affect also the random number used during this operation (it will be different than the originally used value) so the final outcome $R_0 + R_2$ won't reveal the faulty scalar multiplication output but rather a random number (the faulty random number won't be R^{2t} thus won't be eliminated when adding $R_2 = -R^{2t}$ to R_0), as is argued in [18] for a similar case (for RSA).

By exploiting the MPL intrinsic characteristic (a traditional fault detection method) there is no need for an additional, redundant, RNS base modulo (m_R) in order to introduce a potent fault detection mechanism in the proposed SCA/FA countermeasure approach. Using such modulo in every RNSMM will be introducing additional computation effort (efficiency reduction) to the RNSMM base extension operation and additional memory requirements for precomputations. This additional cost in every RNSMM of every Scalar multiplication round is exchanged with the operations done in step 6 of Algorithm 2.

Algorithm 2. LRA SCA-FA Blinded MPL algorithm
Input: EC base point V, random point $R \in EC(GF(p))$, $e = (e_{t-1}, e_{t-2}, ...e_0)$
Ouput: $e \cdot V$ or random value (in case of faults)
1. Choose random initial base permutation γ_t. Transform V, R to RNS format using γ_t permutation
2. $R_0 = R$, $R_1 = R + V$, $R_2 = -R$
3. $CMF(R_0, R_1, R_2, \acute{B_{n,\gamma_t}}, B_{n,\gamma_t})$
4. **For** $i = t - 1$ **to** 0
   (a) $R_2 = 2R_2$, always performed in initial permutation γ_t
   (b) choose a random base permutation γ_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 γ_i
   else
   $R_1 = R_0 + R_1$ and $R_0 = 2R_0$ in permutation γ_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 7 RNSMMs as shown in Algorithm 3. The RBP function performs base transformation from base permutation γ to permutation $\acute{\gamma}$, as described in Algorithm 4, and requires 12 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.

Algorithm 3. Point Coordinate Montgomery Form computation
$CMF(P_0, P_1, P_2, B_n, \acute{B_n})$
Input: $P_0 : (X_0, Y_0, Z_0)$, $P_1 : (X_1, Y_1, Z_1)$,$P_2 : (X_2, Y_2, Z_2)$ EC point $\in EC(GF(p))$
Output: $P_0,\ P_1,\ P_2$: EC point $\in EC(GF(p))$ under Montgomery form
$X_0 = RNSMM(X_0, M_{B\cup\acute{B}}\ modP_{B\cup\acute{B}}, P, B_n, \acute{B_n})$
$Y_0 = RNSMM(Y_0, M_{B\cup\acute{B}}\ modP_{B\cup\acute{B}}, P, B_n, \acute{B_n})$
$Z_0 = RNSMM(Z_0, M_{B\cup\acute{B}}\ modP_{B\cup\acute{B}}, P, B_n, \acute{B_n})$
$X_1 = RNSMM(X_1, M_{B\cup\acute{B}}\ modP_{B\cup\acute{B}}, P, B_n, \acute{B_n})$
$Y_1 = RNSMM(Y_1, M_{B\cup\acute{B}}\ modP_{B\cup\acute{B}}, P, B_n, \acute{B_n})$
$Z_1 = RNSMM(Z_1, M_{B\cup\acute{B}}\ modP_{B\cup\acute{B}}, P, B_n, \acute{B_n})$
$X_2 = RNSMM((-X_0, M_{B\cup\acute{B}}\ modP_{B\cup\acute{B}}, P, B_n, \acute{B_n})$
$Y_2 = Y_0$
$Z_2 = Z_0$
Return (X_0, Y_0, Z_0), (X_1, Y_1, Z_1),(X_2, Y_2, Z_2)

Algorithm 4. Random Base Permutation
$RBP(P_0, B_{n,\gamma}, \acute{B}_{n,\gamma}, B_{n,\acute{\gamma}}, \acute{B}_{n,\acute{\gamma}})$
Input: $P_0 : (X_0, Y_0, Z_0)$ EC point $\in EC(GF(p))$ in RNS bases $B_{n,\gamma}, \acute{B}_{n,\gamma}$
Output: $P_0 : (X_0, Y_0, Z_0)$ EC point $\in EC(GF(p))$ in RNS bases $B_{n,\acute{\gamma}}, \acute{B}_{n,\acute{\gamma}}$
$A_1 = RNSMM(X_0, M_{B\cup\acute{B}}\ mod\ P_{B\cup\acute{B}}, P, \acute{B}_{n,\acute{\gamma}}, B_{n,\acute{\gamma}})$
$X_0 = RNSMM(A_1, 1, P, \acute{B}_{n,\gamma}, B_{n,\gamma})$
$A_1 = RNSMM(Y_0, M_{B\cup\acute{B}}\ mod\ P_{B\cup\acute{B}}, P, \acute{B}_{n,\acute{\gamma}}, B_{n,\acute{\gamma}})$
$Y_0 = RNSMM(A_1, 1, P, \acute{B}_{n,\gamma}, B_{n,\gamma})$
$A_1 = RNSMM(Z_0, M_{B\cup\acute{B}}\ mod\ P_{B\cup\acute{B}}, P, \acute{B}_{n,\acute{\gamma}}, B_{n,\acute{\gamma}})$
$Z_0 = RNSMM(A_1, 1, P, \acute{B}_{n,\gamma}, B_{n,\gamma})$
Return (X_0, Y_0, Z_0)

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 $B_{n,\acute{\gamma}}$
- The n moduli that constitute base $B_{n,\gamma}$
- 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
- the $p\ mod m_i$ value
- the $\left\langle -p^{-1} \right\rangle_{m_i}$ value
- A matrix of $2n$ elements calculating $\left\langle m_j^{-1} \right\rangle_{m_i}$ 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 $\gamma \in \{0, 1, \frac{2n!}{n!n!} - 1\}$

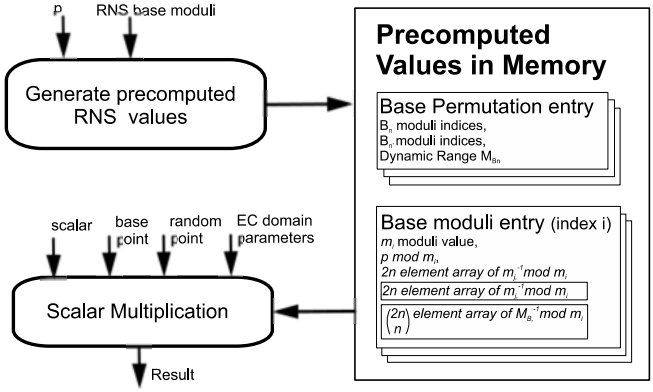

Fig. 2. Overview of the employed implementation approach

V. Case Study and Security Analysis

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. Following the approach of [8] and [5] a 4 moduli RNS bases ($n = 4$) RNS realization was used that have the form $B_n : (2^k - 2^{t_1} - 1, 2^k - 2^{t_2} - 1, 2^k - 2^{t_3} - 1, 2^k - 2^{t_4} - 1)$ and $\acute{B_n} : (2^k, 2^k - 1, 2^{k+1} - 1, 2^{k-1} - 1)$. For the above approach an Edwards EC was adopted with $c = 1$ and $d = 2$ defined over GF(p) (twisted EC) where $p = 2^{192} - 2^{64} - 1$ (NIST prime field) (i.e 192 bit length GF(p) numbers). Edwards Curves were chosen instead of Weierstrass ones since the first have complete formulas thus offering

better SCA resistance and have never been tested under the RNS arithmetic framework. However, the twisted Edwards curve shape was used and not the equivalent Montgomery form that can be combined with MPL, in order to keep the solution generic enough so as to be somewhat usable for other EC types. To retain compatibility with NIST Curves and implementations of other similar works [8], the prime field was left to be $p = 192$ (although the implementation can be easily adapted to any Edwards Curve including the popular Curve 25519). For $GF(192)$, we employ RNS bases that have a 200 bit dynamic range consisting of four approximately 50 bit moduli ($k = 50$) for each involved RNS base B_n and $\acute{B}_n$. The employed case study moduli are presented in Table 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}
$\mathbf{m_2}$	$2^{50}-2^{22}-1$	$\mathbf{m_6}$	$2^{50}-1$
$\mathbf{m_3}$	$2^{50}-2^{18}-1$	$\mathbf{m_7}$	$2^{51}-1$
$\mathbf{m_4}$	$2^{50}-2^{10}-1$	$\mathbf{m_8}$	$2^{49}-1$

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 SCA 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 600MHz quad-core Cortex A7 processor.
- EMV Langer probe RF-U 5-2.
- Lecroy Waverunner 610Zi sampling at 2.5GS/sec.

The rationale behind choosing Raspberry Pi as our target is twofold. Firstly, it is always interesting to attack a real-world, original target, that has not dedicated hardware for SCA. Secondly, a fast multi-core processor is necessary to handle the required workload for RNS base arithmetic.

A. *Profiling of the device*

In this subsection, we give some technical details of the acquisition and the steps towards profiling of the device. The experimental setup for our analysis is quite challenging, since the Raspberry Pi is not a common target to perform SCA. The high levels of noise combined with the high frequency of the processor makes it hard to establish a stable setup and collect the whole computation with a high-end oscilloscope. Our Pi runs at 600MHz [2], which means that from the Nyquist principle, the sampling rate should be at least 1.2GS/sec. Recent studies [19], suggest that the sampling rate is sufficient to be twice the frequency that leaks information, which is not the same as the clock frequency in modern microprocessors; the leakage frequency is usually smaller. However, we do not obtain the required level of detail to perform SCA analysis even by using the textbook rule of Nyquist. Therefore, the sampling rate is increased to 2.5GS/sec in our experiments. This is the highest rate for which some useful operations can be recorded, i.e. the first two rounds of the scalar multiplication. More precisely, we can collect 20M samples, which corresponds to 8 msec. of the scalar multiplication.

Figure 3 shows traces of 20M samples in the case of simple RNS (blue) and our proposed implementation (green). In the second case, both base permutation randomization and randomized input point are activated as countermeasures.

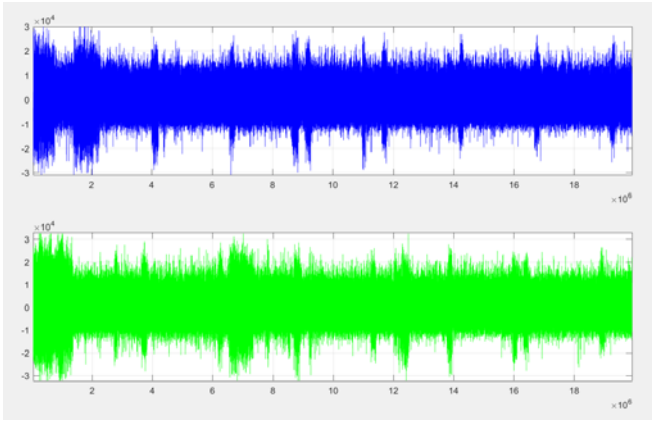

Fig. 3. Traces with no randomization of point or base (blue) and with the proposed randomization countermeasures (green).

In the traces, we can identify the initialization and loading of the RNS bases in the beginning of the scalar multiplication ($0 - 2.2 \times 10^6$ in the first case and $0 - 7.2 \times 10^6$ in the second case), followed by constant time point doublings and point additions. In the non-randomized case, the whole scalar multiplication lasts 0.4572 sec. Each iteration consists of one point doubling and one point addition and lasts about 2 msec. Taking into account the limitations from the oscilloscope, we can record up to 12 msec. of the scalar multiplication, which corresponds to 4-5 rounds. In the case where both randomization of the base permutation and the input point is activated, there are three point operations instead of two, as explained in our proposed Algorithm 2. Scalar multiplication lasts 0.8677 sec., which is almost twice longer as the unprotected implementation. Some extra time needed for initialization of the basis and choosing the base moduli, results in making only 2 rounds of the scalar multiplication visible in our traces.

By performing auto-correlation of a trace with itself, we can identify each operation (point doubling and point addition) and its duration. For instance, in the non-randomized case, each operation takes 2246000 samples, which corresponds to 0.8984 msec. Figure 4 depicts the results of auto-correlation

[2]The given frequency from the specifications of model 2B is 900MHz, but our target runs at 600MHz; if that was not the case, we would need to underclock the CPU, in order to be able to use the given oscilloscope.

of (aligned and averaged) traces for both non-randomized and randomized cases.

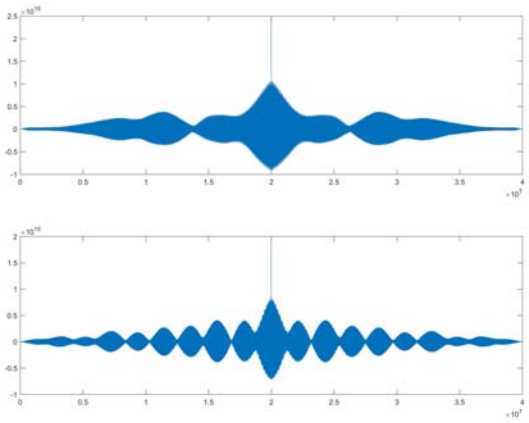

Fig. 4. Auto-correlation for non-randomized and randomized traces with itself

At this point, it is worth noticing that our implementation is constant time, in the sense that different point operations take exactly the same time to be completed. There is however a slight difference in the peaks that correspond to point doubling or point addition, which is an interesting remark from a side-channel point of view.

B. SCA resistance

As a first step towards evaluation of our proposed implementation, we compared an implementation with the combined countermeasures of randomizing the input point to the scalar multiplication and randomized base permutations, and a simple RNS implementation of MPL, as shown in figure 3. The only SCA protection of the simple implementation is the regularity of MPL and the inherent fault injection resistance of RNS. As it is shown by the EM attack of Perin et al. in [20], the RNS implementation of MPL is vulnerable to the relative-doubling attack [22]. With the following attack scenario, we show that our proposed implementation is resistant against the relative-doubling attack and against the Online Template Attacks (OTA) introduced in [6], which is more powerful attack in the sense that it can be applied to both left-to-right and right-to-left scalar multiplication algorithms.

We take two traces from each algorithm, one with input point P and one with input point $2P$. Then, we choose as template the beginning of the trace, where $2P$ is given as input and we perform auto-correlation of this template with the traces that had input P. A distinguished high peak is at the position that our template is chosen in the original trace, as expected. However, when our proposed algorithm is used, the result of auto-correlation does not reveal any information about the beginning of the second iteration, since there is no distinguished peak, as shown in figure 5. In the case, where the simple algorithm was used, the result of the auto-correlation is similar to our algorithm. Therefore, in both cases, where RNS is used, OTA cannot be applied with a single trace.

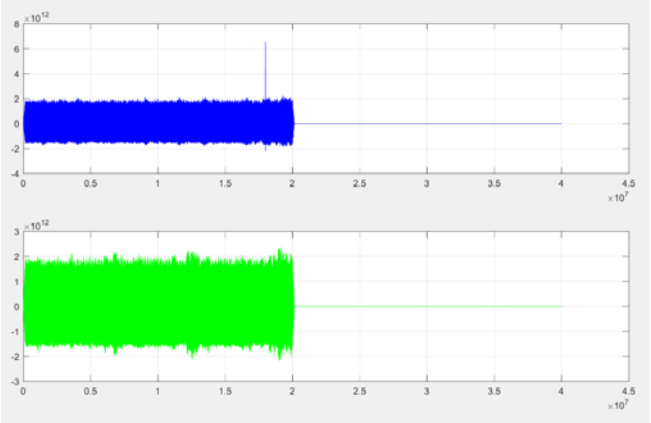

Fig. 5. Correlation 2P-P for the trace with input $2P$ (blue) and for the trace with input P when all randomizations are on (green).

This first analysis shows that our proposed algorithm can indeed provide high levels of security by resisting some types of correlation and template attacks. However, a more detailed (differential) analysis of the leakage traces containing several statistical tests will be performed, in order to evaluate completely our implementation. The high noise levels of the Raspberry Pi may lead us to choosing a different platform to perform other types of attacks.

VI. Conclusions

This paper presented a highly regular implementation of RNS scalar multiplication algorithm for an ARM-A processor. This is the first software implementation using RNS, aiming to provide the security benefits of RNS in a reasonable running time. Our combined countermeasure consisting of randomization of the EC base point and random permutation of the base moduli provides resistance against the most common PA and FA attacks. Indeed, a preliminary analysis of our traces shows high regularity in the leakage traces. Moreover, our proposed implementation can resist the relative-doubling and OTA types of attacks, while a simple RNS implementation with MPL is shown in previous works to be vulnerable to the relative-doubling attack. As future work, we plan to evaluate our algorithm in more sophisticated attack scenarios, in order to prove in practice our theoretical security analysis.

Acknowledgment

This work is supported by EU COST IC1204 TRUDEVICE, through the Short Term Scientific Mission of L. Papachristodoulou to the 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] L. Batina, L. Chmielewski, L. Papachristodoulou, P. Schwabe, and M. Tunstall. Online template attacks. In *proc. of 15th International Conference on Cryptology in India INDOCRYPT 2014, New Delhi, India, Dec. 14-17, 2014*, pages 21–36, 2014.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

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

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

[18] C. Kim and J. Quisquater. Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures. *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, 4462:215–228, 2007.

[19] J. Longo, E. De Mulder, D. Page, and M. Tunstall. *SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip*, pages 620–640. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[20] 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.

[21] 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.

[22] S. Yen, L. Ko, S. Moon, and J. Ha. Relative Doubling Attack against Montgomery Ladder. *Information Security and Cryptology-ICISC*, pages 117–128, 2006.