A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$

Nicolas Guillermin[1,2]

[1] DGA Information Superiority, Bruz, France
[2] IRMAR, Université Rennes 1, France

Abstract. We present a new hardware architecture to compute scalar multiplications in the group of rational points of elliptic curves defined over a prime field. We have made an implementation on Altera FPGA family for some elliptic curves defined over randomly chosen ground fields offering classic cryptographic security level. Our implementations show that our architecture is the fastest among the public designs to compute scalar multiplication for elliptic curves defined over a general prime ground field. Our design is based upon the Residue Number System, guaranteeing carry-free arithmetic and easy parallelism. It is SPA resistant and DPA capable.

Keywords: elliptic curve, high speed, RNS, prime field, FPGA

1 Introduction

Twenty five years after their introduction for cryptographic applications [14], elliptic curves are well established in the field of public key cryptography. A standard of the National Institute of technology (NIST) recommends their use for digital signature [17]. The most time consuming operation in elliptic curve based protocols is the scalar multiplication. As a consequence, scalar multiplication has attracted a lot of attention in public literature. Available designs may differ greatly depending on the target implementation (GPGPUs, CPUs, ASICs, FPGAs) and the aim they try to achieve which may be related to speed, size, power consumption or security issues. We refer the reader to [3, 25, 10, 21] for example of known implementations.

In this paper, we describe the fastest available architecture for computing $[k]G$ over curves defined over $\mathbb{F}_p$ for general prime p in FPGA. Our architecture is based on Residue Number Systems (RNS) and is resistant against side channel attacks. We have made a FPGA implementation of our design. Actually, FPGA implementations are particularly interesting for at least two reasons : they are well suited to provide a good local protection level, and they constitute generally the first step towards faster ASIC implementations.

Target application of such special purpose designs are all the fields where both high speed, low latency and high level resistance against attacks are required (example : IPSEC set-top box).

Related work : A great overview of high speed hardware accelerator for ECC is given by [13]. Designs can be split in two categories : those which support elliptic curves over $\mathbb{F}_{2^n}$, and those over $\mathbb{F}_p$. Architectures of the first group give the best speed to security ratio. It is mostly due to the field structure (No carry is propagated). State of the art implementations show a latency under 20 μs for a 2^{80} security [8]. Nevertheless large characteristic remain interesting, mostly because $\mathbb{F}_p$ offers less structure than $\mathbb{F}_{2^n}$, and may be safer. Some architecture can also support both field types [22].

Some implementations are specific to a pseudo-Mersenne prime [7]. These implementations may be faster than the one which do not depend on the relying field. Nevertheless the ability of changing the curve is also an asset for security (finding weak curves is still an active research area). Our architecture is of this kind. To our knowledge the best architecture is the one of

Mentens [11], which computes a 1 ms 160 bit scalar multiplication on a Xilinx Virtex 2 pro. Most of the implementations are based on a multi-precision Montgomery representations of numbers, allowing reduction without expensive divisions.

Another way to represent big numbers is the Residue Number System. It provides fast and carry free arithmetic. A modified version of the Montgomery algorithm [1] makes it suitable for arithmetic in $\mathbb{F}_p$. Szerwinski et al [25] used it to produce the fastest software implementation of scalar multiplication. Kawamura et al [10, 16] proposed a very efficient architecture for RSA signature on an ASIC. Their contribution is analysed in section 3, since it is the starting point of our work.

The higher p is, the more efficient RNS is (because its high parallelization ability). Then we could think that applying RNS to ECC will be less interesting than RSA. We show in this paper that this drawback is compensated by 2 advantages. First, the RNS ability to execute patterns like $AB + CD$ in only one reduction, while both products are almost free, reduces the time of point operations in ECC while it is useless for RSA (2.2). Second, ECC operations are parallelizable, therefore we can deepen the pipelines while keeping a high rate occupation (3.2).

Our contribution : We present a complete redefinition of main module of Kawamura. Thanks to it we can use elliptic curve and we reach high speed on a FPGA. We design the first architecture to break 1 ms for 160 bit elliptic curve scalar multiplication over prime field of any characteristic, even on a 130 nm node FPGA (the Altera Stratix family in our case). Our scalable architecture keeps its advantage even for larger groups (up to 512 bits). We also propose an algorithm for RNS-Radix transformation that does not cost a single gate, and base choice considerations for RNS use with elliptic curves.

Structure of the paper : The section 2 deals with mathematical backgrounds of RNS and elliptic curves. The section 3 describes and analyses the choices that are made to improve Kawamura's architecture, and and the section 4 gives the results of implementations, and compares it to other existing design. Design schemes are at the end of the paper.

2 Mathematical background

Notations : In all the paper, for $a, b \in \mathbb{N}$, we denote by $|a|_b$ the result of a modulo b.

2.1 RNS

Overview : Let $B = \{m_1, \cdots, m_n\}$ be a set of co-prime natural integers, and $M = \prod_{i=1}^{n} m_i$. The residue number system (RNS) representation $\{X\}_B$ of $X \in \mathbb{N}$ such that $0 \le X < M$ is the unique set of positive integers $\{x_1, \cdots, x_n\}$ with $x_i = |X|_{m_i}$. This representation allows fast arithmetic in $\mathbb{Z}/M\mathbb{Z}$ since

$$\{X \odot Y\}_B = \{|x_1 \odot y_1|_{m_1}, \cdots, |x_n \odot y_n|_{m_n}\} \tag{1}$$

for $\odot \in (+, -, \times, /)$, $/$ being only available for Y coprime with M. The integer X is recovered thanks to the Chinese remainder theorem :

$$X = \left| \sum_{i=1}^{n} |x_i \times M_i^{-1}|_{m_i} \times M_i \right|_M \text{ where } M_i = \frac{M}{m_i}. \tag{2}$$

Note that M_i^{-1} is then well defined in $\mathbb{Z}/m_i\mathbb{Z}$. In the rest of the paper, B is called a RNS base and the $\{m_i\}_{i=1,\ldots,n}$ are called channels of B, since every calculation are done independently modulo these channels.

Algorithm 1 $Red_{Montg}(X, p, B, \tilde{B})$

```
Require: $B$ and $\tilde{B}$ RNS bases with $M > \alpha p$ and $\tilde{M} > 2p$
Require: $p$ co-prime with $M$ and $\tilde{M}$
Require: $\{X\}_B$ and $\{X\}_{\tilde{B}}$ RNS representation of $X < \alpha p^2$ in $B$ and $\tilde{B}$
Require: precalculations : $\{|-p^{-1}|_M\}_B$,$\{|M^{-1}|_{\tilde{M}}\}_{\tilde{B}}$ and $\{p\}_{\tilde{B}}$
Require: algorithm $B_{ext}(A, B_1, B_2)$ computing $\{|A|_{M_2}\}_{B_2}$ from $\{A\}_{B_1}$
Ensure: $\{S\}_B$ and $\{S\}_{\tilde{B}}$ such that $|S|_p = |XM^{-1}|_p$ and $S < 2p$
1: $Q \leftarrow X \times |-p^{-1}|_M$ in $B$
2: $\tilde{Q} \leftarrow B_{ext}(Q, B, \tilde{B})$
3: $\tilde{R} \leftarrow X + \tilde{Q} \times p$ in $\tilde{B}$
4: $\tilde{S} \leftarrow \tilde{R} \times M^{-1}$ in $\tilde{B}$
5: $S \leftarrow B_{ext}(S, \tilde{B}, B)$
6: return $S$ and $\tilde{S}$
```

RNS Montgomery reduction algorithm : The Montgomery reduction application was first introduced in [15] for the purpose of multiprecision arithmetic. The paper [1] presents an adaptation in the context of RNS representation.

In the following we recall the main results of this last paper. Let p be a prime, $\alpha > 2$ an integer, B and $\tilde{B}$ be two RNS bases with their channel products M and $\tilde{M}$ such that $M > \alpha p$ and $\tilde{M} > 2p$. For all input $a < \alpha p^2$ given in B and $\tilde{B}$ the Montgomery algorithm computes S in B and $\tilde{B}$ such that $S < 2p$ and $|S|_p = |a \times M^{-1}|_p$. The factor M^{-1} is not a concern if a lot of computation in modular arithmetic are to be done in a row, which is the case in most applications related to cryptography. Actually, thanks to the use of Montgomery representative $\phi(X) = |XM|_p$ (see [15]), one only needs a transformation at the beginning of any calculation $\phi(X) = Red_{Montg}(X \times |M^2|_p, p, B, \tilde{B})$ and the corresponding invert transformation $\phi^{-1}(Y) = Red_{Montg}(Y, p, B, \tilde{B})$ at the end.

The base change $B_{ext}(X, B, \tilde{B})$ is due to the fact that one can not divide by M in B. The second base extension computes $\{S\}_B$ from $\{S\}_{\tilde{B}}$. Both are then available for another computation. Unlike the multiprecision Montgomery algorithm, we can not execute the final reduction, since it is not easy to know if S is more or less than p (comparison is a greedy operation in RNS representation). The result S is then kept between 0 and $2p$. The main consequence is that $\tilde{M}$ has to be up to $2p$. The choice of B depends on the maximal number we wish to reduce. Proposition 1 shows that $\tilde{M}$ does not need to be more than M. Even if $\tilde{Q}$ is not equal to Q but to $|Q|_{\tilde{M}}$, the algorithm still gives the correct output.

Proposition 1. *Given $\alpha > 2$, if $M > \alpha p$ and $\tilde{M} > 2p$ then $Red_{Montg}(X, p, B, \tilde{B})$ gives the correct output for every X between 0 and αp^2.*

Proof. Be $X < \alpha p^2$, $M > \alpha p$ and $\tilde{M} > 2p$. $Q = |XP^{-1}|_M$, therefore $Q < M$. By B_{ext} , $\tilde{Q} = |Q|_{\tilde{M}}$. As $X_{\tilde{B}} = |X|_{\tilde{M}}$ and $p < \tilde{M}$, $\tilde{S}$ is equal to $|(X + Qp)/M|_{\tilde{M}}$. As $Q < M$, $T = (X + Qp)/M < 2p$. Therefore $\tilde{S} = T$, and as $\alpha > 2$ $S = T$ too. As $T \equiv |XM^{-1}|_p$, we can conclude that S and $\tilde{S}$ are the expected results.

RNS base extension : The greediest steps of this algorithm are the two base extensions $B_{ext}(X, B, \tilde{B})$ and $B_{ext}(X, \tilde{B}, B)$. They are a classical $O(n^2)$ algorithm, where n is the RNS base size, and the elementary operation is a modular multiplication/addition on a channel. The main concern of every algorithm implementing B_{ext} is to provide a way to compute the final reduction by M, to calculate γ such that

$$X = \sum_{i=1}^{n} |x_i \times M_i^{-1}|_{m_i} \times M_i - \gamma M. \tag{3}$$

Once γ is calculated, X is easily recovered on every channel $\tilde{m}_i$ by multiplying and accumulating the result. Three different algorithm are proposed by literature :

Algorithm 2 $Montg - ladder(k, G, \mathbf{C}_{p,a_4,a_6})$

```
Require: $k \in \mathbb{N} = \sum k_i.2^i, G$ a point of $\mathbf{C}_{p,a_4,a_6}$
Ensure: $R = (k]G$
1: $R \leftarrow \mathcal{O}$ ; $S \leftarrow G$
2: for $i$ from $\log_2(k)$ to 0 do
3:    if $(k_i = 0)$ then
4:       $R \leftarrow 2R$ ; $S \leftarrow R + S$
5:    else
6:       $S \leftarrow 2S$ ; $R \leftarrow R + S$
7:    end if
8: end for
9: return $R$
```

– a Mixed Radix System (MRS) approach [2] which natively avoids final reduction, but is hard to implement in hardware because of the structure of the algorithm, but remains a good alternative in software,
– an extra modulus approach proposed by Shenoy and Kumaresan [24]. The idea of this algorithm is to have a m_e coprime with M and $\tilde{M}$, and to use $X[m_e]$ to compute γ. The main drawback of this approach is that we need to keep $|X|_{m_e}$ all along during the calculation, while we just need it during the base extension,
– a floating point approach proposed by Posch and Posch [19], and improved by Kawamura et al. [10]. The main idea is to transform the equation 3 as follow:

$$X = \sum_{i=1}^{n} \frac{M\xi_i}{m_i} - M\lfloor \sum_{i=1}^{n} \frac{\xi_i}{m_i} \rfloor \text{with } \xi_i = |x_i M_i^{-1}|_{m_i} \tag{4}$$

The main drawback of the floating point approach is a potential emergence of an offset due to the approximation while computing $\lfloor \sum_{i=1}^{n} \xi_i/m_i \rfloor$. In [10] ξ_i/m_i is approximated by $\xi_i/2^r$ where r is chosen as word depth in the proposed architecture. As m_i is chosen as a pseudo-Mersenne prime near 2^r, the offset of the calculation may be easily limited to 1/2. In [10] is explained how this possible error can be without consequences for the result, as soon as $\tilde{M} < 6p$. The output of Red_{Montg} will be less than $3p$, whatever the input is (proposition 1 is then easily adapted, with $\tilde{M} > 6p$ and $\alpha > 3$ the output of the algorithm 1 will be less than $3p$).

2.2 Elliptic curves

Overview : In this paper, considered elliptic curves $\mathbf{C}_{p,a_4,a_6}$ are seen as sets of couples $(x, y) \in \mathbb{F}_p^2$ verifying the following equation, with p prime and extra conditions on a_4 and a_6 which are not discussed here.

$$y^2 = x^3 + a_4 x + a_6 \tag{5}$$

Together with the point at infinity $\mathcal{O}$, $\mathbf{C}_{p,a_4,a_6}$ is an abelian group. The composition law has a geometric meaning described by the vertical and tangent. Some specific curve shapes (forms of the equation) spare multiplications and reduction while computing $P + Q$ and $2P$ over $\mathbf{C}_{p,a_4,a_6}$ [9, 5].Nevertheless the Weierstrass form represents all elliptic curves over prime fields (through isomorphism over $\mathbb{F}_p^2$). Other representations can only represent curves with subgroups (e.g order 4 for Edwards curves and Montgomery form, order 3 for Hessian curves...). Here we consider general curves in Weierstrass form, given by 5.

Addition and doubling formulæ : The Montgomery ladder [9] algorithm is a SPA-resistant square and multiply algorithm, computing $[k]G$ over $\mathbf{C}_{p,a_4,a_6}$ using one double and one add per bit of k.

Moreover, one can use projective coordinates X_P, Y_P, Z_P of point $P = x_P, y_P$ where $x_P = X_P/Z_P$ and $y_P = Y_P/Z_P$ when $Z_P \neq 0$. With projective coordinates every point of the curve has $p-1$ different representation. This can be used to execute leak-resistant computation (by changing the point representation before realizing the scalar multiplication) [4]. Point additions and doubling are then computed without inversion in $\mathbb{F}_p$.

Combined with algorithm 2, we can spare the computation of $Y_{[k]G}$ ($y_{[k]G}$ is recomputed at the end of the algorithm if necessary). Formulæ for adding and doubling points optimized for RNS are given in [20]. We briefly recall them in the following table.

$P+Q$	$2P$
$A \leftarrow Z_P X_Q + X_P Z_Q$	$E \leftarrow Z_p^2$
$B \leftarrow 2X_P X_Q$	$F \leftarrow 2X_P Z_P$
$C \leftarrow 2Z_P Z_Q$	$G \leftarrow X_P^2$
$D \leftarrow a_4 A + a_6 C$	$H \leftarrow -4a_6 E$
$Z_{P+Q} \leftarrow A^2 - BC$	$I \leftarrow a_4 E$
$X_{P+Q} \leftarrow BA + CD + 2x_{P-Q} Z_{P+Q}$	$X_{2P} \leftarrow FH + (G-I)^2$
	$Z_{2P} \leftarrow 2F(G+I) - EH$

At the end of the scalar multiplication, an inversion is needed to recompute $x_{[k]G} = X_{[k]G}/Z_{k[G]}$. In our results this final inversion is taken in account, considering that we use little Fermat's theorem to compute the inversion (which is possible with our design, and does not cost any gate except in the sequencer).

The main feature of RNS compared to classical representation is that multiplication is almost free while all the computation complexity is on reduction. Therefore, it is interesting to find $AB+CD$ pattern in the addition and doubling law of the curve. This is done by the table given above in 13 reductions for 1 Montgomery ladder step (1 point-addition and 1 point doubling).

2.3 Base choice

Our purpose is to use Kawamura's base extension in a massively parallel architecture. A value r is set as the word depth, and B and $\tilde{B}$ are chosen to be pseudo-Mersenne values $m_i = 2^r - \epsilon_i$, with $\epsilon_i < 2^q$ and $q < r/2$. $\tilde{B}$ is chosen exactly the same manner. Regarding addition and doubling formulæ, we can set α of algorithm 1 to 45. Indeed, the maximum value we have to reduce is $Z_{2P} \leftarrow 2F(G+I) - EH$. As F,G,H and I are less than $3p$, $2F(G+I)$ is at most $36p^2$. Since we can not afford to set negative input in Red_{Montg}, and we are unable to verify that $2F(G+I) > EH$, we have to calculate $(3p-E)H$ which is positive, and less than $9p^2$. Therefore $M > 45p$. As it is shown in Radix-RNS transformation subsection, 2^r is set as the m_0 value. In order to spare gates in our design, q has to be as small as possible. If the targeted technology is a Stratix family FPGA, we will use 18×18 or 36×36 multipliers. Here are the main features of chosen bases for the use of 18×18 and 36×36 multipliers (r is the word size in bits, n is the number of parallel rower modules and q is the max size of the ϵ_i in bits) :

curve	160	192	256	384	512	160	192	256	384	512
r	17	18	18	18	18	34	33	33	36	35
n	10	11	15	22	29	5	6	8	12	15
q	7	7	8	8	9	5	6	6	7	8

3 Hardware architecture

3.1 Architecture overview

Already published architecture using RNS : Kawamura [10] proposed an architecture suitable for RSA calculation using his base extension algorithm. His general architecture is the same as ours and is given by the upper outline of the figure 1. He divided his design in multiple "Rower" modules, which were in charge of calculating $|\sum_{i=1}^{n}(M\xi_i)/m_i|_{m_j}$. and a "Cox" module in charge of calculating $\lfloor\sum_{i=1}^{n}(\xi_i/2^r)\rfloor$. "Cox" design is very simple (a small adder). In [16] an improvement took the advantage of setting one cox per Rower. He then spared one cycle per reduction, computing γ in the same cycle. The results were interesting, but the Rower pipeline was not deep enough to reach high clock frequency (3 stages).

Improvements in our design : Our architecture is an improvement of Kawamura's [10] [16] which makes it

– suitable for elliptic curves,
– able to provide protections against side-channel attacks,
– designed to reach high clock frequencies.

The first limitation of Kawamura's architecture is the usage of only one RAM per channel. It only can execute a squaring, or multiply by a ROM data. This is not a limitation for RSA, the exponentiation algorithm only executes a square and a multiply by a constant, and so does the base reduction. This limitation is no more acceptable for elliptic curves. A general purpose register file (GPRs) must be added in order to multiply 2 local variables. Also the needed precalulation must be redefined. This point is focused in the subsection 3.3.

The second limitation of Kawamura's architecture is the design of the pipeline core, which executes the operation $acc = |x \times y + acc|_{m_i}$. Kawamura's goal was to keep busy every pipeline stage 100% of the cycles. To do so, he designed a 3 stage pipeline, and needed to use 3 times less Rower than the number of channels he had in B and $\tilde{B}$. Our architecture increases the pipeline depth to reach higher clock frequencies. We show that a 100% pipeline occupation is not really necessary for elliptic curves : it is easy to keep a good pipeline occupation even with deeper pipelines, with as many Rower as channels in B and $\tilde{B}$, and with less channels for elliptic curves than for RSA (considering 160 bits curves versus 1024 bits bases for RSA). This point is focused in the subsection 3.2.

Our architecture is showed on figure 1. It is the same as in [10] for the upper scheme part. Thus the Rower design (lower part of the scheme) is completely different. It is mainly composed with n parallel channels which execute $acc = |x \times y + acc|_{m_i}$ at each cycle and get operands and put the result from/in one of the 16 General Purpose Registers (GPR). Therefore, our architecture is able to compute a multiplication in $\mathbb{Z}/M\mathbb{Z}$ at each clock cycle. Our Rower architecture is described in 3.2.

We propose a RNS-Radix transformation in subsection 3.4, and eventually discuss about consequences of our choice for resistance against side channel attacks in 3.5.

3.2 Pipeline Architecture

In order to get high speed, the Arithmetical and Logical Unit(ALU) must execute $|r_1 \times r_2|_m$ and accumulate the result at each clock cycle.

Considering this constraint, algorithm 3 computes the modular multiplication for any pseudo-Mersenne number. The accumulation may be executed at every step of the algorithm. For every value P,P_{lsb} are the r less significant bits of P, while $P_{msb} = \lfloor P/2^r \rfloor$.

As it is shown in algorithm 3 , there are in the proposed pipeline structure only 5 generic operands:

– a $r \times r$ multiplier,
– a $r \times q$ multiplier,
– a $q \times q$ multiplier,
– 2 r modulo-adder taking two entries less than m_i, (4 if we consider accumulation, see the pipeline subsection)
– a r modulo-adder taking one over two entries less than m_i, the other being less than 2^r.

Algorithm 3 $MM(r_1, r_2, m_i)$

```
Require: $r_1$ and $r_2 < 2^r$
Require: $m_i = 2^r - \epsilon_i$ a pseudo-Mersenne number
Ensure: $|r_1 \times r_2|_{m_i}$
1: $P = r_1 \times r_2$
2: $Q = P_{msb} \times \epsilon_i$   $R = |P_{lsb}|_{m_i}$
3: $R = Q_{msb} \times \epsilon_i$   $S = |Q_{lsb} + P_{lsb}|_{m_i}$
4: $T = |R + S|_{m_i}$
5: return $T$
```

Algorithm 4 $3_{add}(P, Q, R)$

```
Require: $P$,$Q$ and $R$ : vector($r$)
Ensure: $P + Q + R$
1: $X$ : vector($r$) = $P$ xor $Q$ xor $R$
2: $C$ : vector($r$) = MAJ($P$,$Q$,$R$)
3: return $X + 2C$
```

The multipliers are not an issue for FPGA, since both Altera Stratix and Xilinx Virtex families have got multiplier blocks. The following assumption are taken :

– A $a \times b$ multiplier will be implemented in a single FPGA DSP block, for every a and $b < 36$, even if b is 9 bit wide.
– A $a \times b$ multiplier will be implemented in a single 9×9 blocks if both a and b are smaller than 9.

These facts have been verified for every synthesis during this study.

Modular addition takes advantage on the fact that an addition by three operands of size r only costs one LUT pass through and one addition of $r + 1$ operands, by using algorithm 4. Then, if a and b are less than m_i, $|a + b|_{m_i}$ can be computed by computing in parallel $r_1 = a + b$ and $r_2 = a + b + \epsilon_i$ and by considering the carry of r_2 to choose the correct result. Figure 2 describes the adder design. If $m_i < a \leq 2^r$, an extra addition $a + b + 2\epsilon_i$ is required.

Two pipeline architectures are proposed in this paper (figure 2). Both try to balance the pipeline stages with one another, but make different assumptions :

– The first one makes the assumption that the a modular addition is twice faster than a multiplication.
– The second one makes the assumption that a modular addition runs as fast as multiplication.

To increase the pipeline occupation, we overlap independent operations : for example, the computation of B and C start before the one of A is finished. Then, the wait states of the calculation of A, are taken up by B and C. This technique increases the pipeline occupation, but may increase the number of needed general purpose registers. This is a trade-off. It is analysed in the subsection 3.3.

On an Altera Stratix II chip, the maximum clock frequency of the 5 stage pipeline is 110 MHz, while the 6 stage reaches 158 MHz. At is is shown in subsection 3.3, the percentage of idle states is respectively 95% and 90% for a pipeline of depth 5 and 6, and a design with 5 channels (for a 160 bit curve with a channel length of 33 bits). This is the worst case of this study for pipeline occupation. We can then conclude that the 6 stage pipeline is the best choice for Stratix II technology. This study has to be done again for each target technology (including ASICs).

3.3 Memory

Precalculations and ROM content : The Rower main ROM is filled with the precalculated values described in this subsection.

Algorithm 5 *Reduction*(GPR_1, GPR_2) on a single Rower

```
Require: $X$ a value we wish to reduce $GPR_1 = |X|_{m_i}$ and $GPR_2 = |X|_{\tilde{m_i}}$
Ensure: $Red_{Montg}(X, p, B, \tilde{B})$ in $GPR_1$ and $GPR_2$
1: cycle 1 : $GPR_1 \leftarrow |GPR_1 \times p^{-1} M_i^{-1}|_{m_i}$
2: wait 1 : wait for $GPR_1$
3: cycle $1'$ : $out \leftarrow GPR_1$
4: cycle $2 + j$ $(j \in [0, \cdots, n-1])$ : $GPR_1$ accumulates $|in \times M_j p (M\tilde{M_i})^{-1} + M_{cox}|_{\tilde{m_i}}$
5: cycle $2 + n$ : $GPR_1$ accumulates $|GPR_2 \times (M\tilde{M_i})^{-1}|_{\tilde{m_i}}$
6: wait 2 : wait for $GPR_1$
7: cycle $3 + n$ : $out \leftarrow GPR_1$ and $GPR_2 \leftarrow |GPR_1 \times \tilde{M_i}|_{\tilde{m_i}}$
8: cycle $4 + n + j$ $(j \in [0, \cdots, n-1])$ : $GPR_1$ accumulates $|in \times \tilde{M_j} + M_{cox}|_{m_i}$
9: wait 3 : wait for $GPR_1$
```

Not considering reduction, we need during the calculation the following 3 variables : a_4, a_6 and $3p$: a_4 and a_6 to compute D and I, $3p$ to compute any subtraction (for example Z_{P+Q} and Z_{2P}). To compute H it is possible to precompute $-4a_6$. Eventually we need $|M^2|_p$ to compute Montgomery representatives. For computation we also need 0, 1 and -1 for every channel. For radix-RNS transformation we need 2^r and for RNS-radix we need 2^{-r} and -2^{-r} only on $\tilde{B}$ (see subsection 3.4).

Algorithm 5 gives a fast version of the algorithm in our architecture. Cycle $1'$ can be executed with another instruction. During cycles $2 + j$ and $4 + n + j$, the main bus is set to $out[j]$ the output of the j^{th} channel. The needed precomputed values are

- for a channel m_i of B (modulo m_i) :
  $p^{-1}M_i^{-1}$, $\tilde{M_j}$ for $j \in [0; \cdots; n-1]$.
- for a channel $\tilde{m_i}$ of $\tilde{B}$ (modulo $\tilde{m_i}$) :
  $M_j p (M\tilde{M_i})^{-1}$ for $j \in [0; \cdots; n-1]$, $(M\tilde{M_i})^{-1}$.

For a FPGA implementation, all these values may be set in ROMs, a curve change can be done by loading a different bitstream. Our results are given for fixed ROM. If it is necessary to change the curve during runtime, or if an ASIC implementation is needed, user may choose between 2 options :

- use RAMs instead of ROMs. This allows to change bases too, but all the values above have to be computed.
- reduce the number of curve-dependant values. Only 5 precomputed values per channel are needed : p, a_4, a_6, $|M^2|_p$ and $|-p^{-1}|_M$. It costs 2 extra cycles and 2 extra wait per reduction.

For each Rower design, two extra small ROMs are needed, each one containing two values. The ROM_{m_i} holds m_i and $\tilde{m_i}$. The ROM_{cox} holds the value $|-Mp\tilde{M_i}^{-1}|_{\tilde{m_i}}$ and $|-\tilde{M}|_{m_i}$ when the cox module set up the signal *cox*, these values are injected in the pipeline.

General purpose register file : Elliptic curves formulæ use local variables which have to be multiplied with one another, contrary to RSA which only has to square or multiply by a constant. That is why our architecture uses for every channel a general purpose register file (GPRs). Since every local variable has to be evaluated in M and $\tilde{M}$, one need twice as GPR as the maximum of local variables.

As it is explained in the pipeline subsection, operations are overlapped : intermediate result computation may start before the previous is finished. This may implicate an increase of the number of needed registers.

The following table shows that 16 GPR are enough : local variables are limited to 7 and reductions are at least executed by 2, most of time by 3. An 8th local variable is taken by x_G the exponentiated point abscissa. This leads to a pipeline fill rate of 90% for the 6 stage pipeline and 16 GPR per channels. This is a good trade-off.

step	calculation	living variables
1	$A\ B\ C$	$X_P\ Z_P\ A\ B\ C$
2	$D\ Z_{P+Q}$	$X_P\ Z_P\ A\ B\ C\ D\ Z_{P+Q}$
3	$X_{P+Q}\ E\ F$	$X_P\ E\ F\ Z_{P+Q}\ X_{P+Q}$
4	$G\ H\ I$	$E\ F\ G\ H\ I\ Z_{P+Q}\ X_{P+Q}$
5	$X_{2P}\ Z_{2P}$	$X_{2P}\ Z_{2P}\ Z_{P+Q}\ X_{P+Q}$

3.4 Radix-RNS transformation

The RNS representation is not practical for using outside the design. Moreover, there is no need for extra material either to transform a number X from its classical multiprecision representation $(X_{n-1}, \cdots, X_0)$ where $X = \sum_{i=0}^{n-1} X_i 2^i)$ to its RNS representation $(x_1,...x_n)$, nor the contrary. Using our architecture, the Radix-RNS transformation is trivial, if the sequencer can set up the main bus to the X_i values, and $|2^r|_{m_i}$ is in ROM.

For RNS-Radix transformation, the main idea is to set the channel m_0 of B to 2^r. If it is so, $X_0 = x_0$. To find X_1, all we need to do is to compute $X \leftarrow (X - x_0)/2^r$. As 2^r is not co-prime with M, this computation is done over $\tilde{B}$, and the use of $B_{ext}(X, \tilde{B}, B)$ gives $X_1 \leftarrow x_0$. By repeating this operation we compute $X_2, \cdots, X_n$.

This algorithm is not the most efficient but does not cost a single gate in our architecture. It never costs more than 0.3% of the total scalar multiplication time.

3.5 Side channel attacks

Our architecture supports an inherent capability to treat simple power analysis (SPA), or differential power analysis (DPA) and fault threats. Indeed, the Montgomery ladder is particularly efficient to counter both side channel attacks and fault attacks (no operation is dummy). Our finite state sequencer does not have any branch capability, bits of k are only read at the beginning of each Montgomery-ladder step to invert registers. Therefore no information leaks from the computation time.

Moreover, randomness can be introduced at the very beginning of the algorithm by changing G representation, replacing $(x_G, 1)$ by $(x_G \times a_1, a_1)$ and $\mathcal{O} = (a_2, 0)$, where a_1 and a_2 are random values. This countermeasure avoids Fouque's attacks [6] on collisions. The only SPA vulnerability is address-bit SPA attack [12], which is difficult to realize on real design. Moreover, to be DPA resistant k may be added with $a_3 \times \#C_{p,a_4,a_6}$ where a_3 is a random value. The impact on speed is $log(a_3)/log(p)$ on the speed. There is no impact on the design size. More robust randomizations of k are also possible.

4 Result and comparison

In this section we give the overall results, and compare it to different architecture given in the open literature.

4.1 Results

Our target technology is the Altera Stratix family. This choice has very few impact on results compared to Xilinx Virtex family [18]. We chose Altera because of the availability of the Quartus toolchain during the study. Among all the Altera products, we focus on 2 generations. First Stratix are the Altera FPGA at the 130 nm process node. The Xilinx equivalent is the Virtex II-pro. We also have fit our design in the Stratix II generation FPGA (the 90 nm process node), much more efficient. The equivalent is the Virtex IV by Xilinx.

We randomly chose elliptic curves of the following size : 160, 192, 256, 384 and 512 bits. No restriction were given for p, a_4 and a_6 but to be a valid elliptic curve. The result given below does not depend on the effective choice of p but only on $log_2(p)$. The number of Rower n as well as the word depth r are also given. These values are the most efficient considering the curve size.

For every fit we give the considered FPGA family and the exact reference of the chip. The maximum reachable frequency as well as the computation time for a whole scalar multiplication $[k]G$ are given. The size of the exponent k is the same as the size of p. The final inversion, the $y_{[k]G}$ recalculation, the Radix-RNS and RNS-Radix transformations are included in the result.

Eventually the FPGA occupation is given. The Stratix FPGA are composed by logic elements (LE). LE are equivalent to the Look-Up Table (LUT) of the Virtex II-pro. Therefore a Virtex II-pro slice counts for two LE. No equivalent to the slice exists in the Stratix family. On the Stratix II family Altera introduced the Altera Logic Module (ALM), the Virtex IV slice equivalent. The number of used DSP blocks (multipliers) is also given. Altera gives the DSP occupation in numbers of 9×9 multipliers used. Stratix and Stratix II DSP blocks can indeed be configured as one 36×36 multiplier, two 18×18 or eight 9×9 multipliers.

Family	curve	model	n	r	size	DSP	frequency	speed
Stratix	160	EP1S20F484C5	5	34	11431 LE	74	92.6	0.57 ms
	192	EP1S30F780C5	6	33	12480 LE	80	89.6	0.72 ms
	256	EP1S60F780C5	8	33	16200 LE	125	90.7	1.17 ms
	384	EP1S80F1020C5	11	36	25279 LE	176	90.0	2.25 ms
	512 [3]	EP1S80F1020C5	15	35	48305 LE	176	79.6	4.03 ms
Stratix II	160	EP2S30F484C3	5	34	5896 ALM	74	165.5	0.32 ms
	192	EP2S30F484C3	6	33	6203 ALM	92	160.5	0.44 ms
	256	EP2S30F484C3	8	33	9177 ALM	96	157.2	0.68 ms
	384	EP2S60F484C3	11	36	12958 ALM	177	150.9	1.35 ms
	512	EP2S60F484C3	15	35	17017 ALM	244	144.97	2.23 ms

The results given above show some properties of the chosen design. First the maximum frequency hardly falls with the size of the design. Indeed, no carry is propagated on the whole size of the operands due to RNS. The critical path is then not related to the design structure. For some fit it is in the sequencer, and for some other in a Rower. No further instruction has been given to the fitter except the pursuit of the maximal frequency.

As speed was our main concern, no RAM blocks were used. If size is a matter, ROMs and GPR may be fitted in RAM blocks, to spare logic. The Stratix family lacks some DSP for 512 bit curves, while Stratix II have far enough resources to fit for any cryptographic size (the EP2S60F484C3 is a middle size matrix in the Stratix 2 family).

4.2 Comparison

In this subsection we compare our design with other papers in the open literature. Our architecture supports any elliptic curve over $\mathbb{F}_p$ and is resistant to side channel attacks. We compare it with 3 papers we consider significant:

- the first one is another design based on RNS. In this paper, modular multiplication is realized through an Horner scheme. To our knowledge it is the only implementation using RNS for curves. It is realized by Schinianakis et al [23]. It is implemented on an ASIC and on a FPGA.
- the second one is described in [11], and is based on a multi-precision algorithm. The main idea is an important work on the pipeline architecture for the classical multiprecision algorithm and on long word additions, through carry-look adder. It is realized on a Virtex II-pro. The scalar multiplication is based on a NAF recoding. This is to our knowledge the fastest implementation of elliptic curve scalar multiplication with generic curves. It outperforms every implementation given in [13] and is as fast as [21] using less resources.

[3] the EP1S80 does not have enough DSP blocks, multipliers are fitted in the LE blocks, and frequency falls.

– the third one is described in [7]. The design is specific to a particular curve, p being a pseudo Mersenne value. The main idea is a dual clock design, according DSP blocks to run at their maximum speed in Virtex 4 design, 500 MHz. It is the fastest FPGA implementation of elliptic curve scalar multiplication over $\mathbb{F}_p$, but with restrictions on p.

paper	curve	FPGA family	FPGA model	size	freq.(MHz)	speed
This work	160 any	Stratix	EP1S20F484C5	11431 LE	92.6	0.57 ms
	256 any	Stratix	EP1S60F780C5	16200 LE	90.7	1.17 ms
	160 any	Stratix II	EP2S30F484C3	6203 ALM	165.5	0.32 ms
	256 any	Stratix II	EP2S30F484C3	9177 ALM	157.2	0.68 ms
[23]	160 any	Virtex	XCV1000E-8	21000 LUT	58	1.77 ms
	256 any	Virtex	XCV1000E-8	36000 LUT	39.7	3.95 ms
[11]	160 any	Virtex II-pro	XC2VP30	2171 sl.	72	1 ms
	256 any	Virtex II-pro	XC2VP30	3529 sl.	67	2.27 ms
[7]	224 NIST	Virtex 4	XC4VFX12	1580 sl.	487	0.36 ms
	256 NIST	Virtex 4	XC4VFX12	1715 sl.	490	0.49 ms

As a conclusion, our implementation is largely faster and smaller than [23]. Architecture [11] has got a better time surface trade-off (the ratio is about 1.5 if we consider that a Virtex 2 pro slice is equal to 2 Stratix LE, slices having not correspondant in Stratix products). Nevertheless it does not compute $k[G]$ as fast as ours, even on comparable technologies. It is eventually not resistant against side channel attacks. An overhead is needed to be SPA resistant. Ours is natively SPA resistant.

Architecture described in [7] is faster and presents a better size-area tradeoff regarding to ours (assuming that a Stratix II ALM and a Virtex IV slice are equivalent, and only considering slices). Guneysu's work only computes $[k]P$ over NIST curves, using pseudo-mersenne p. We can consider that using pseudo mersenne primes reduces the time complexity by a factor between 2 and 1.68 if lazy reduction is used. Then we can see that our results is competitive in term of latency regarding to his for 224 bit, and becomes better for 256 bits. Moreover, Guneysu's clock speed (500 MHz) is particularly high and may represent an obstacle for industrial integration, and for an ASIC implementation (making the multipliers work twice as fast as the rest of the design would be impossible). Of course it is difficult to realize a fair comparison at this point since the two designs do not target the same curves, but RNS is a competitive alternative for general F_p curves, especially for high security levels.

5 conclusion

In this paper is presented a hardware architecture realizing elliptic curve scalar multiplication over any curve in $\mathbb{F}_p$, which uses RNS representations to speed up the computation. RNS supports a wide parallelization capability for arithmetic in $\mathbb{F}_p$. The overhead given by the elementary operation ($|a \times b|_m$) is well pipelineable, even with large pipelines (6 stages), and contrary to RSA, the inherent parallelism of elliptic curve operations allows to easily fill the pipeline. Capability to support high clock frequency does not fall with the curve size.

In our future work, we will study other technologies like ASICs (with the usage of RAMs instead of the actual ROMs, other curve shapes, or other operations in elliptic curve cryptography, like pairings.

Acknowledgement : we would like to thank the anonymous referees for their detailed review of this paper and their helpful suggestion. Thanks to David Lubicz, Sylvain Duquesne and Jeremie Detrey for their contribution to this work.

Fig. 1. General architecture and focus on the Cox and Rower design

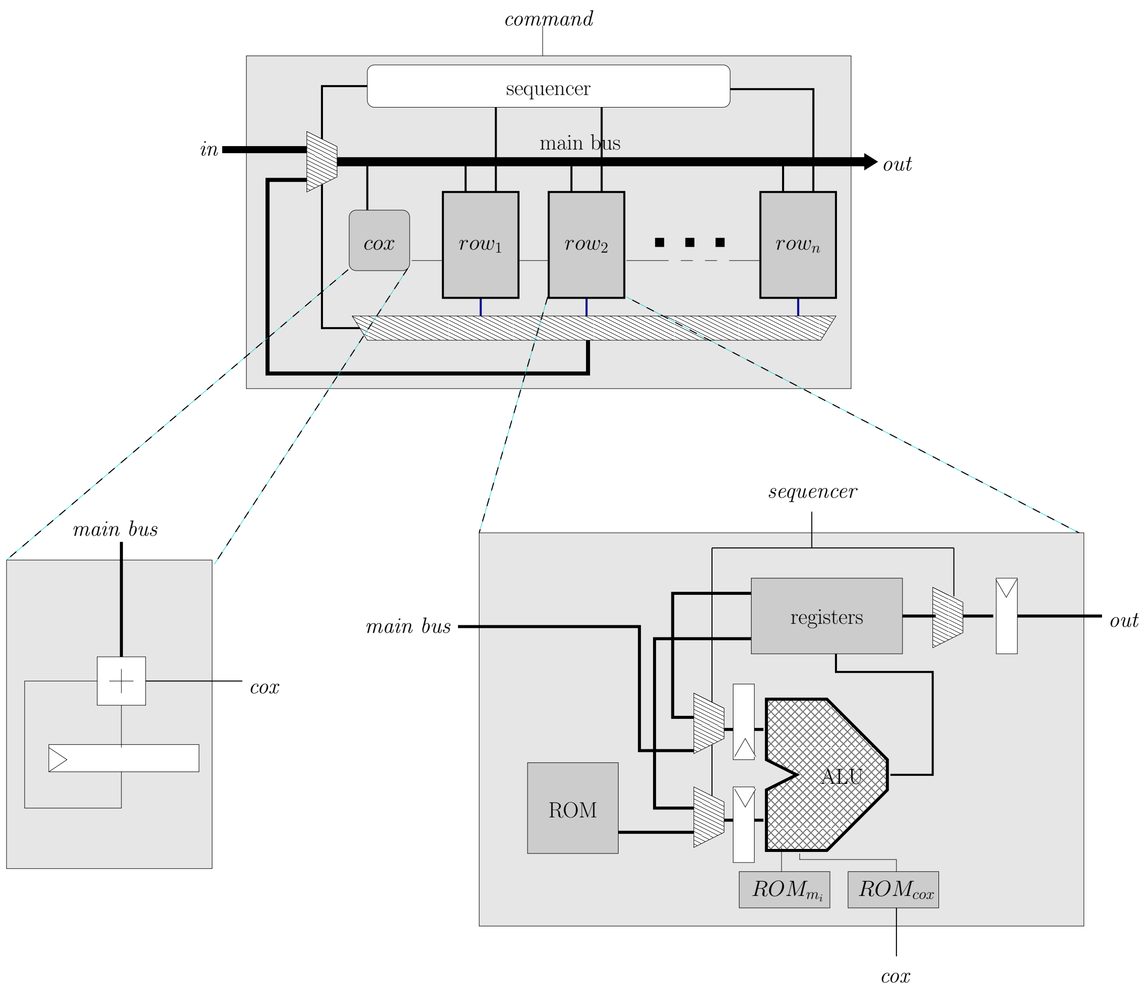
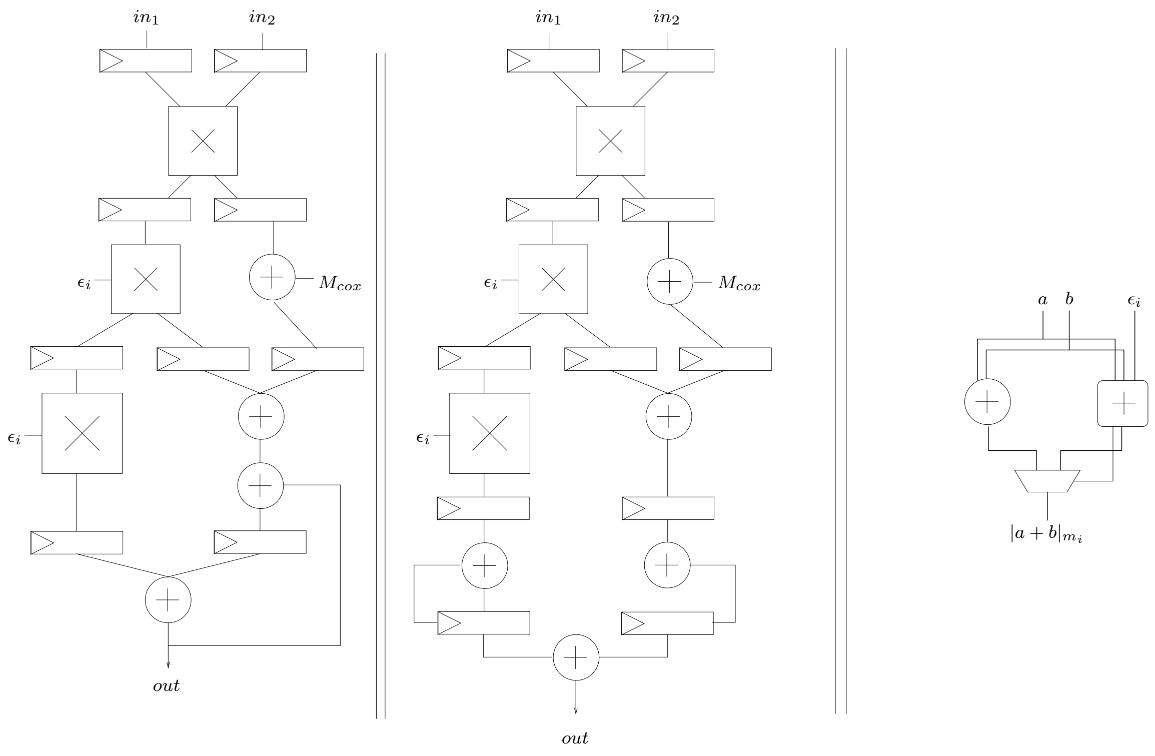


Fig. 2. 5 and 6 stage pipeline, and an adder modulo m_i



References

1. Jean-Claude Bajard, Laurent-Stphane Didier, and Peter Kornerup. An rns montgomery modular multiplication algorithm. *IEEE Transactions on Computers*, 47(7):766–776, 1998.
2. Jean-Claude Bajard, Laurent Imbert, Pierre-Yvan Liardet, and Yannick Teglia. Leak resistant arithmetic. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 116–145. Springer Berlin / Heidelberg, 2004.
3. Lu Chen, Chen Yanpu, , and Bian Zhengzhong. An implementation of fast algorithm for elliptic curve cryptosystem over GF(p). *Journal of Electronics (China)*, 21(4):346–352, July 2004.
4. Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *CHES*, pages 292–302, 1999.
5. Harold Edwards. A normal form for elliptic curves. *Bull. Amer. Math. Soc. 44*, 2007.
6. Pierre-Alain Fouque and Frédéric Valette. The doubling attack - *hy upwards is better than downwards.* In *CHES*, pages 269–280, 2003.
7. Tim Güneysu and Christof Paar. Ultra high performance ecc over nist primes on commercial fpgas. In *CHES '08: Proceeding sof the 10th international workshop on Cryptographic Hardware and Embedded Systems*, pages 62–78, Berlin, Heidelberg, 2008. Springer-Verlag.
8. Kimmo U. Jarvinen and Jorma O. Skytta. High-speed elliptic curve cryptography accelerator for koblitz curves. *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, 0:109–118, 2008.
9. Marc Joye and Sung-Min-Yen. The montgomery powering ladder. In *CHES '02: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems.*
10. Shinichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo. Cox-rower architecture for fast parallel montgomery multiplication. In *Advances in Cryptology EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 523–538. Springer Berlin / Heidelberg, 2000.
11. Nele Mentens. *Secure and Efficient Coprocessor Design for Cryptographic Applications on FPGAs.* PhD thesis, Ruhr-University Bochum, 2007.
12. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In *CHES*, pages 144–157, 1999.
13. Guerric Meurice de Dormale and Jean-Jacques Quisquater. High-speed hardware implementations of elliptic curve cryptography: A survey. *J. Syst. Archit.*, 53(2-3):72–84, 2007.
14. Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology—CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer-Verlag, 1986, 18–22 August 1985.
15. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.
16. Hanae Nozaki, Masahiko Motoyama, Atsushi Shimbo, and Shin-ichi Kawamura. Implementation of rsa algorithm based on rns montgomery multiplication. In *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pages 364–376, London, UK, 2001. Springer-Verlag.
17. National Institute of Science and Technology. The digital signature standard. Technical report, http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf.
18. White Paper. Stratix vs. virtex-ii pro fpga performance analysis. Technical report, www.altera.com/literature/wp/wpstxvrtxII.pdf.
19. Karl C. Posch and Reinhard Posch. Modulo reduction in residue number systems. *IEEE Trans. Parallel Distrib. Syst.*, 6(5):449–454, 1995.
20. M. Ecegovac S. Duquesne, J.C. bajard. Combining leak-resistant arithmetic for elliptic curves define over $\mathbb{F}_p$ and rns representation.
21. K. Sakiyama, N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede. Reconfigurable modular arithmetic logic unit for high-performance public-key cryptosystems. In *Reconfigurable Computing: Architectures and Applications*, volume 3985 of *Lecture Notes in Computer Science*, pages 347–357. Springer Berlin / Heidelberg, 2006.
22. Akashi Satoh and Kohji Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Transactions on Computers*, 52:449–460, 2003.
23. Dimitrios M. Schinianakis, Apostolos P. Fournaris, Harris E. Michail, Athanasios P. Kakarountas, and Thanos Stouraitis. An rns implementation of an fpelliptic curve point multiplier. *Trans. Cir. Sys. Part I*, 56(6):1202–1213, 2009.
24. P. P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in rns. *IEEE Trans. Comput.*, 38(2):292–297, 1989.
25. Robert Szerwinski and Tim Gayneysu. Exploiting the power of GPUs for asymmetric cryptography. In *Cryptographic Hardware and Embedded Systems CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 79–99. Springer Berlin / Heidelberg, 2008.