MPHELL: A fast and robust library with unified and versatile arithmetics for elliptic curves cryptography (extended version)

Titouan Coladon, Philippe Elbaz-Vincent, Cyril Hugounenq

MPHELL: A fast and robust library with unified and versatile arithmetics for elliptic curves cryptography (extended version)

Titouan Coladon
Univ. Grenoble Alpes
Institut Fourier
Grenoble, France
titouan.coladon@univ-grenoble-alpes.fr

Philippe Elbaz-Vincent
Univ. Grenoble Alpes
Institut Fourier
Grenoble, France
philippe.elbaz-vincent@univ-grenoble-alpes.fr

Cyril Hugounenq
Univ. Grenoble Alpes
Institut Fourier
Grenoble, France
cyril.hugounenq@univ-grenoble-alpes.fr

***Abstract*—We propose a new versatile elliptic curves cryptography library based on unified arithmetics and various low-level arithmetics with a focus on protection against simple power analysis and an abstract layer for easy customisations. The implementations are oriented toward industrial applications and embedded devices. The number arithmetic used in the library is partly inherited from GMP with several improvements using adapted Montgomery representation and windowing techniques. We also present an improved AMNS (Adapted Modular Number System) arithmetic with competitive running time. The abstraction layer allows for the integration of external arithmetics (e.g., other libraries or hardware co-processor), general number systems and randomization of arithmetics. The library has the advantage of proposing standard elliptic curves but gives also the possibility to use curves in different settings such as Weierstrass form in co-Z coordinates, Jacobi quartic or Edwards forms (as well as their associated conversions functions). It has been extensively tested on x86-64, ARM 32/64 bits, STM32 architectures and also in real-world applications. We present some comparative elliptic curves signatures timings for different curves without taking into account the specificity of the curves in our library (as opposed to OpenSSL for instance).**

***Index Terms*—elliptic curves cryptography, unified arithmetics, adapted modular number system**

I. Introduction

Creating secure implementations for elliptic curves cryptography (ECC) while preserving performances is not an easy task as shown by the attacks [1]–[4] on OpenSSL [5] and GnuPG [6]. Unified formula for elliptic curves cryptography was introduced in the early 2000s [7]–[9] in order to prevent simple power analysis (SPA) or even differential power analysis (DPA) and have been extended to a wide family of coordinate systems [10]–[14]. Building on previous works and the need for robust implementations, we propose a new versatile ECC library, called `MPHELL` [1] , based on unified arithmetics with a focus on protection against simple power analysis and an abstract layer for easy customisations. It has been extensively tested on x86-64, ARM 32/64 bits, STM32 architectures and also in real-world applications. Our library has the advantage to propose standard elliptic curves (all those from [15]) but gives also the possibility to use curves in different settings such as Weierstrass form in co-Z coordinates, Jacobi quartic or Edwards forms (as well as their associated conversion functions)[2]. The number arithmetic used is inherited from GMP [16] and has some improvement using Montgomery representation [17] and windowing techniques. It also has a "Modular Number System" module [18] with a focus on the "Adapted Modular Number System" (AMNS) for which we extend and improve the results of [19]. Part of the mathematics behind the elliptic curves arithmetics were described in [20]. Our contribution intends to better address the needs of a fast arithmetic library for elliptic curves with the following features:

- Secure against simple power analysis,
- Easy to customize (e.g. usable with several types of curves or versatile number arithmetics such as AMNS),
- Using optimized number arithmetic,
- Usable in industrial context,
- Usable on microcontrollers (e.g., STM32), ARM 32 bits and 64 bits, and x86 architectures (32 bits and 64 bits)
- Competitive against other ECC libraries.

The library has been designed with GNU/Linux systems as main targets (frequent on embedded systems) and for curves over prime fields.

This work is organized as follows: in section II, we present an improved AMNS, following the works of Didier, Dosso and Véron [19] and Dosso [21] and show optimality results for some family of elliptic curves. In section III, we detail the design of a new library, called `MPHELL` , for the arithmetic of elliptic curves with different types of low-level arithmetics (e.g., AMNS) and unified arithmetics for elliptic curves in order to be SPA resistant. In section IV, we give detailled timings for `MPHELL` on different types of architectures and compare it to common libraries. In section V we present our conclusions.

[1]https://www-fourier.univ-grenoble-alpes.fr/mphell/

[2]The formulae used are mainly available in the Elliptic Curve Formula Database http://www.hyperelliptic.org/EFD.

In the following, for a prime number p, we denote by $\mathbb{F}_p$ the finite field with p elements (represented as $\mathbb{Z}/p\mathbb{Z}$). We will also denote by $\log$ the logarithm in base 2.

II. Improving AMNS

This section proposes several improvements on the work of Didier, Dosso and Véron [19] and Dosso [21]. Let us first introduce some notations: p will denote a prime number. For an arbitrary integer $n > 1$, we denote by $\mathbb{Z}[X]_n$ the set of integers polynomials of degree less (or equal) than n. Given a polynomial Q in $\mathbb{Z}[X]_n$ we denote by $\|Q\|_k$ the real k-norm of $\mathbb{R}^{n+1}$ restricted to $\mathbb{Z}[X]_n$ (the polynomials being seen as vectors). The positive integer ϕ will be either 2^{32} or 2^{64} depending on the targeted architecture (32 bits or 64 bits). Let $A \in \mathbb{Z}[X]_n$, we denote by $A \mod (E, \phi)$ the polynomial reduction $A \mod E$ where the coefficients of the result are computed modulo ϕ. We note $\overline{A}$ the polynomial A with its coefficients reduced modulo 2.

A. A reminder on AMNS

A Modular Number System (MNS), introduced by Bajard *et al* [18], allows to represent elements of $\mathbb{F}_p$ as polynomials. Such MNS is defined by a 4-tuple (p, n, γ, ρ) such that for all $x \in \mathbb{F}_p$ there exists $V \in \mathbb{Z}[X]$ such that $V(\gamma) = x \pmod p$ with $\deg(V) < n$ and $\|V\|_\infty < \rho$. In order to represent all the elements of $\mathbb{F}_p$, we need $p < (2\rho - 1)^n$. An Adapted Modular Number System (AMNS) is an MNS such that $\gamma^n = \lambda \pmod p$ with $|\lambda| \neq 0$ "small" (often lower than 10). γ is a root modulo p of the polynomial $E = X^n - \lambda$. E is called the external reduction polynomial. We use it to reduce the degree of AMNS polynomials after multiplication by replacing X^n by λ in the computation. An AMNS is defined by a 5-tuple (p, n, γ, ρ, E).

Another reduction is needed to keep polynomials of the AMNS such that $\|V\|_\infty < \rho$. We need an algorithm called "internal reduction" acting on the size of the polynomial coefficients. Different methods exist to achieve this reduction, but the best known one was introduced by Negre and Plantard [22] and uses a Montgomery scheme. In such setting, every x in $\mathbb{F}_p$ is represented by $V \in \mathbb{Z}[X]_{n-1}$ such that $V(\gamma) = x\phi \pmod p$.

We recall here the internal Montgomery reduction algorithm [19](section 4.1) which returns S such that $S(\gamma) = V(\gamma)\phi^{-1} \pmod p$ and $\|S\|_\infty < \rho$. This algorithm reduces the coefficients of V using the polynomials M and $M' = -M^{-1} \mod (E, \phi)$. The modulus ϕ is either 2^{32} or 2^{64} according to the target architecture. We set $Q = V \times M' \mod (E, \phi)$, $T = Q \times M \mod E$ and $S = (V + T)/\phi$.

In practice, $\|M\|_1$ must be "small enough" to keep this internal reduction algorithm consistent.

B. Improving the generation of AMNS

According to [21](Proposition 2.2), the limit on $\|V\|_\infty$ is almost accurate. We can write $\|V\|_\infty \leqslant \omega \times (\rho - 1)^2$ to be more precise. The conditions on ρ and ϕ are there to make the AMNS internal reduction work [19](Algorithm 3). It turns out that we can improve these conditions. Set $M = m_0 + m_1X + ... + m_{n-1}X^{n-1}$. The accurate limit is $\|T\|_\infty \leqslant \phi \times \alpha$ with $\alpha = |m_0| + |\lambda| \times (|m_1| + |m_2| + ... + |m_{n-1}|)$.

We remind the definition of $\omega : \omega = 1 + (n-1)|\lambda|$.

To get $\|S\|_\infty \leqslant \rho$ we need :

$$\begin{aligned} \frac{\omega \times (\rho-1)^2 + \alpha \times \phi}{\phi} &< \rho, \\ \frac{\omega \times (\rho-1)^2}{\rho - \alpha} &< \phi. \end{aligned} \tag{1}$$

This equation gives the condition $\rho > \alpha$. In order to find a suitable ρ, we study the minimum of the real function $f(x) = \frac{|\omega| \times (x-1)^2}{x-\alpha}$ which is well defined and derivable on its domain $\mathbb{R}\backslash\{\alpha\}$. Its derivative is given by $f'(x) = \frac{|\omega| \times (x-1) \times (x-2\alpha+1)}{(x-\alpha)^2}$ and the minimum of f happens when $x = 2\alpha - 1$.

If $f(2\alpha - 1) < \phi$ then the internal reduction polynomial M can be used to create an AMNS. In practice, because we want ρ to be a power of 2, we need $f(2^{\log(2\alpha-1)}) < \phi$. We can set $\rho = 2^{\log(2\alpha-1)}$. This gives a smaller value for $f(\rho)$ than taking $\rho = 2^{\log(2\omega\|M\|_\infty)}$. Then, we can deduce new limits. We replace : $\rho \geqslant 2\omega\|M\|_\infty$, $2\omega\rho \leqslant \phi$ and $\|V\|_\infty \leqslant \omega\rho^2$ from [21] by $\rho > \alpha$, $\frac{\omega \times (\rho-1)^2}{\rho-\alpha} < \phi$ and $\|V\|_\infty \leqslant \omega(\rho-1)^2$.

We notice that $\rho > \alpha \geqslant \|B\|_1 > \frac{1}{2}\|B\|_1$, with B the matrix defined in [21](Theorem 2.2). The theorem's condition is satisfied. This calculation for ρ does not improve significantly the AMNS generation process, but it can help to find better AMNS in several cases. Two points have been improved:

- The limit: It is now $y = f(\rho)$ which have to be lower than $\phi = 2^{64}$, and not anymore $y = 2\omega\rho$,
- The calculation of ρ: to get $f(\rho)$ as small as possible and keep ρ as a power of 2 we use $\rho = 2^{\log(2\alpha-1)}$ instead of $\rho = 2^{\log(2\omega\|M\|_\infty)}$.

We can see on Figure 1 below the function f and its minimum ($y = f(2\alpha - 1)$) in green. In red the line $y = 2^{64}$ which is the maximum for $f(\rho)$ (or for $2\omega\rho$ with the old limit) to keep the internal reduction consistent. The blue line is the value $y = f(\rho)$ with $\rho = 2^{\log(2\alpha-1)}$ which is lower than 2^{64}: with the new limit and the new computation for ρ this AMNS is accepted. The orange line on top is the value of $y = 2\omega\rho$, with $\rho = 2^{\log(2\omega\|M\|_\infty)}$ which must be $< 2^{64}$ (red line) in the condition of [21]. This represents the old limit and the old computation of ρ, the AMNS would have been rejected with these values. The orange line under is the value of $y = f(\rho)$ with $\rho = 2^{\log(2\omega\|M\|_\infty)}$. This line uses the new limit but the old computation for ρ, the AMNS would have also been rejected. The graphic shows that the new limits and the new calculation of ρ allow to accept this AMNS, which would have been rejected by the conditions of [21]. We can notice that with $\rho = 2\alpha - 1$, the value $f(\rho)$ is minimal. But because ρ must be a power of 2 and $\rho > \alpha$ we can compare $f(\rho_1)$ and $f(\rho_2)$ with $\rho_1 = 2^{\log(2\alpha-1)}$ and $\rho_2 = 2^{\log(\alpha+1)}$. It leads to an improved choice of ρ, namely if $f(\rho_1) < f(\rho_2)$ we choose $\rho = \rho_1$, otherwise we choose $\rho = \rho_2$.

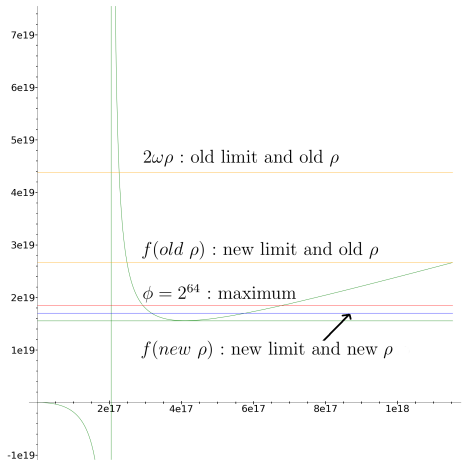

Figure 1. The new limits for AMNS.

C. Characterization of AMNS parameters

In order to give a characterization of AMNS parameters, we will prove that the following claim is necessary and sufficient (cf. Proposition 2.7 from [21]): *Assuming E and M from an AMNS as above, and* Res *denotes the resultant of polynomials in $\mathbb{Z}[X]$. Then* $\gcd(\mathrm{Res}(E, M), \phi) = 1$ *if and only if* $M' = -M^{-1} \pmod{(E, \phi)}$ *exists.*

Proof. From [21], we know that the condition is sufficient. It remains to show that it is necessary.

We will show that the conditions below, equivalent to $\gcd(\mathrm{Res}(E, M), \phi) = 1$, are necessary for M' to exist:

$$\begin{cases} \text{if } \lambda \text{ is even then } m_0 \text{ is odd (proposition 2.8 of [21]),} \\ \text{if } \lambda \text{ is odd then } \gcd(\overline{M}, X^n - 1) = 1. \end{cases} \tag{2}$$

Here is a proposition of a sufficient and necessary condition on M, such that M' exists. Instead of looking at the resultant we look to a linear equation system over the ring $\mathbb{Z}/\phi\mathbb{Z}$.
Let $M = m_0 + m_1 X ... + m_{n-1} X^{n-1}$ and $M' = x_0 + x_1 X ... + x_{n-1} X^{n-1}$ be an unknown polynomial. Then

$$M \times M' \pmod{E} = A \times \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}. \tag{3}$$

with

$$A = \begin{pmatrix} m_0 & \lambda m_{n-1} & \lambda m_{n-2} & \dots & \lambda m_2 & \lambda m_1 \\ m_1 & m_0 & \lambda m_{n-1} & \dots & \lambda m_3 & \lambda m_2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ m_{n-2} & m_{n-3} & m_{n-4} & \dots & m_0 & \lambda m_{n-1} \\ m_{n-1} & m_{n-2} & m_{n-3} & \dots & m_1 & m_0 \end{pmatrix}, \tag{4}$$

where the first line are the coefficients of degree 0, the second line the coefficients of degree 1 until the last line with the coefficients of degree $n-1$.

Hence, M is invertible $\pmod{E, \phi}$ if and only if there exists $M' = x_0 + x_1 X ... + x_{n-1} X^{n-1}$ such that $M \times M' \pmod{E, \phi} = 1$. This is equivalent to find a solution over the ring $\mathbb{Z}/\phi\mathbb{Z}$ of the following linear equation:

$$A \times \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{5}$$

The system above has a unique solution over $\mathbb{Z}/\phi\mathbb{Z}$ if and only if $\gcd(\det(A), \phi) = 1$ and as ϕ is a power of 2 the system has a solution if and only if $\det(A)$ is odd. The proposition 2.1 of [21] allows to replace the coefficients of A by their remainder modulo 2 to know the parity of $\det(A)$. Set $B = A \pmod 2$. If λ is even, then B is a triangular matrix with only $m_0 \pmod 2$ on the diagonal and we get back the condition $\det(A)$ is odd if and only if m_0 is odd. Else, if λ is odd, then B is the transpose of H_3 as defined in [21](p.64). Then the condition $\gcd(\overline{M}, X^n - 1) = 1$ is necessary and sufficient for $\det(A)$ to be odd. Finally, if $\gcd(\det(A), \phi) = 1$ then A is inversible in $\mathbb{Z}/\phi\mathbb{Z}$ and the solution is given by

$$M' = A^{-1} \times \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{6}$$

This demonstrates that the conditions (2) are necessary for M' to exist. And conclude that the condition of [21](Proposition 2.7) is necessary. □

D. An example of optimal parameters for AMNS

For the NIST prime P-521, we want to show that the best AMNS parameters is with $M = 2^{52}X - 1$. Indeed with the external polynomial $E = X^{10} - 2$, a root
$\gamma = 1524291284333980581729295522359944485228807686$
$848130444755447734192076044345588681699368214386 4$
$706890428842437116243275856679568746524830597 12$,
of E is such that $M(\gamma) \pmod p = 0$. So $\gamma^{-1} = 2^{52} \pmod p$. As M has almost all its coefficients equal to zero, it already speeds up the internal reduction. In addition $(2^{52})^2 = 0 \pmod \phi$ and $M' = 2^{52}X + 1$ has also almost all its coefficents equal to zero and makes again the internal reduction faster.

Indeed instead of using $2n^2$ multiplications we use only $2n$ multiplications which are always some number multiplied by 2^{52}. By consequences, the internal reduction which is the most time consuming operation on AMNS becomes almost free for the NIST P-521.

We want to show that the above M is optimal. Set $M = aX - 1$ with $|a| < \rho < 2^{64}$. The condition $M(\gamma) = 0 \pmod p$ is equivalent to $\gamma = a^{-1} \pmod p$. To have $M' = aX + 1$ in order to get $MM' = -1 \pmod{2^{64}}$ we need $a^2 = 0 \pmod{2^{64}}$. This last condition is equivalent to $a = \pm k \times 2^{32}$ for some integer k and as $|a| < 2^{64}$, we must have $|k| < 2^{32}$. Hence, there are less than $2 \times 2^{32} = 2^{33}$ possible values for $a = \gamma^{-1}$ and so for γ. As $2 \leqslant \gamma \leqslant p - 2$, the probability to have a "fitting" γ is $\dfrac{2^{33}}{p-4} < \dfrac{2^{33}}{2^{520}} = \dfrac{1}{2^{487}}$. If we consider that for a given p we can choose the external polynomial $E = X^{10} - \lambda$ with $1 \leqslant |\lambda| \leqslant 10$ and that for each external polynomial E

Table I
AMNS PARAMETERS FOR NIST AND BRAINPOOL (BP) CURVES.

p	NIST P-224	NIST P-256	NIST P-384	NIST P-521
n_{opt} $(k=64)$	4	5	7	9
n $(k=64)$	4	5	7	10
n_{opt} $(k=32)$	8	9	13	17
n $(k=32)$	11	13	18	22

p	BP 224	BP 256	BP 384	BP 512
n_{opt} $(k=64)$	4	5	7	9
n $(k=64)$	4	5	7	10
n_{opt} $(k=32)$	8	9	13	17
n $(k=32)$	11	12	18	25

there are less than $n = 10$ roots (the degree of $\gcd(X^p - X, E)$ is exactly the number of distinct roots of E in $\mathbb{F}_p$). Then the probability to have such a M is lower than $2 \times 10 \times 10 \times \frac{1}{2^{487}}$. Now if a is a power of 2 (like for the NIST P-521), there are less than 64 possibilities for a. And the probability to have such an AMNS for a given p of 521 bits is less than $2 \times 10 \times 10 \times \frac{2^6}{2^{520}} = 200 \times \frac{1}{2^{514}}$.
The case above could be adapted to Mersenne prime $p = 2^{p_0} - 1$. We have $2^{p_0} = 1 \pmod p$. As p_0 is a prime number the degree n of the external reduction polynomial is inversible modulo p_0: there exists n_0 such that $n \times n_0 = 1 \pmod{p_0}$. We can take $\gamma = 2^{n_0}$ as a root for $E = X^n - 2$ modulo p. Indeed $(2^{n_0})^n - 2 = 2^{n_0 \times n} - 2 = 0 \pmod p$.

To find the internal reduction polynomial of the form $M = 2^m \times X^k - 1$, we search for $m < \log(\rho)$ and $0 < k < n$ such that $m + n_0 \times k = 0 \pmod{p_0}$. Indeed $M(\gamma) = 2^m \times (2^{n_0})^k - 1 = 2^{m+n_0 \times k} - 1 = 0 \pmod p$.

It is likely that such an AMNS always exists by looking at the interval $p_0 - n_0$ knowing that $n_0 > p_0/n$ and will give an AMNS with internal reduction properties as describe above. The same strategy could be applied on generalized Mersenne prime by using adequate external reduction polynomial. For instance, with $p = 2^{192} - 2^{64} - 1$ we could take $E = X^3 - X - 1$ and $\gamma = 2^{64}$ as a root.

E. Influence of ϕ on the AMNS size

1) Generation of AMNS with different ϕ: According to the value of $\phi = 2^k$ (2^{32} or 2^{64}), the generated AMNS are more or less efficients. We denote by n_{opt}, the *optimal size for an AMNS*, namely the smallest size (in block number) such that all values from the field can be represented and by definition it is given by $n_{\text{opt}} = \frac{\log(p)}{k} + 1$. Other parameters such as λ, M, M' and ρ have influence on the AMNS performances but the most important is the AMNS size: $n = \deg(E)$. If $n = n_{\text{opt}}$ we cannot improve its value. In practice, it is not always possible to have $n = n_{\text{opt}}$, and we should find the smallest n.

The table I gives value of n_{opt} for $k = 64$ and $k = 32$ and the values of n we get for the AMNS generation of NIST [15] and Brainpool [23] elliptic curve arithmetics (of same size).

2) Why $\phi = 2^{64}$ leads to better AMNS for Elliptic Curves: When $\phi = 2^{32}$ the generated AMNS have always a size larger than the ideal one. We give here some hint of understanding.

Table II
OPTIMAL VALUES n_{opt} AND n DEPENDING OF THE SIZE OF p

size of p (in bits)	256	384	512
n $(k=64)$	5	7	10
n_{opt} $(k=64)$	5	7	9
n $(k=32)$	12	18	25
n_{opt} $(k=32)$	9	13	17

We look for an approximate lower bound of $f(\rho)$ as a function of n. We use $|\lambda| = 2$ because all higher values will increase $f(\rho)$. If $\rho_0 = 2\alpha - 1$, then $f(\rho_0) = 2\omega(\rho_0 - 1)$ but with ρ as a power of 2, $f(\rho) \geqslant 2\omega(\rho_0 - 1)$. Let us approximate ω by $2n$ (for $|\lambda| = 2$, $\omega = 2n - 1$), we will denote this operation $\omega \simeq 2n$ and apply it to the subsequent operations. Then $\alpha \simeq |\lambda| \times ||M||_1 \geqslant |\lambda| \times ||M||_2$ and $||M||_2 \leqslant \sqrt{(n)} p^{1/n}$ (Minkowski bound). We get the following identities:

$$\begin{aligned} f(\rho_0) &= 2\omega(\rho - 1), \\ f(\rho_0) &= 2^2\omega(\alpha - 1), \\ f(\rho_0) &\simeq 2^3 n(\alpha - 1), \\ f(\rho_0) &\simeq 2^3 n(|\lambda| ||M||_2), \\ f(\rho_0) &\simeq 2^4 n ||M||_2, \\ f(\rho_0) &\simeq 2^4 n^{3/2} p^{1/n}. \end{aligned} \tag{7}$$

And $f(\rho) \geqslant f(\rho_0)$ so $2^4 n^{3/2} p^{1/n}$ is an approximate lower bound for $f(\rho)$.

To have a working AMNS we need $f(\rho) < \phi$ so $\log(f(\rho)) \leqslant 32$ or 64. We need to find the smallest n such that $\log(2^4 n^{3/2} p^{1/n}) \leqslant 32$ or 64.

We give in table II the values of n, and the n optimal with $k = 32$ or 64. This explains why AMNS are more efficient with $\phi = 2^{64}$ than $\phi = 2^{32}$. The more p grows the more n needs to increase to keep $\log(2^4 n^{3/2} p^{1/n}) \leqslant 32$ or 64. And for a same p, n needs to increase more when $k = 32$ than when $k = 64$. By consequences AMNS with $\phi = 2^{64}$ are well adapted to elliptic curve cryptography of nowdays (but probably less for RSA primes).

III. DESIGN OF THE LIBRARY AND CHOICES FOR THE ARITHMETIC OF CURVES

The library `MPHELL` is logically built with abstraction layers. The first one concerns the low-level big number arithmetic, GMP or MbedTLS are available for this layer. The second one is the prime field arithmetic, either built from the big number layer or directly provided (e.g., AMNS, or Intel IPP[3] prime field arithmetic). Then comes the field arithmetic layer, only able to deal with prime field, quadratic and cubic extention of prime field (i.e., fields of size p^2 and p^3), used for conversion between elliptic curve settings. The elliptic curve layer is the last piece dealing with different types of coordinates systems, abstracting Weiestrass, Jacobi quartics and Edwards arithmetics and the conversion between them.

[3]Intel Integrated Performance Primitives

Only unified coordinates systems are used in order to provide countermeasure against SPA [14], [24]. A random number generator (RNG) is provided and allows to choose several DRBG[4] and also "true random sources" for the seed. An external hardware RNG can also be used, like the one from STM32F4 board.

The field arithmetic versatility allows to choose the best solution according to the target:

- AMNS is efficient when 128 bits integers are available,
- Intel IPP has good performance for NIST curves, but works only on Intel targets,
- GMP is a compromise between genericity and performances and we can build countermeasures over it.

This flexibility keeps `MPHELL` performing well on different targets. Like in other libraries we use different methods for signing and verifying since verification does not need to be protected against attacks whereas signing does. We choose for the signing to implement complete formulae with unified addition since other methods offers opportunities to some side channel attacks. Therefore we use a simple double and add algorithm for the signature. By design, the library can be modified by users to add some countermeasures such as coordinate randomizations (first introduced by [25]) which offer some beneficial protection [26].

A. Weierstrass curves

Weierstrass curves are mainly the ones found in the standards (e.g [15], [27]) but there is no explicit recommandation for the arithmetic. We propose therefore two implementations, one which is based on projectives coordinates and another one based on projective jacobian coordinates. For the projectives coordinates, we used the formulae from [28] for unified addition which costs $11\text{M}+6\text{S}+1\times a+10\text{A}+4\times 2+1\times 4$, as well as dedicated addition from [29, formula 3] which costs $12\text{M}+2\text{S}+6\text{A}+1\times 2$ and doubling from [28] which costs $7\text{M}+3\text{S}+5\text{A}+4\times 2+1\times 3$ in the favorable case of $a_4=-3$ and $5\text{M}+6\text{S}+1\times a+7$ A$+3\times 2+1\times 3$ otherwise. For the Jacobian Coordinates we used formulae from [30] which costs $8\text{M}+4\text{S}+15.5\text{A}$ per `ZADDC` operation (ladder) (see [30, section 4] for details), even if [31] provides faster operations but with the uncertainty of patents. For dedicated addition and doubling we adapted formulae from the ones of [32] which are similar to [28] but uses less additions. They cost $12\text{M}+4\text{S}+6\text{A}+1\times 2$ for addition and for doubling it costs $5\text{M}+3\text{S}+5\text{A}+4\times 2+1\times 3$ in the favorable case where $a_4=-3$ and $7\text{M}+3\text{S}+4\text{A}+4\times 2+1\times 3$ otherwise.

In the table III, we summarize the different costs of operations for different libraries [5], [33]–[38], neglecting the constants, and the pros and cons of `MPHELL` against common libraries [5], [33], [35]–[37]. Here are written M for multiplication, S for squaring and A for addition. The costs are computed according to the source code available of the different libraries (including `MPHELL`).

[4]Determinitic Random Bit Generator

B. Edwards curves

Since the work of [39] and later [40] the Edwards curves became widely used and standardized [15], [41]. Hence we present our implementations of Edward Curves. We use mainly the work of [42] for twisted extended edwards coordinates.

For a dedicated addition we use the formulae from [42, §3.2] which costs $9\text{M}+1\times a+7\text{A}$.

For a dedicated doubling we use the formulae from [42, §3.3] which costs $4\text{M}+4\text{ S}+1\times a+6\text{A}+1\times 2$.

For a unified addition we use the formulae from [42, §3.1] which costs $9\text{M}+1\times a+1\times d+7\text{A}$.

We also use formulae in (Edwards) projective coordinates from [43, §6] which costs $10\text{M}+1\text{S}+1\times a+1\times d+7\text{A}$.

For the dedicated doubling we use the formulae from [43, §6] which costs $3\text{M}+4\text{S}+1\times a+6\text{A}+1\times 2$. We only present in Table III `libSodium` [38] since other libraries ([5], [33], [37]) follow the same methods and others uses non performant formulae [34], [35].

C. Jacobi Quartic curves

The formulae are made only for the extended homogeneous projective coordinates. We propose a unified addition from [44] which costs $8\text{M}+2\text{S}+13\text{A}$ and a dedicated doubling which costs $3\text{M}+5\text{S}+1\times a+3\text{A}$.

We have summarized all the formulae used in `MPHELL` in table IV.

D. Scalar multiplication

1) Protected: For protected scalar multiplication we use a naive implementation with a double and add with unified formulae except for the case of weierstrass curve with jacobian coordinates where we use formulae from [30].

We have summarized the methods in the table V.

2) Unprotected: For a general scalar multiplication we use the method of sliding window with varying size w according to the size l of the scalar. Hence we choose $w=3$ for $l\leqslant 70$, $w=4$ for $l\leqslant 197$, $w=5$ for $l\leqslant 539$ and $w=6$ otherwise. Our method differs from [45, Algorithm 3.38] as we do not perform substraction of points. Therefore for those formulae we used the dedicated addition and doubling presented earlier.

E. Verification of signatures

For the verification we use a formula like the one of [45, Algorithm 3.48] (used for example in [32]) which uses also a sliding window method but which does a simultaneous doubling for the two points.

We have summarized the method in the table VI, libraries that use same formula for protected and unprotected version are not listed here.

IV. Experimental results and comparison with other libraries

A. Results on x86-64 architectures

In the figure 2 ECDSA and EdDSA signatures[5] timings are shown for different elliptic curves without taking

[5]Average number of signatures per second, calculated over 10 000 ECDSA signatures

Table III
COST OF ADDITIONS AND DOUBLINGS OPERATIONS ON $\mathbb{F}_p$ IN [5], [33]–[38]

Library	Curve	Addition	Doubling	Coordinates
OpenSSL	SECP224R1	12M+4S+6A+1 × 2 if $Z_2 \neq 1$ 9M+3S+6A+1 × 2 if $Z_2 = 1$	3M+5S+8A, +1 × 3 + 1 ×4 + 2 ×8	Projective Jacobian standard
OpenSSL	SECP256R1	10M+5S+9A+4 × 2 if $Z_2 \neq 1$ 7M+4S+6A+5 × 2 if $Z_2 = 1$	3M+5S+8A +1 × 3 + 1 × 4 + 2 × 8	Projective Jacobian standard
OpenSSL	SECP256R1 & SECP521R1	12M+4S+6A+1 × 2 if $Z_2 \neq 1$ 8M+3S+6A	4M+4S+5A+3 × 2 +1 × 3 + 1 ÷ 2	Projective Jacobian standard
libreSSL	Standard weierstrass curves	12M+4S+8A+1 × 2	3M+4S+5 A+7 × 2 if $a_4 = -1$ 4M+4S+6A+8 × 2 if $a_4 = -3$ 4M+6S+5A+8 × 2	Projective coordinates
MbedTLS	Weierstrass	8M+3S+6A+1 × 2	3M+4S+4A+1 × 3 if $a_4 = 0$ 4M+4S+6A+1 × 3 if $a_4 = -3$ 3M+6S+5A+1 × 3 + 1 × a_4 otherwise	Mixt for add Jac. for doubling
libgcrypt	Weierstrass	10M+4S+7A+1 × 2 + 1 × 3	7M+6S+4A+2 × 2 if $a_4 \neq -3$ 7M+5S+5A+2 × 2 if $a_4 = -3$	Jacobian Projective
IntelIPPCP	NIST and SM2 Curves All curves	12M+4S+6A+1 × 2 8M+3S+6A+1 × 2	4M+4S+4A+2 × 2 + 1 × 3 + 1 ÷ 2 3M+6S+4A+1 × a_4 + 2 × 2 + 1 × 3 + 1 × 4 + 1 × 8	Jacobian Jacobian Jacobian and affine
wolfSSL	Weierstrass	12M+4S+21A+1 optional ÷2	3M+6S+20A+1 × a_4	Projective
wolfSSL	Weierstrass	9M+3S+21A+1 optional ÷2 when $Z_2 = 1$	4M+4S+22A if $a_4 = -3$	Projective
libSodium	Ed25519 Ed25519	7M+7A 7M+7A	3M+4S+5A+1 × 2 4M+4S+5A+1 × 2	Extended with prec. Extended

Table IV
COST OF ADDITIONS AND DOUBLINGS IN OPERATIONS ON $\mathbb{F}_p$ IN MPHELL

Curve	Addition	Doubling	Coordinates
Weierstrass	11M+6S+10A	same as addition	Projective
Weierstrass	12M+2S+6A	7M+3S+5A 5M+6S+7A	Projective
Weierstras	12M+4S+6A	5M+3S+5A 7M+3S+4A	Jacobian
	8M+4S+15.5A per ZADDC		
Jacobi Quartic	8M+2S+13A	3M+5S+3A	Jacobian
Edwards	9M+7A	4M+4S+6A	Extended
Edwards	9M+7A	Same as addition	Extended
Edwards	10M+1S+7A	3M+4S+6A	Projective

Table V
METHODS USED FOR SCALAR MULTIPLICATION IN ECDSA SIGNATURES

Library	Curve	Method
Mbed TLS	Weierstrass	Comb method with $(\log_2(p) < 384)?w = 4 : w = 5$, [45, Alg. 3.44]
Mbed TLS	Montgomery	Montgomery ladder [45, Alg. 3.40]
OpenSSL	NISTP curves	Inteleaved point multiplication [46, §3.3]
OpenSSL	NISTz256	WNAF with $w = 7$
LibreSSL	Weierstrass & Montgomery	Montgomery ladder
libgcrypt	Weierestrass & Edwards	Always double and add [45, Alg.3.27]
Intel	NIST	Fixed base NAF with $(\log(p) < 384)?w = 7 : w = 5$ [45, Alg 3.42]
libSodium	Ed 25519	Fix based NAF with $w = 5$
wolfSSL	Weierstrass	Joye ladder [47]
MPHELL	Weierstrass	Method from [30]
MPHELL	Edwards & Jacobi	Unified double and add

into account specificity of the curves (as in OpenSSL for instance) on x86-64 architectures. The libraries used for the comparisons are: MPHELL(v5.0.0), OpenSSL [5](v3.0.0), MbedTLS [34](v2.25), LibECC, Intel IPPCP [36](v2020.0.0), Miracl [48](v7.0.0),LibGCrypt [49](v1.8.5), libSodium [38](v1.0.16),LibreSSL [33](v3.3.1). For the signature MPHELL performs well despite the unified operation (protection against Simple Power Analysis). Intel IPP is faster for the NIST curves where it uses some dedicated arithmetic based on the pseudo Mersenne prime specificities. libSodium is also faster but dedicated to the Ed25519 curve and without protection against SPA. For the verification in the figure 3, which requires no protection, MPHELL is faster on all curve except for the Ed25519 where libSodium is still the best.

Table VI
METHODS USED FOR SCALAR MULTIPLICATION IN ECDSA VERIFICATIONS

Library	Curve	Method
OpenSSL	NISTP curves	window method with $w = 5$
OpenSSL	NISTz256	window NAF with $w = 5$
LibreSSL	Weierstrass & Montgomery	window NAF with $\log(p) >= 300?w = 4 : w = 3$
libgcrypt	Edwards	Double and add
libgcrypt	Weierstras	binary NAF [45]
Intel	NIST	windows method with $w = 5$
libSodium	Ed 25519	sliding window method with $w = 5$ [45, Alg.3.38]
wolfSSL	Weierstrass	simultaneous point multiplication [45, Alg 3.48] with $w = 5$
MPHELL	All curves	sliding window with $(\log(p) \leqslant 529)?w = 5 : w = 6$

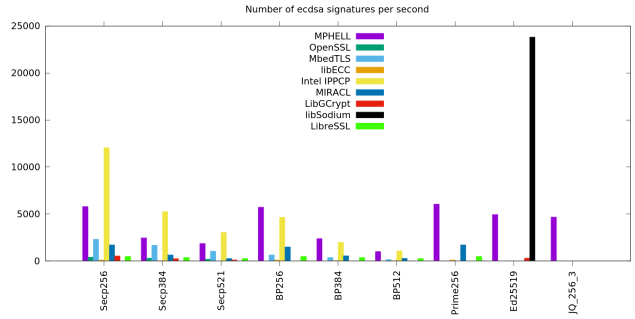

Figure 2. Comparison between MPHELL and other libaries on x86 64 bits, for ECDSA/EdDSA signatures

The figures 4 and 5 show the performance of MPHELL under AMNS, as described in §II, with respect to GMP and Intel IPP for the field arithmetic layer on x86 (64 bits) and illustrate the pertinency of this approach.

B. Results on embedded devices (ARM and STM32)

The following timings show the performance of our library and arithmetic choices on ARM v8 (Raspberry 4) and on STM32F4 for ECDSA signatures and verifications and compare it to MbedTLS [34], which is a reference SSL/TLS library for embedded devices based on ARM architectures. On Raspberry Pi 4 (Figures 6 and 7), MPHELL is faster than MbedTLS [34] for the signature, except for some NIST

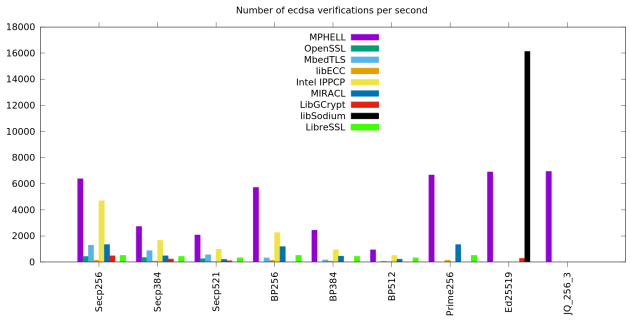

Figure 3. Comparison between MPHELL and other libaries on x86 64 bits, for ECDSA/EdDSA verifications

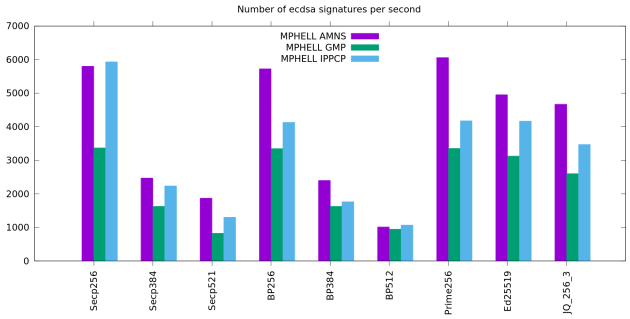

Figure 4. Comparison between MPHELL field arithmetics on x86 (64 bits), for ECDSA/EdDSA signatures

curves, where MbedTLS has a dedicated arithmetic. For the verification MPHELL is always faster. On this target GMP is the best option for the field arithmetic. Indeed the 128 bits integers are not available and the poor quality of the AMNS defined with $\phi = 2^{32}$ leads to poor performances.

On the STM32F4 (Figures 8 and 9), despite the small memory available, MPHELL works well. For the signature MbedTLS has faster timings for the NIST curves where it uses some dedicated arithmetic based on the pseudo Mersenne prime specificities.

Nevertheless MPHELL has the best performances for the

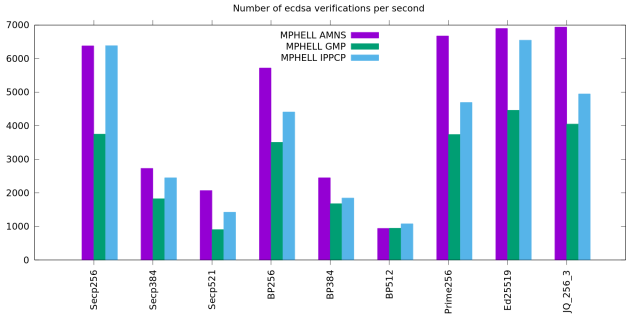

Figure 5. Comparison between MPHELL field arithmetics on x86 (64 bits), for ECDSA/EdDSA verifications

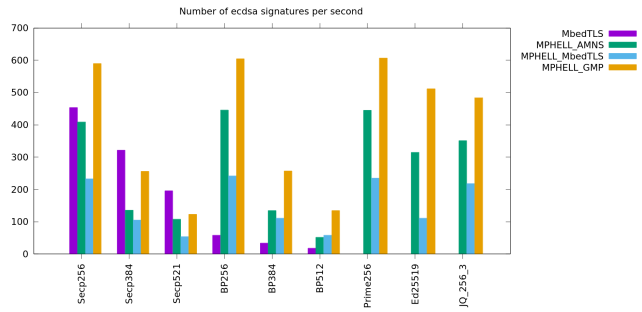

Figure 6. Comparison between MPHELL (differents field arithmetics) and MbedTLS on Rasperry PI 4, for ECDSA/EdDSA signatures

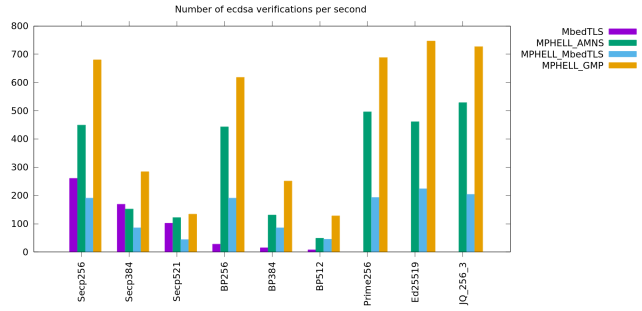

Figure 7. Comparison between MPHELL (differents field arithmetics) and MbedTLS on Rasperry PI 4, for ECDSA/EdDSA verifications

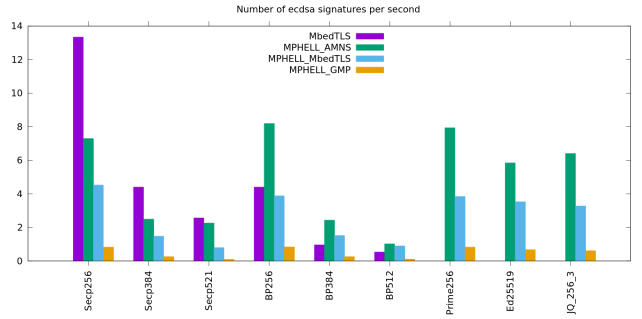

Figure 8. Comparison between MPHELL (differents field arithmetics) and MbedTLS on STM32F4, for ECDSA/EdDSA signatures

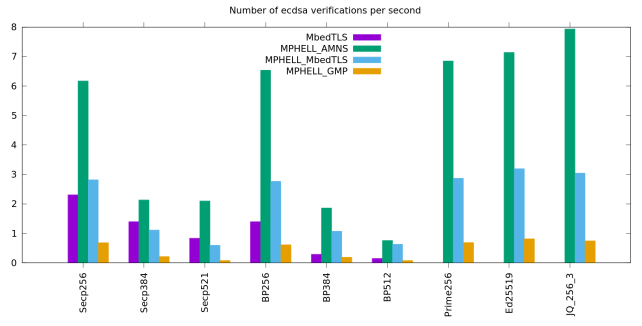

Figure 9. Comparison between MPHELL (differents field arithmetics) and MbedTLS on STM32F4, for ECDSA/EdDSA verifications

other curves, even with the SPA protection. We can see, surprisingly that AMNS gives competitive timings on STM32F4 despite the fact that 128 bits integers are not available and that the AMNS used are by consequent far from the ideal ones. For the verification MPHELL is currently the fastest.

V. Conclusion

We have seen that MPHELL , using our improvement on AMNS, can be efficient. It is the fastest for ECDA signatures verifications. On STM32F4 board, we have shown that our AMNS implementation offers competitive timings despite the fact that we do not have optimal parameters in this case. We have also seen that MPHELL offers overall competitive performances when it uses SPA resistant unified addition. But alone this method could not be enough when dealing general SCA [2]–[4] or with Machine Learning attacks [50], [51]. We know from [52], that randomized arithmetics could be used to provide countermeasures against such attacks, as also shown in [53]. The abstract layer of MPHELL allows for such implementation of randomized arithmetics for AMNS or other MNS and will be the subject of a forthcoming work including Machine Learning attacks.

Acknowledgment

The authors are grateful to the referees for careful reading of the paper and valuable suggestions and comments. This work has been partially supported by ANR-15-IDEX-02, ANR ARRAND (ANR-15-CE39-0002), SECURIOT-2-AAP FUI 23, ANR-19-FLJO-0001 EASIMoB and ANR-11-LABX-0025-01.

References

[1] J. Van de Pol, N. P. Smart, and Y. Yarom, "Just a little bit more," in *Cryptographers' Track at the RSA Conference*. Springer, 2015, pp. 3–21.

[2] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom, "Ecdsa key extraction from mobile devices via nonintrusive physical side channels," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1626–1638.

[3] D. Genkin, L. Pachmanov, I. Pipman, and E. Tromer, "Stealing keys from pcs using a radio: Cheap electromagnetic attacks on windowed exponentiation," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 207–228.

[4] ——, "Ecdh key-extraction via low-bandwidth electromagnetic attacks on pcs," in *Cryptographers' Track at the RSA Conference*. Springer, 2016, pp. 219–235.

[5] T. O. S. project, "Openssl library," 1988. [Online]. Available: https://www.openssl.org

[6] G. GnuPG, "Gnu privacy guard," 1998. [Online]. Available: https://www.gnupg.org/

[7] P. Y. Liardet and N. P. Smart, "Preventing spa/dpa in ecc systems using the jacobi form," in *Cryptographic Hardware and Embedded Systems — CHES 2001*, c. K. Koç, D. Naccache, and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, p. 391–401.

[8] M. Joye and J.-J. Quisquater, "Hessian elliptic curves and side-channel attacks," in *Cryptographic Hardware and Embedded Systems — CHES 2001*, c. K. Koç, D. Naccache, and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, p. 402–410.

[9] O. Billet and M. Joye, "The jacobi model of an elliptic curve and side-channel analysis," in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, M. Fossorier, T. Høholdt, and A. Poli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, p. 34–42.

[10] D. Stebila and N. Thériault, "Unified point addition formulæ and side-channel attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, L. Goubin and M. Matsui, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, p. 354–368.

[11] R. R. Goundar, M. Joye, A. Miyaji, M. Rivain, and A. Venelli, "Scalar multiplication on weierstraß elliptic curves from co-z arithmetic," *Journal of cryptographic engineering*, vol. 1, no. 2, p. 161, 2011.

[12] J. Plût, "On various families of twisted jacobi quartics," in *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, ser. Lecture Notes in Computer Science, A. Miri and S. Vaudenay, Eds., vol. 7118. Springer, 2011, pp. 373–383.

[13] S. Ghosh, A. Kumar, A. Das, and I. Verbauwhede, "On the implementation of unified arithmetic on binary huff curves," in *Cryptographic Hardware and Embedded Systems - CHES 2013*, G. Bertoni and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, p. 349–364.

[14] S. Pontie, "Sécurisation matérielle pour la cryptographie à base de courbes elliptiques," Ph.D. dissertation, Université Grenoble Alpes, 2016.

[15] F. NIST, "Fips 186-5–digital signature standard (dss)(draft)," 2019.

[16] T. Granlund and the GMP development team, *GNU MP: The GNU Multiple Precision Arithmetic Library*, 2020, version 6.2.1. [Online]. Available: https://gmplib.org/

[17] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[18] J.-C. Bajard, L. Imbert, and T. Plantard, "Modular number systems: Beyond the mersenne family," pp. 159–169, 2005.

[19] L. Didier, F. Dosso, and P. Véron, "Efficient modular operations using the adapted modular number system," *J. Cryptogr. Eng.*, vol. 10, no. 2, pp. 111–133, 2020.

[20] M.-A. Cornelie, "Implantations et protections de mécanismes cryptographiques logiciels et matériels," Theses, Université Grenoble Alpes, Apr. 2016.

[21] F.-Y. Dosso, "Contribution de l'arithmétique des ordinateurs aux implémentations résistantes aux attaques par canaux auxiliaires." Theses, Université Toulon, Jun. 2020.

[22] C. Negre and T. Plantard, "Efficient modular arithmetic in adapted modular number system using lagrange representation," *In Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia, pages 463–477*, 2008.

[23] M. Lochter and J. Merkle, "Rfc 5639-elliptic curve cryptography (ecc) brainpool standard-curves and curve generation," *Federal Office for Information Security of the German Federal Republic, secunet Security Networks, IETF*, 2010.

[24] E. Brier and M. Joye, "Weierstraß elliptic curves and side-channel attacks," in *International Workshop on Public Key Cryptography*. Springer, 2002, pp. 335–345.

[25] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *International workshop on cryptographic hardware and embedded systems*. Springer, 1999, pp. 292–302.

[26] N. Mukhtar, A. P. Fournaris, T. M. Khan, C. Dimopoulos, and Y. Kong, "Improved hybrid approach for side-channel analysis using efficient convolutional neural network and dimensionality reduction," *IEEE Access*, vol. 8, pp. 184 298–184 311, 2020.

[27] "IEEE Standard Specifications for Public-Key Cryptography," *IEEE Std 1363-2000*, pp. 1–228, 2000.

[28] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Asiacrypt*, vol. 4833. Springer, 2007, pp. 29–50.

[29] H. Cohen, A. Miyaji, and T. Ono, "Efficient elliptic curve exponentiation using mixed coordinates," in *Asiacrypt*, vol. 98. Springer, 1998, pp. 51–65.

[30] K. H. Kim, J. Choe, S. Y. Kim, N. Kim, and S. Hong, "Speeding up elliptic curve scalar multiplication without precomputation." *IACR Cryptology ePrint Archive*, vol. 2017, p. 669, 2017.

[31] M. Hamburg, "Faster montgomery and double-add ladders for short weierstrass curves," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 189–208, 2020.

[32] *Intel Integrated Performance Primitives Cryptography Developer Reference*, Intel, 2017, intel Integrated Performance Primitives 2018.

[33] "Libressl library," 2014. [Online]. Available: https://www.libressl.org

[34] A. Ltd, "mbed tls," 2015. [Online]. Available: https://tls.mbed.org/

[35] W. Koch *et al.*, "Libgcrypt cryptographic library," Aug 1998. [Online]. Available: https://gnupg.org/software/libgcrypt/

[36] I. corporation, "Intel integrated performance primitives cryptography." [Online]. Available: https://github.com/intel/ipp-crypto/releases
[37] T. Ouska, "wolfssl," 2006.
[38] F. Denis, "The sodium cryptography library," Jun 2013. [Online]. Available: https://download.libsodium.org/doc/
[39] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, pp. 1–13, 2012.
[40] M. Hamburg, "Ed448-goldilocks, a new elliptic curve." *IACR Cryptology ePrint Archive*, vol. 2015, p. 625, 2015.
[41] S. Josefsson and I. Liusvaara, "hrefhttps://tools.ietf.org/html/rfc8032Edwards-curve digital signature algorithm (EdDSA)," in *Internet Research Task Force, Crypto Forum Research Group, RFC*, vol. 8032, 2017.
[42] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson, "Twisted edwards curves revisited." in *Asiacrypt*, vol. 5350. Springer, 2008, pp. 326–343.
[43] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted edwards curves," in *International Conference on Cryptology in Africa*. Springer, 2008, pp. 389–405.
[44] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson, "Jacobi quartic curves revisited," in *Australasian Conference on Information Security and Privacy*. Springer, 2009, pp. 452–468.
[45] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer Science & Business Media, 2006.
[46] E. Käsper, "Fast elliptic curve cryptography in open ssl," in *International Conference on Financial Cryptography and Data Security*. Springer, 2011, pp. 27–39.
[47] M. Joye, "Highly regular right-to-left algorithms for scalar multiplication," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 135–147.
[48] M. Scott, "Miracl library," 2005. [Online]. Available: https://miracl.com/
[49] W. Koch and M. Schulte, "The libgcrypt reference manual," *Free Software Foundation Inc*, pp. 1–47, 2005.
[50] L. Weissbart, S. Picek, and L. Batina, "One trace is all it takes: Machine learning-based side-channel attack on eddsa," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2019, pp. 86–105.
[51] G. Perin, Ł. Chmielewski, L. Batina, and S. Picek, "Keep it unsupervised: Horizontal attacks meet deep learning," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 343–372, 2021.
[52] J. Courtois, L. A. Abbas-Turki, and J.-C. Bajard, "Resilience of randomized RNS arithmetic with respect to side-channel leaks of cryptographic computation," *IEEE Transactions on Computers*, vol. 68, no. 12, pp. 1720–1730, Jun. 2019.
[53] N. Mukhtar, L. Papachristodoulou, A. Fournaris, and L. Batina, "Machine-learning assisted side-channel attacks on rns-based elliptic curve implementations," *Transactions on Cryptographic Hardware and Embedded Systems*, 2019.