Finite field arithmetic in large characteristic for classical and post-quantum cryptography

Sylvain Duquesne

Finite field arithmetic in large characteristic for classical and post-quantum cryptography*

Sylvain Duquesne

Univ Rennes, CNRS, IRMAR - UMR 6625, F-35000 Rennes, France
sylvain.duquesne@univ-rennes1.fr
https://perso.univ-rennes1.fr/sylvain.duquesne/

Abstract. Both classical and post-quantum cryptography massively use large characteristic finite fields or rings. Consequently, basic arithmetic on these fields or rings (integer or polynomial multiplication, modular reduction) may significantly impact cryptographic devices' efficiency and power consumption. In this paper, we will present the most used and the less common methods, clarify their advantages and drawbacks and explain which ones are the more relevant depending on the implementation context and the chosen cryptographic primitive. We also explain why recent proposals such as RNS, PMNS or Montgomery-friendly primes may be a good alternative to classical methods depending on the context and suggest directions for further research to improve them.

Keywords: Finite Field · Arithmetic · Cryptography · Modular reduction · Multi precision · Polynomial rings

Introduction

Most of the public key cryptosystems use large finite fields or rings as well as their polynomial extensions. The consequence of this massive usage is that basic arithmetic on these fields and rings may impact the efficiency and power consumption of cryptographic devices. Nevertheless, the sizes and degrees involved are very diverse. For example

- Discrete logarithm on the multiplicative subgroup of a finite field, as well as RSA, uses 1024 to 4096-bits integers.
- Elliptic curve cryptography uses 256 to 512-bits prime fields.
- Pairing-based cryptography uses 256 to 1024-bits prime fields together with small degrees extension fields.
- Isogeny-based post-quantum cryptography uses 400 to 1000-bits prime fields and quadratic extensions.
- Lattice-based post-quantum cryptography uses 13 to 60-bits prime fields and large-degree polynomial rings.

* This work was supported in part by French project ANR-11-LABX-0020-01 "Centre Henri Lebesgue"

Because of this large spectrum, there is a wide collection of algorithms for performing finite field or ring arithmetic which have each their range of interest depending on the context. Many arithmetic operations (squarings, inversions, additions, Frobenius map, square roots, ...) are involved in various protocols. In this paper, we will concentrate on multiplications and modular reduction because there are the most impacting operations in practice.

We will present the most used methods (schoolbook, interpolation based, Montgomery) as well as less common ones (RNS, PMNS) and some that are specific to particular modules (Mersenne and pseudo-Mersenne primes, Montgomery friendly primes). We will discuss their advantages and drawbacks and explain which ones are the most relevant depending on the implementation context and the chosen cryptographic primitive. We will detail how they can be combined and in which direction they should or could be improved in further works. For some of them, we give some trails for such improvements. We also explain to what extent recent techniques (PMNS, Montgomery-friendly primes) are competitive with more classical ones and spotlight their advantages depending on the context.

The paper first recalls multi-precision arithmetic and then concentrates on modular reduction methods and polynomial operations.

1 Multiprecision and large integer multiplication

Hardware devices have the native capacity to perform arithmetic operations on bounded inputs, usually the processor's word size (denoted w in this paper). This word size is classically between 8 and 128 bits; most common architectures use 32 or 64-bit words nowadays. However, classical cryptography and isogeny-based cryptography use much larger integers. Multiprecision is the way a device deals with such large integers. It consists in writing large integers in base β (where $\beta = 2^w$) and then using specific algorithms to get the result of a specific operation involving large inputs.

For example, to add 2 n-words integers $a < \beta^n$ and $b < \beta^n$, we have to add them word by word

$$a + b = \sum_{i=0}^{n-1} a_i \beta^i + \sum_{i=0}^{n-1} b_i \beta^i = \sum_{i=0}^{n-1} (a_i + b_i)\beta^i \tag{1}$$

Unfortunately, this is not so easy because of carries. Indeed, $a_i + b_i$ may be greater than or equal to β. The right formula is then

$$a + b = \sum_{i=0}^{n-1} a_i \beta^i + \sum_{i=0}^{n-1} b_i \beta^i = \sum_{i=0}^{n} (a_i + b_i + q_{i-1} \bmod \beta)\beta^i$$

where $q_{i-1} = 0$ or 1 is the quotient of the Euclidean division of $a_{i-1} + b_{i-1} + q_{i-2}$ by β. Contrary to Equation (1), $a + b$ is now written in base β because its coefficients are less than β.

The carry is just 0 or 1, so it is not a big deal to add it to the next coefficient, and the complexity of multi-precision addition remains linear in n. Nevertheless, its potential propagation makes difficult the parallelization of this naive algorithm. Moreover, parallelization is very important today because multi-core devices are very current. It is possible to partially get around this problem using carry-save variants of this algorithm. Their concept is to delay and accumulate carries to the end of the computation [67].

1.1 Schoolbook multiplication

We can follow the same process for multiplying 2 large integers. Let us consider a and b, two n-word integers written in base β. Then ab can be computed by a succession of basic word multiplications and additions in the way pupils learn multiplication in base 10 at school (which gives the name to the method)

$$\sum_{i=0}^{n-1} a_i \beta^i \sum_{i=0}^{n-1} b_i \beta^i = \sum_{i=0}^{2n-2} c_i \beta^i \text{ with } c_i = \sum_{k+l=i} a_k b_l \tag{2}$$

The complexity is n^2 word multiplications but again ab is not written in base β in formula (2) because c_i will generally overcome β. In order to take carries into account, we use Algorithm 1.
As was the case for multiprecision addition, the complexity is not affected (n^2 basic word multiplications) but managing carries makes this algorithm difficult to parallelize again. Note that if a specific squaring algorithm is used, it will be more efficient than using Algorithm 1 because if $a = b, a_i b_j$ and $a_j b_i$ are the same and then are computed only once.

1.2 Karatsuba multiplication

There is a well-known way to reduce the number of basic multiplications required for multiplying large integers. It is due to Karatsuba [52]. To multiply 2-word integers with the schoolbook method, we do

$$(a_0 + a_1\beta)(b_0 + b_1\beta) = a_0 b_0 + (a_0 b_1 + a_1 b_0)\beta + a_1 b_1 \beta^2$$

Karatsuba's method consists in computing the middle term as

$$a_0 b_1 + a_1 b_0 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$$

This saves one multiplication (at the cost of 3 extra additions) because $a_0 b_0$ and $a_1 b_1$ are already computed as the constant term and the term in β^2, respectively. This process can be recursively applied to deal with larger integers (the

Algorithm 1: Schoolbook multiplication of large integers

Input: 2 integers $0 \leq a, b < \beta^n$ written in base $\beta = 2^w$:
$a = \sum_{i=0}^{n-1} a_i \beta^i, b = \sum_{i=0}^{n-1} b_i \beta^i$, with $0 \leq a_i, b_i < \beta$

Result: $r = ab < \beta^{2n}$ written in base β: $r = \sum_{i=0}^{2n-1} r_i \beta^i$, with $0 \leq r_i < \beta$

```
r_i ← 0  ∀i ∈ [0..2n − 1]
for i = 0 to n − 1 do
    carry ← 0
    for j = 0 to n − 1 do
        t ← r_{i+j} + a_j b_i + carry
        carry ← t quo β
        r_{i+j} ← t mod β
    end for
    r_{i+j+1} ← carry
end for
return {r_i}_{i=0..2n−1}
```

well-known divide-and-conquer strategy). We then get the schoolbook method's complexity of $n^{\log_2 3}$ basic word multiplications instead of n^2. The main drawback is that the number of addition will grow much faster than the number of multiplications will decrease.

Then, depending on the context and the relative cost of additions and multiplications, the schoolbook or the Karatsuba method should be preferred. More precisely, the Karatsuba method can be used for the first steps of the divide-and-conquer strategy for large integers, but it is not necessarily adequate for the last steps. Indeed, operands are smaller so that the relative cost of additions is higher than those of multiplications. Then, in practice, both methods are combined. The threshold for switching from Karatsuba to a schoolbook depends on the implementation context. More details on this shallow Karatsuba approach and the choice of the threshold can be found in [69, 75]. For example, for the current ECC key sizes, Karatsuba approach is not competitive for software implementation [44].

Many other ways exist to improve the complexity of large integer multiplications at the cost of extra additions or other small operations. We do not present them here because there are not used in cryptography (their threshold of use is too far regarding the sizes of integers used in cryptography), but also because we will give them in Section 4 in the polynomial context, which is more straightforward (in particular, no carries are involved).

2 Modular reduction in the general case

The specificity of the finite field arithmetic is the reduction step modulo the characteristic p of the field. However, note that the algorithms for reducing modulo p presented in this section do not require p to be prime and are indeed not always used in cryptography for primes (e.g. for RSA ciphering).

2.1 Schoolbook reduction

The reduction of an integer modulo p consists in finding the remainder of the Euclidean division of this integer by p. Again, the easiest way is to use the algorithm learned at school. Assume we want to reduce a m-word integer a modulo a n-word integer p (with of course $m \geq n$) written in base β:

$$a = \sum_{i=0}^{m-1} a_i \beta^i \text{ and } p = \sum_{i=0}^{n-1} p_i \beta^i$$

We assume that we have at our disposal a basic division of a 2-word integer by a 1-word integer, provided that the quotient is a 1-word integer. The principle of the schoolbook division is essentially to divide the main significant word a_{m-1} of a by the one of p (or the 2 main ones if $a_{m-1} < p_{n-1}$). The quotient q gives a first approximation of the result, and a is updated as $a - qp$ (up to some powers of β), and the process is iterated until a becomes less than p. This process is precise in Algorithm 2.

Algorithm 2: Schoolbook reduction

Input: $a = \sum_{i=0}^{m-1} a_i \beta^i$ and $p = \sum_{i=0}^{n-1} p_i \beta^i$
Result: a reduced modulo p
  Compute q_0, r_0 s.t. $a_{m-1} = p_{n-1} q_0 + r_0$ with $r_0 < p_{n-1}$
  $a \leftarrow a - q_0 p \beta^{m-n}$
  for $i = 1$ **to** $m - n$ **do**
    Compute q_i, r_i s.t. $a_{m-i}\beta + a_{m-i-1} = p_{n-1} q_i + r_i$ with $r_i < p_{n-1}$
    $s \leftarrow a - q_i p \beta^{m-n-i}$
    while $s < 0$ **do**
      $q_i \leftarrow q_i - 1$ $s \leftarrow s + p\beta^{m-n-i}$
    end while
    $a \leftarrow s$
  end for
  return a

The loop of this algorithm is $m-n+1$-long and each step is, up to additions, made of one basic division (computing q_i and r_i) and n basic multiplications

(multiplying the 1-word integer q_i by the n-word integer p). The overall complexity is then $m-n+1$ basic divisions and $n(m-n+1)$ basic multiplications. If a is the result of a multiplication in $\mathbb{F}_p$, we have $m = 2n$, and the complexity is then n divisions and n^2 multiplications. This is not so bad compared to multiplication. However, basic divisions are usually very expensive, so we want to avoid them. It can be, for example, done thanks to the Newton method applied to the function $f(x) = \frac{1}{x} - b$, which computes the inverse of b without any division. However, the best way to do it is to use the so-called Montgomery or Barrett reductions.

2.2 Montgomery reduction

During the '80s, Peter Montgomery [58] and Paul Barrett [18] introduce new methods to perform a Euclidean division without basic divisions. Their approach are slightly different but the principle is the same: write $\frac{a}{p}$ as $\frac{a}{\beta^n}\frac{\beta^n}{p}$. If the second term is precomputed (it does not depend on a), the division by p is then replaced by a division by β^n, which is trivial in base β. Of course, this approach works for real divisions, not directly for Euclidean ones. Then, they need to consider correcting terms to get the right result. Barrett and Montgomery's approaches have their advantages and drawbacks. The main drawback of Montgomery reduction is that it requires changing the representation of numbers, but it is mitigated if several operations are performed successively. This is classically the case in cryptographic primitives, so the Montgomery reduction is the most popular in this domain, and we will detail it in this subsection.

To simplify, we assume we want to reduce modulo p the result a of multiplication of 2 integers less than p (which is the most common use case). The basic version of the Montgomery reduction is given by Algorithm 3.

Algorithm 3: Montgomery reduction modulo p

Input: $a < p\beta^n$, $\beta^{n-1} \leq p < \beta^n$
the precomputed value $p' = -p^{-1} \bmod \beta^n$
Result: $r < 2p$ such that $r = a\beta^{-n} \bmod p$
$q \leftarrow ap' \bmod \beta^n$
$r \leftarrow (a + qp)/\beta^n$

It is easy to prove that the output $r < 2p$ and $r = a\beta^{-n} \bmod p$. But these two properties are not exactly the expected ones for the reduction of a mod p. For the first one, we can of course subtract p if necessary to get $r < p$. As an alternative, the output of Algorithm 3 can be used directly as input for the next step by adding a condition on p, specifically $4p < \beta^n$.

The fact that $r = a\beta^{-n}$ instead of a modulo p can be overcome by using the so-called Montgomery representation of numbers defined by $\overline{x} = x\,\beta^n \bmod p$. This representation is, of course, stable for the addition ($\overline{x}+\overline{y} = \overline{x+y}$), but it is also stable for the modular multiplication using Algorithm refalgo:Montgo. Indeed, when we multiply x and y in Montgomery representation, we get a β^{2n} factor. But β^n is removed during the Montgomery reduction process, so we finally get the Montgomery representation of xy and we then have $\overline{x}\,\overline{y} = \overline{x\,y} \bmod p$. Thanks to this stability, we can chain several operations in Montgomery representation. Changing representation may only be necessary at the beginning and end of the complete cryptographic computation. Moreover, this change of representation can be easily obtained thanks to Algorithm 3. Indeed, the Montgomery representation of $x < p$ is obtained with $x\,(\beta^{2n} \bmod p)$ as an input. Similarly, we can recover x using Algorithm 3 with $\overline{x}$ as an input. Hence, we usually ignore the conversion cost from Montgomery to classic representation (and reciprocally) in cryptographic applications.

Concerning the complexity, 2 multiplications are involved in the algorithm, but there are incomplete ones. The first one (ap') is reduced mod β^n, so it is unnecessary to compute its most significant part. For the second one, we know in advance that $a+qp$ will be exactly divisible by β^n (by definition of q), so its least significant part is zero and may not need to be computed. The other operations in Algorithm 3 have negligible cost in base β, so the overall cost should be the one of only one full multiplication, say n^2. Nevertheless, this analysis does not take carries into account. For example, the least significant part of the second operation is zero but may produce some carries, so it must be computed anyway. For concrete applications and precise complexity analysis, we use Algorithm 4: a word-by-word version which requires n^2+n basic word multiplications [27, 29].

Algorithm 4: Word version of the Montgomery reduction modulo p

Input: $a < p\beta^n$, $\beta^{n-1} \le p < \beta^n$
the precomputed value $p' = -p^{-1} \bmod \beta$
Result: $r = a\beta^{-n} \bmod p$ and $0 \le r < p$

```
r ← a;
for i = 0 to n − 1 do
    r_0 ← r mod β
    q ← p' r_0 mod β
    r ← (r + qp)/β
end for
r' ← r − p + β^n
if r' ≥ β^n then                                   // else r < p
    r ← r' mod β^n                          // r ← r − p if r ≥ p
return r
```

There also exists a version of Algorithm 4 that is interleaved with the multiplication steps ([27, 56]) to get a complete modular multiplication algorithm. It has some advantages (smaller intermediate results) and drawbacks (only compatible with schoolbook multiplication). We do not give it here, but details can be found in chapter 5 of [42], for example.

The Montgomery reduction has been used mainly in cryptography for decades as long as the module does not have a specific form allowing fast reduction. The only constraint is that p is odd so that it can be inverted modulo β. It is, for example, regularly used in RSA implementations, Elliptic Curve Cryptography, pairing-based protocols or, more recently, isogeny key exchanges or lattice-based cryptography using primes modules. For example, the reference and optimized implementation of the NIST candidates Frodo and NTRUprime for post-quantum standardization [61] are using Montgomery reduction. However, we still have carries issues with this technique. To avoid them, one can use the well-known Chinese reminder theorem.

2.3 The Residue Number Systems (RNS)

The principle of Residue Number Systems (RNS) is to represent an integer a by its residues $(a_1, a_2, \ldots, a_n)$ modulo a set of coprime numbers $\mathcal{B} = (m_1, m_2, \ldots, m_n)$. This set is called a RNS basis and we have $a_i = a \bmod m_i$ which we will also denote $|a|_{m_i}$ for clarity. We generally assume that $0 \leq a < M = \prod_{i=1}^{n} m_i$. The main interest of such a system is that it independently distributes large integer operations on the small residue values. In particular, large integer operations become linear relatively to their sizes, and there is no carry propagation. These systems were introduced and developed in [43, 71, 72]. A good introduction can be found in [55].

However, they cannot be used directly in the cryptographic context because we need to reduce modulo a prime p that cannot be factorized in small moduli by definition. For constructing an arithmetic over $\mathbb{F}_p$, we assume that $p < M = \prod_{i=1}^{n} m_i$ and we use a variant of the Montgomery reduction algorithm presented in Subsection 2.2 to replace reductions modulo p by reductions modulo M instead of β^n [4, 5, 53, 64]. Montgomery reduction algorithm cannot be used without adaptation. Otherwise, we would have to divide by M on the RNS basis. This is not possible because $M = \prod_{i=1}^{n} m_i$ is not invertible modulo m_i. It is then necessary to introduce an auxiliary RNS basis to handle the inverse of M. The consequence is that arithmetic operations (e.g. the initial multiplication) must be performed on the two bases. This doubles the cost of arithmetic operations, but this is not an important issue since this cost is now linear, contrary to classical arithmetic. The biggest issue is that some changes of basis become necessary before and after dividing by M.

As a derivative of the Montgomery reduction algorithm, this algorithm has the same drawbacks that can be solved in the same way. The Montgomery rep-

Algorithm 5: RNS reduction based on Montgomery approach

Input: 2 coprime RNS basis $\mathcal{B}$ and $\mathcal{B}'$ such that $p < M < M'$
p given in basis $\mathcal{B}'$ and $-p^{-1}$ precomputed in basis $\mathcal{B}$
$a < Mp$ represented in both RNS basis
Result: $r = aM^{-1} \pmod p$ represented in both RNS basis, with $r < 2p$

1 $q \leftarrow a \times (-p^{-1})$ in $\mathcal{B}$
2 $[q \text{ in } \mathcal{B}] \longrightarrow [q \text{ in } \mathcal{B}']$ `// First basis extension`
3 $r \leftarrow (a + q \times p) \times M^{-1}$ in $\mathcal{B}'$
4 $[r \text{ in } \mathcal{B}\,] \longleftarrow [r \text{ in } \mathcal{B}']$ `// Second basis extension`

resentation of a in this case is $\overline{a} = aM \mod p$ and is stable for Montgomery addition and multiplication. And again, changing representation is rare and easy to perform with a or $a(M^2 \bmod p)$ as input of Algorithm 5 [9].

Instructions 1 and 3 of Algorithm 5 are component-by-component operations performed independently for each basis element, so they are very efficient (linear complexity). On the other hand, the basis extensions of instructions 2 and 4 have quadratic complexity. To convert a RNS representation to another RNS basis, we usually use a Lagrange interpolation: if $(a_1, a_2, \ldots, a_n)$ is the RNS representation of a in the basis $\mathcal{B} = \{m_1, \ldots, m_n\}$, then,

$$a = \sum_{i=1}^{n} \left| a_i M_i^{-1} \right|_{m_i} M_i - \alpha M \text{ where } M_i = \frac{M}{m_i}$$

Furthermore, the main challenge is to compute α efficiently. There are several approaches in the literature. In [53], it is shown that the first conversion can be only an approximation because q is then multiplied by p. However, the second conversion needs to be exact and can be done if $w < 2p < (1 - \frac{1}{\rho})M'$ for some $\rho \geq 2$ and $\frac{c_i}{2^w} < \frac{1}{n}(1 - \frac{1}{\rho})$ assuming $m_i = 2^w - c_i$. The same idea is used in [12], but the conditions are relaxed. In [73], a binary tree construction is used with a logarithmic depth with a modulo reduction at each level. Finally, it is shown in [7] that the overall complexity of Algorithm 5 can be optimized up to $\frac{7}{5}n^2 + \frac{8}{5}n$ multiplications at the moduli level.

For efficiency reasons, the size of the elements of the RNS basis is related to the word size (usually 1 or 2 words), and they must be chosen carefully so that reductions modulo the m_i are cheap. Moduli of special form as in Section 3 are usually chosen so that the reduction modulo m_i is, in this case, obtained with few shifts and additions. For example, Kawamura et al [53] are using moduli $m_i = 2^w - c_i$ with $c_i < 2^{w/2}$. In [12], a double Montgomery reduction is suggested to avoid internal modular reduction constraints and pseudo-Mersenne use. J. van der Hoven suggests in [73] to use *s-gentle moduli*, for example for $s = 2$, $m_i = 2^{2w} - \epsilon_i^2$ with $0 \leq \epsilon_i < 2^{(w-1)/2}$.

RNS arithmetic has many advantages compared to standard arithmetic. In particular, multiplication becomes a linear operation instead of a quadratic one. However, also, no more carriers are involved, and conjointly with the independence of the components, it makes RNS arithmetic very easy to implement and parallelize, especially in hardware, as shown in [45]. We can also introduce redundancy in the representation by adding new moduli. This allows for the introduction of randomization in the representation of elements that can be used as protection against differential side-channel attacks. Moreover, an RNS-based architecture is very flexible: with a given structure of n modular digit operators, it is possible to handle any value of $p < M$. Hence, the same architecture can be used for different cryptographic primitives, different levels of security and several base fields for each of these levels. The main drawback is the cost of the basis extensions, which makes the RNS reduction significantly more expensive than the classical Montgomery reduction. However, remember that RNS multiplication is much cheaper than classical multiprecision multiplication. So, there is a gap between the reduction and the multiplication cost, which does not occur in classical systems. We can take advantage of this gap by accumulating multiplications before reducing it. This method is called lazy reduction and is detailed in Section ??.

RNS systems have been successfully used in various cryptographic primitives, especially in hardware implementation (because it is necessary to have an efficient reduction modulo the m_i, which is challenging to achieve in software).
The first target was RSA (see [9] for example). The results are fascinating and competitive. However, the integers are very large in RSA, so a large basis are required for high-security levels, and it cannot always be realized with classical pseudo-Mersenne primes as in [53], especially if the word size is small. At this point, it would be very interesting to find other families of moduli allowing fast reduction, to develop variants of the RNS method, which is less demanding in terms of basis size (for example, by considering 2-words moduli). Some works already went in such direction recently [6, 12, 73] but for sure many improvements remain to be find and would be interesting not only for RSA.
RNS has also been successfully used in elliptic curve cryptography [7, 8, 45] and even allows to break of speed records in pairing-based cryptography [31]. Again, work remains to be done to adapt and optimize the elliptic curve and pairing formulas to exploit the complexity gap between the RNS multiplication and reduction steps more efficiently. RNS will also probably give interesting results in isogeny-based cryptography because, as in pairing-based cryptography, small degree extension fields are used in this context so that, as explained in Section 5, the lazy reduction can be systematically used, and this is advantaging RNS arithmetic.
More recently, an RNS-based implementation has been proposed for some protocols of Fully Homomorphic Encryption [49]. However, it generally has few interests in lattice-based cryptography because no large integers are used in this

case, so we do not get the gain of fast multiplication.

Finally, RNS arithmetic may be helpful for constrained devices and hardware and when we plan to use the base field level parallelization.

2.4 PMNS

Another way to avoid carriers is to work with polynomials. The so-called PMNS (Polynomial Modular Number System) represents integers modulo p by a polynomial that gives the integers once evaluated in a given value $\gamma < p$: a polynomial P represents $P(\gamma) \bmod p$ [10]. For efficiency reasons, P must satisfy some constraints:

- The degree of P is (strictly) bounded degree by a given integer n.
- The coefficients of P are bounded by a given parameter ρ:

$$||P||_\infty < \rho \simeq \sqrt[n]{p}.$$

  Note that ρ cannot be less than $(\sqrt[n]{p}+1)/2$ otherwise all the elements of $\mathbb{F}_p$ cannot be represented [63]

For example, if we choose $\gamma = 7$, then $-X - 1$ represents 11 modulo 19. Note that X^2 also represents 11 so this representation is a redundant system [39, 63].

Thanks to this representation, large integer arithmetic is replaced by polynomial arithmetic, which has many advantages (no carries, parallelization, independence of the components). The main problem is the growing of degrees and coefficients when operations are performed on the representative polynomials. Then degrees and coefficients may outreach the given constraints for a representative to be valid. So reduction techniques are necessary.

The easiest operation is to reduce the degree. This is called the external reduction, and this is done by reducing modulo a given polynomial E. This polynomial must be chosen as a monomial of degree n so that the degree of the reduced polynomial is less than $n - 1$. It must also cancel γ modulo p so that the reduced polynomial represents the same element in $\mathbb{F}_p$. For example, if this polynomial is well chosen and sparse, this reduction step is very efficient. It should also be chosen with small coefficients for efficiency reasons but, above all, to minimize the growth of the coefficients induced by this step [37].
This is, for example, the case if we consider a polynomial of the form $X^n - \lambda$, ideally with λ very small [10] or a power of 2. In this case, these systems are called AMNS (Adapted Modular Number System) [10]. In any case, the external reduction cost is usually only some extra additions [63].

Reducing the coefficients is more complicated and costly. This step is called internal reduction. The idea is the same. Namely, we use a polynomial M that cancels in γ. We then hope that dividing by M will give a polynomial with

smaller coefficients representing the same integer modulo p. Various methods have been proposed to do it efficiently [10, 11, 18, 60]. The most conclusive one is based on the Montgomery reduction [60]: Algorithm 6 requires an auxiliary integer ϕ, which is usually a power of 2 and uses Montgomery's usual trick to replace expensive divisions by M with cheap divisions by ϕ.:

Algorithm 6: Internal reduction using Montgomery reduction

Input: $R \in \mathbb{Z}_{n-1}[X]$ with $||R||_\infty \geq \rho$
the precomputed polynomial $M' = -M^{-1} \bmod (E, \phi)$
Result: $S \in \mathbb{Z}_{n-1}[X]$ s.t. $S(\gamma) = R(\gamma)\phi^{-1} \bmod p$
$Q \leftarrow R \times M' \bmod (E, \phi)$
$S \leftarrow (R + Q \times M)/\phi \bmod E$

Assuming that a Montgomery representation of elements is used ($\overline{R} = R\phi$), this algorithm outputs a polynomial S representing the same element as the input and whose coefficients are reduced if M and ϕ are satisfying the following conditions (for AMNS):

$$\rho > 2n|\lambda|\,||M||_\infty \text{ and } \phi > 2n|\lambda|\rho$$

Contrary to the external reduction (which has linear complexity), the internal reduction is quadratic, as an instance of the Montgomery technique. As for the RNS reduction, we will not get any complexity gain compared to the classical Montgomery reduction of Section 2.2, but we get simpler arithmetic that can easily be parallelized.
Moreover, this representation is naturally redundant so that we can protect the arithmetic against differential side-channel attacks. For example, we can add a random polynomial that cancels in γ modulo p [37]. It is also shown in [38] that this representation allows base field arithmetic without conditional branching, which can be very interesting in the grey box context.

Until recently, the main drawback of this method was the parameter generation. We apply LLL to the lattice of polynomials cancelled by γ to get M. But it is not so trivial to ensure that the polynomial M' exists as an integer polynomial and that the parameter ϕ is a power of 2 to minimize the costs. A generator has been very recently given in [38].
In terms of cryptographic applications, there are still few implementations in the literature because of this parameter generation issue. However, it is very promising for ECC, pairing or isogeny context [22], especially for constrained devices and when parallelization is considered at the base field level. It may also be a good alternative if side-channel resistance is needed at the base field level, thanks to the redundancy of the representation.

3 Modular reduction for special primes

The underlying base field can be chosen without restrictions in some cryptographic contexts. In this case, the module p can be taken in a specific form that allows much better complexity for reduction thanks to a dedicated algorithm. Usually, only several shifts and additions are needed for the modular reduction instead of $O(n^2)$ multiplications as in Section 2. The most known form is one of the generalized Mersenne primes we will describe now, but we will see that there are not the only ones. The main drawback of this strategy is that it requires an implementation/architecture dedicated to the specificity of p, which cannot be used for other base fields. Consequently, it is not always practical in either software or hardware implementation, and many users may prefer flexible products.

3.1 Mersenne and pseudo-Mersenne primes

The more efficient modules for reduction are the so-called Mersenne primes. They have the form $p = 2^e - 1$ and are historically used to break large prime numbers records. In this case, the reduction Algorithm 7 splits the input after e bits and uses that $2^e = 1 \bmod p$. Then adding the low and the high parts gives a reduced form of the input. It is not completely reduced since it is less than $2p$ instead of p, so an extra subtraction may be necessary to get the right result. Note that this subtraction is not directly done with p but with $2^e - 1$, which is mathematically equivalent but practically easier.

Algorithm 7: Reduction modulo a Mersenne prime $p = 2^e - 1$

Input: $0 \leq a < 2^e p$
Result: $r = a \bmod p$ and $0 \leq r < p$
Write $a = a_1 2^e + a_0$
$r \leftarrow a_0 + a_1$
$r' \leftarrow r + 1$ // $r' \leftarrow r - p + 2^e$
if $r' \geq 2^e$ **then** // `if` $r \geq p$
$r \leftarrow r' \bmod 2^e$ // $r \leftarrow r - p$
return r

The complexity of this algorithm is only one (large) addition. This is of course much cheaper than all other reduction algorithms, especially if e is a multiple of the word-size. However, Mersenne primes are very rare in the cryptographic range. Only 3 of them can be encounter in the cryptographic literature:

- $2^{521} - 1$ introduced in [66] and used as a standard for elliptic curve cryptography at the 256-bits security level.
- $2^{127} - 1$ used for hyperelliptic curves at the 128-bit security level [20, 65].

– $2^{31}-1$ used as part of a RNS basis on 32-bit architecture.

In order to get more candidates, Mersenne primes can be generalized to any prime having the form $2^e - c$ for small values of c. Replacing 2^e by c in the writing of a large integer a in base 2^e will indeed reduce the size of a.

$$a = a_0 + a_1 2^e \Rightarrow a = a_0 + a_1 c \bmod p$$

However, $a_1 c$ is not reduced enough, so this reduction step needs to be applied again to this term. It is easy to prove that if $|c|$ is less than $2^{\frac{e}{2}}$, then a third reduction step is not necessary, and we can then use Algorithm 8 involving only 2 reduction steps. In this case, the result is less than $3p$ (as the sum of 3 integers less than p), so 2 steps of subtraction of p may be necessary.

Algorithm 8: Reduction modulo $p = 2^e - c$, pseudo-Mersenne s.t. $|c| < 2^{\frac{e}{2}}$

Input: $0 \leq a < 2^e p$
Result: $r = a \bmod p$ and $0 \leq r < p$
  Write $a = a_0 + a_1 2^e$
  $b \leftarrow a_0 + a_1 c$
  Write $b = b_0 + b_1 2^e$
  $r = b_0 + b_1 c$
  $r' \leftarrow r + c$; // $r' \leftarrow r - p + 2^e$
  if $r' \geq 2^e$ **then** // if $r \geq p$
    $r \leftarrow r' - 2^e$ // $r \leftarrow r - p$
    $r' \leftarrow r + c$ // repeat the previous steps if r is still $\geq p$
    if $r' \geq 2^e$ **then**
      $r \leftarrow r' - 2^e$
    end if
return r

The final cost of Algorithm 8 is then 2 multiplications by c (and some additions). Consequently, c must be chosen carefully so that multiplications by c are fast to compute. Of course, it can be chosen very small, as suggested by Crandall [35]. This is, for example, the case of the prime $2^{255} - 19$ introduced by Bernstein to define the famous elliptic curve Curve25519 [19]. However, it can also be chosen very sparse in the base of the machine word. This is, for example, the case of Solinas primes [32, 70], which are built as evaluation in the power of 2 (ideally in β) of sparse polynomials with very small coefficients. The polynomial should be irreducible [59]. For example, if we choose the polynomial $X^3 - X - 1$ and apply it to 2^{64}, we get a pseudo-Mersenne prime with $c = 2^{64} + 1$ so that multiplying by c just adds up to 64-bit architecture. This prime and other Solinas ones are suggested by the NIST [62] and included in many standards for elliptic curve cryptography.

Pseudo-Mersenne class is also the main provider of moduli (non necessarily primes) for RNS basis because of their efficient reduction algorithm and the fact

that they can be freely chosen (as long as there are coprime). However, pseudo-Mersenne primes cannot be used in cryptographic contexts where modules have inherent constraints (e.g. RSA, pairings, Ring-lattices, isogenies).

3.2 Montgomery-friendly prime numbers

In this subsection, we are interested in the so-called Montgomery-friendly primes [6, 25, 26, 50]. They are build in the form $p = 2^{e_2}\alpha \pm 1$ with e_2 larger than the word size w so that the Montgomery reduction Algorithm 4 is particularly efficient. Indeed, this algorithm involves multiplication by the inverse of p modulo β. In the case of Montgomery-friendly primes, p is ± 1 modulo β so that this multiplication is free. Then, the only operation to perform in the `for` loop is the computation of $(r + qp)/\beta$ where $q = r \bmod \beta$. As $p = \beta 2^{e_2-w}\alpha - 1$ and assuming that $r = \sum r_i\beta^i$ at the beginning of each step of the `for` loop, this operation can be computed as:

$$(r + qp)/\beta = \left(\sum r_i\beta^i + r_0\beta 2^{e_2-w}\alpha - r_0\right)/\beta = \sum_{i\neq 0} r_i\beta^{i-1} + r_0 2^{e_2-w}\alpha$$

Algorithm 4 can then be rewritten as Algorithm 9 and the cost of each step of

Algorithm 9: Word version of the Montgomery reduction if $p = 2^{e_2}\alpha - 1$

Input: $0 \leq a < p\beta^n$, $\beta^{n-1} \leq p < \beta^n$ with $\beta = 2^w$ and $w \leq e_2$
Result: $r = a\beta^{-n} \bmod p$ and $0 \leq r < p$

```
r ← a
for i = 0 to n − 1 do
    r_0 ← r mod β
    r ← (r − r_0)/β + r_0 × α2^{e_2−w}
end for
r' ← r + (β^n − p) ;
if r' ≥ β^n then
    r ← r' − β^n
return r
```

the for loop is reduced to one multiplication of $\alpha 2^{e_2 \bmod w}$ by the single word value r_0. Assuming $\alpha 2^{e_2 \bmod w}$ fits on n_α words, Algorithm 9 then requires nn_α word multiplications. Of course the case $p = 2^{e_2}\alpha + 1$ is very similar.

As detailed in [6], the choice of α and e_2 is crucial to get an efficient reduction algorithm.

- If α is a power of 2, the multiplication by α is very efficient, but we finally get a Mersenne prime.
- If $\alpha = 2^{e_2'} - c$ with c small, we get an equivalent of pseudo-Mersenne primes.
- If $\alpha = 2^{e_2'} - c$ with c sparse involving only coefficients of the word size, we get an equivalent of the Solinas primes.

- If e_2 (and e'_2) is a multiple of the word size, this will reduce the word size n_α to the one of α.

If both α and e_2 are wisely chosen as above, the Montgomery reduction Algorithm 9 is very efficient by construction (each step of the for loop costs only one multiplication of a single word by c). Note that it will not give better efficiency than classical pseudo-Mersenne primes. It only provides more choices of good primes. This can be useful in many cryptographic contexts.

Such primes have been used in elliptic curve cryptography for years, such as

- $p_{192} = 2^{64}(2^{128} - 1) - 1$, which was already a Solinas prime used in most standards [62],
- $p_{448} = 2^{224}(2^{224} - 1) - 1$, used for the curve Ed448 [51],
- $p_{480} = 2^{448}(2^{32} - 1) - 1$ or $p_{512} = 2^{32}(2^{480} - 1) - 1$ for higher security levels.

Other Montgomery-friendly primes, which do not appear as pseudo-Mersenne primes, together with elliptic curves defined over the corresponding prime field, can be found in the recent cryptographic literature [26, 28, 50]. For example the primes $2^{240}(2^{16} - 88) - 1$ and $2^{240}(2^{14} - 127) - 1$ are recommended for the 128-bits security level. However, the parameters e_2 and e'_2 were not multiples of the word size in these proposals. The most accomplished example is given in [6]: $p_{256} = 2^{192}(2^{64} - 4) - 1$ can be seen as a good alternative to the Bernstein prime $2^{255} - 19$ because reduction modulo p_{256} requires only a few additions while the Bernstein prime requires multiplication by 19. Moreover, it is possible to find an elliptic curve satisfying the security requirements given in [21] with a smaller curve coefficient than Curve25519 [6].

The most natural application is isogeny-based cryptography because primes involved have the form $2^{e_2}3^{e_3} - 1$ and are naturally Montgomery-friendly. It was even the initial motivation to study these primes [6]. For example, for the famous SIKE prime $p_{503} = 2^{250}3^{159} - 1$ given in the SIKE proposals [2, 3] for the security category 2, we have $n = 8$ if $w = 64$ and $\alpha = 3^{159}2^{58}$ so that $n_\alpha = 5$. The overall cost of the reduction algorithm is then 40 word-multiplications (which fits with the reference implementation of SIKE). In order to follow the requirements on α and e_2 given above, it is not restrictive to consider primes p with a small cofactor f, as long as $p + 1$ is divisible by a sufficiently large power of 2 and 3 to ensure the existence of the isogenies [6]. This gives for example the prime $p_{512} = 31 \cdot 2^{256}\,3^{158} - 1$ which ensures the same security than p_{503} but with $n_\alpha = 4$ so that the overall cost of the reduction algorithm is only 32 word-multiplications instead of 40. More such primes for various security levels are given in [6].

The main interest of these new primes is that we get new prime numbers with an efficient reduction algorithm that can be used when pseudo-Mersenne or Solinas primes are too rare. The situation is, in fact, even better because,

contrary to pseudo-Mersenne primes, the value of c can be chosen even. Consequently, for the same size of constants c_i it is possible to find roughly twice many candidates. This can, for example, be very interesting for generating RNS bases handling larger values of p or involving smaller c_i values [6] (and then more efficient reduction algorithms). Consequently, RNS-based arithmetic can be considered for larger fields or rings.

Montgomery-friendly prime numbers form a large family and offer new (prime) numbers that can be used in many cryptographic contexts, such as elliptic curves, isogeny-based cryptography, or RNS arithmetic for large numbers.

Note that the prime used for the KYBER lattice-based primitive winner of the NIST post-quantum standardization process [1], $q = 13.2^8-1$, is Montgomery-friendly. However, the efficient reduction process could not be used at first sight (but on 8-bits devices which is quite rare today). For sure, some additional work has to be done to take advantage of the form of this prime in future implementations of KYBER. One could also use Montgomery-friendly prime numbers as RSA primes without loss of security while α is random and sufficiently large. Using these primes is recent in cryptography and requires further research.

4 Polynomial rings and extension fields

Polynomials are used in various contexts in public key cryptography, mainly to build extension fields and their interest. For extension fields or polynomial rings, a reduction polynomial is necessary. In any case, it can be chosen sparse so that reducing is not really an efficiency problem, just like Mersenne primes. So we will concentrate in this section on multiplication or squaring algorithms or sometimes on both multiplication and reduction as one common operation. Polynomials in cryptography can be categorized into two main families.

- The first one is for small degree extension fields. For example, isogeny-based post-quantum cryptography is usually defined on finite quadratic fields, so it is important to compute with degree 1 polynomials efficiently. Pairing-based cryptography also massively involves extension fields of degrees 6 to 48. It is not so small, but in practice, the degrees are smooth, and the extension fields are built as a succession of very small degrees extensions. So we only need to deal with small degrees, principally 2 and 3 [42]. However, higher extension degrees were recently used in pairing-based cryptography [34, 46, 47] to overcome the Kim-Barbulescu attack, which takes great advantage of smooth embedding degrees [16, 54].
- The second family is made of very large degrees of polynomials. There are, for example, used in post-quantum cryptography based on Ring-lattices where polynomial rings of large degree (up to several thousand) are common [1, 30]

As was the case for integers multiplication, the reference algorithm for polynomial multiplication is the schoolbook one, given by the same formula with n^2

complexity to multiply 2 polynomials of degree $n-1$

$$\sum_{i=0}^{n-1} a_i X^i . \sum_{i=0}^{n-1} b_i X^i = \sum_{i=0}^{2n-2} c_i X^i \text{ with } c_i = \sum_{k+l=i} a_k b_l$$

It is even simpler than the integer situation because no carry is involved in the polynomial case. As in the case of integers, we can also be smarter to perform such multiplications with various methods depending on the context.

4.1 Interpolation techniques for small degree extensions

All the algorithms for efficient polynomial multiplication are based on the same principle. The general idea is to evaluate the 2 input polynomials of degree $n-1$ in $2n-1$ values. We then get $2n-1$ evaluations of the product polynomial, which has a degree $2n-2$. Finally, the product polynomial can be recovered using an interpolation algorithm. As long as the evaluation values are cleverly chosen, in the sense that evaluation (and interpolation) complexity is low, the multiplication cost becomes linear because only $2n-1$ base field multiplications are required instead of n^2 for the schoolbook method. Depending on the choice of these evaluation points, we obtain different algorithms that may be used in different contexts.

The most know method is undoubtedly the Karastuba multiplication (already seen in Subsection 1.2 in the integer context). It evaluates degree 1 polynomials in $0, 1$ and ∞. If $A(X) = a_0 + a_1 X$ and $B(X) = b_0 + b_1 X$, we have

$$\begin{aligned} &A(0) = a_0, A(1) = a_0 + a_1, A(\infty) = a_1 \text{ and} \\ &B(0) = b_0, B(1) = b_0 + b_1, B(\infty) = b_1 \end{aligned}$$

Then AB evaluated in $0, 1$ and ∞ can be computed with only 3 multiplications.

$$AB(0) = a_0 b_0, AB(1) = (a_0 + a_1)(b_0 + b_1), AB(\infty) = a_1 b_1$$

So that AB can be easily and completely recovered with only few additions by classical interpolation.

$$AB(X) = a_0 b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1) X + a_1 b_1 X^2$$

This method is very suitable for degree 2 finite fields extensions but can be generalized to any higher degree thanks to a "divide and conquer" procedure. It leads to a complexity of $O(n^{\log_3 2}) = O(n^{1.58})$. However, some other choices of evaluation points may be more appropriate for specific situations.

The Toom-Cook method allows, for example, to multiply degree 2 polynomials with only 5 multiplications instead of 9 using the evaluation points $0, 1, -1, 2$ and ∞ [55]. It is, of course, particularly well adapted for degree 3 extensions but,

using the "divide and conquer" procedure, it asymptotically leads to complexity in $O(n^{\log_5 3}) = O(n^{1.46})$. However, it involves more additions than Karatsuba during the evaluation and interpolation steps and even some divisions by small numbers like 2 or 3 that must be treated very carefully and may invalidate the interest of the method depending on the context.

In the case of finite field extensions of small degrees (or towers of small degrees), the reducing polynomials have to be chosen carefully so that the reduction step is as cheap as possible. This step is usually included in the multiplication algorithm for efficiency reasons. The best choice is generally to choose an irreducible polynomial in the form $X^n - \mu$ such that multiplications by μ are easy, following the model of Mersenne primes. For example, the Karatsuba multiplication of degree 1 polynomials modulo $X^2 - \mu$ can be rewritten as

$$AB(X) = a_0b_0 + a_1b_1\mu + ((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1)\,X \tag{3}$$

4.2 Some special squarings: complex and Chung-Hasan

The same analysis should be done for the specific operation of squaring. Of course, the same algorithms as for multiplication apply with a better complexity because there is only one input. For example, squaring a degree 1 polynomial modulo $X^2 - \mu$ can be done in 2 squarings and one multiplication, thanks to the schoolbook method.

$$(a_0 + a_1X)^2 = a_0^2 + \mu a_1^2 + 2a_0a_1X$$

while the Karatsuba method replaces $2a_0a_1$ by $(a_0 + a_1)^2 - a_0^2 - a_1^2$ and then one multiplication by one squaring in the base field (which yields to a global complexity of 3 squarings). But, we can alternatively use the so-called complex method that squares with only 2 multiplications in the base field

$$(a_0 + a_1X)^2 = (a_0 + \mu a_1)(a_0 + a_1) - (\mu + 1)a_0a_1 + 2a_0a_1X$$

This method is of course particularly efficient if $\mu = -1$, which explains its name.

In the case of degree 2 polynomials (and then degree 3 extension fields), the Schoolbook squaring modulo $X^3 - \mu$ requires 3 squarings and 3 multiplictions in the base field

$$(a_0 + a_1X + a_2X^2)^2 = a_0^2 + \mathbf{2a_1a_2}\mu + \left[\mathbf{2a_0a_1} + \mu a_2^2\right] X + \left[a_1^2 + \mathbf{2a_0a_2}\right] X^2 \tag{4}$$

As usual, the Karatsuba method computes the bolded terms of (4) with only one squaring instead of one multiplication in the base field (and so a global complexity of 6 squarings). In this situation, we can also reduce the complexity to 5 operations in the base field using the Chung-Hasan method [33]. There are several variants but for example, the term in X^2 in (4) can be computed as

$$a_1^2 + 2a_0a_2 = (a_0 + a_1 + a_2)^2 - (2a_0a_1 + 2a_1a_2 + a_0^2 + a_2^2).$$

4.3 Application to pairing based cryptography

Small degree extensions are used for isogenies and pairings. Determining which method is the best in a small degree extension is not immediate and not universal because it depends on the relative cost of multiplications, squarings and additions in the base field as well as multiplications by μ (if the reduction polynomial has the form $X^n - \mu$). But the base field may vary a lot depending on the targeted cryptosystem and even for a given use case. The situation is quite simple in isogeny-based cryptography because it is only using quadratic extensions.

Nevertheless, it is less trivial in the pairing context. Indeed extension degrees can be up to 48 and generally have the form 2^i3^j. This means that the extensions are built as a succession of degree 2 and degree 3 intermediate extensions. The first step is to choose this sequence and the reduction polynomials so that the global complexity is the best. It is, for example, shown in [36] choosing that even for a degree 12 extension in the pairing context, all the choices for building $\mathbb{F}_{p^{12}}$ are not equivalent and may result in significant differences in the final pairing cost (because the extension field arithmetic is predominant mainly in any pairing computation). Then, for each level in the extension tower, we have to choose the best algorithm for multiplying and squaring. As already mentioned, this choice depends on the relative cost of basic operations at the previous level. However, these relative costs are not always the same. They depend on the reduction polynomial used but also on the level itself: it is not the same at the ground level (for building $\mathbb{F}_{p^2}$ over $\mathbb{F}_p$, for example) and at the top level (for building $\mathbb{F}_{p^{12}}$ over $\mathbb{F}_{p^6}$ for example). Moreover, all these choices depend on the base field because an efficient reduction polynomial ($\mu = -1$, for example) will not always be irreducible. So there is no uniform answer to get the best possible arithmetic for pairings. New choices must be made for each case, and the final impact on the extension field arithmetic is important. It is also shown that it may influence the choice of the curve parameters themselves [41].

Of course, other operations on polynomials as additions, inversions or images of the Frobenius map may be considered for a complete study of the algorithmic complexity of the arithmetic of extension fields. This is, for example, done in [41] in the pairing context, but we did not go at this level of details here because multiplications and squarings have by far the largest impact.

4.4 Interpolation techniques for large degrees

The situation is quite different for large degree polynomials/extension degrees. The first main reason is that the base field is finite, so the number of evaluation points is naturally limited. We then may not be able to have sufficiently many points (or efficient points) to set up an evaluation/interpolation technique. One elegant solution is to use Chudnovsky-type algorithms. These algorithms use

points on an algebraic curve defined on the base field as evaluation values instead of base field elements [13, 14, 23]. It is interesting because, as the genus grows, we can have more points on the curve than elements in the base field. Their practical use in cryptography is still limited, so we do not give details here, but some recent publications on the subject make a good survey [15].

The second main reason is that large degrees make the famous Fast Fourier Transform (FFT) method competitive, and its complexity becomes theoretically quasi-linear in the degree. This method uses the $2n$-th roots of unity as evaluation points (of course, assuming they are lying in the base field). The problem, in this case, is, of course, to be able to evaluate the polynomials in these points efficiently. This can be done thanks to a famous divide-and-conquer algorithm from Cooley and Turkey [24]. This algorithm becomes computationally attractive only if n is large so that the evaluation complexity in the polynomial multiplication process is balanced by the gain on the number of base field multiplications.

This method was neglected until recently in cryptography because it becomes interesting only for very large integers or polynomials which were not used in this domain. The situation is different now because very large polynomial rings are used in lattice-based cryptography, especially in KYBER [1], which recently won the NIST post-quantum standardization process [61]. The FFT computes the Discrete Fourier Transform (DFT, when the base field is $\mathbb{C}$) or the Number Theoretic Transform (NTT, when the base field is finite, so DFT and NTT are just different names for the same thing depending on the context). Its general principle can be found in many algorithmic books because it is very old, famous, and widely used in many engineering, signal processing, mathematical and physical domains. We give it here in the general case to fix notations and explain how it applies to large polynomial multiplications and cryptographic applications. It is as follows: assume a primitive n-th roots of unity ω lies in the base field (for example, $\omega = e^{\frac{2i\pi}{n}}$ in $\mathbb{C}$) and let $A = \sum_{i=0}^{n-1} a_i X^i$ be a polynomial represented by the sequence of its coefficients $[a_0, a_1, \cdots, a_{n-1}]$. The Discrete Fourier Transform of A is defined as the evaluation of A in all the n-th root of unity:

$$\hat{A} = DFT(A) = [\hat{a}_0, \hat{a}_1, \cdots, \hat{a}_{n-1}] \text{ with } \hat{a}_k = \sum_{i=0}^{n-1} a_i \omega^{ik} = A(\omega^k)$$

We can also define the inverse transform which is nothing but the classical interpolation to recover A:

$$iDFT(\hat{A}) = [a'_0, a'_1, \cdots, a'_{n-1}] \text{ with } a'_j = \frac{1}{n} \sum_{k=0}^{n-1} \hat{a}_k \omega^{-kj} \tag{5}$$

And it is easy to prove that $a'_j = a_j$ so that the names are well chosen and

$$A = iDFT(DFT(A))$$

The application to degree $n-1$ polynomials multiplication immediately uses $2n$-th roots of unity (to get $2n-1$ evaluation points). The first step is to compute the DFT of inputs which can be asymptotically done in $O(n \log n)$ thanks to the Cooley-Turkey algorithm [24]. We then say that we are in the Fourier domain where multiplications can be done component by component (and then in $O(n)$). We finally return to the initial domain computing the inverse DFT (again in $O(n \log n)$) of the result. It is important to note that we can stay in the Fourier domain for consecutive operations, which may save many operations in practice. The inverse transformation needs to be applied only to the final result.

In the particular case of a polynomial ring defined by the specific reduction polynomial X^n+1, it is possible to get an improved version of the multiplication algorithm based on the NTT that mimics a transform of size n instead of $2n$. Let ψ be a primitive $2n$-th root of unity in the base field.

- $\psi^2 = \omega$ is a n-th root of unity
- $\psi^n = -1$

There is an obvious correspondence between this second relation and the combination of the 2 operations we have to perform: reduce mod X^n+1 and evaluate in ψ. This correspondence leads to the following writting for the product of 2 degree $n-1$ polynomials A and B modulo X^n+1

$$C = AB = \sum_{j=0}^{2n-2} c_j X^j \Rightarrow C = \sum_{j=0}^{n-1} (c_j - c_{n+j}) X^j \bmod X^n + 1$$

and in the NTT context, we have

$$\begin{aligned} c_j - c_{n+j} &= \frac{1}{2n} \sum_{k=0}^{2n-1} \hat{c}_k \left(\psi^{-kj} - \psi^{-k(n+j)} \right) \\ &= \frac{1}{2n} \sum_{k=0}^{2n-1} \hat{c}_k \psi^{-kj} \left(1 - (-1)^k \right) \text{ because } \psi^n = -1 \\ &= \frac{1}{n} \sum_{k=0}^{n-1} \hat{c}_{2k+1} \psi^{-j} \omega^{-kj} \end{aligned}$$

This expression is very similar to the inverse DFT of size n given by formula (5). There is only an extra factor ψ^{-j}. As a consequence, we can define a variant of the DFT/NTT of A by

$$\hat{a}_k = \sum_{i=0}^{n-1} a_i \psi^i \omega^{ik}$$

as well as its inverse. This transform is nothing but $A(\psi\omega^k)$. It has size n instead of $2n$ and allows polynomial multiplication in quasi-linear complexity with the modular reduction mod X^n+1 included. That is the reason why this polynomial ring is widely used in Ring/Module lattices cryptography [1, 30].

5 Lazy reduction

Lazy reduction is a technique commonly used in finite fields implementations [57, 68, 74] but rarely presented in the litterature [7, 40] Its principle is to delay the reduction step when we have to compute several products that will be summed. For example, if a, b, c and d lie in $\mathbb{F}_p$, computing and reducing a pattern of the form $ab + cd$ is done with 2 multiplications (ab and cd), one (large) addition and one final modular reduction instead of 2 multiplications, 2 reductions and 1 (normal) addition. Of course, this implies that the reduction algorithm can take larger integers as input (less than $2p^2$ instead of less than p^2 in the above example), but it is not cumbersome in practice. There are 2 main contexts where using lazy reduction is particularly interesting.

The first is when the reduction step is costly compared to the multiplication step so factoring a reduction step for several other operations is particularly worthwhile. This is for example the case if RNS (see Section 2.3) or PMNS (see Section 2.4) arithmetic is used. It has been shown that lazy reduction was very profitable in this case in the elliptic curve context [7, 8] as well as in the pairing context [31, 40]. This was for RNS arithmetic because PMNS was not yet developed, but it is no doubt that the high cost of an internal reduction in PMNS would greatly advantage the lazy reduction technique for any cryptographic primitive. This may be an interesting trail to follow for people interested in PMNS implementations.

The second favourable context for lazy reduction is when patterns of the form $ab + cd$ naturally occur in a cryptographic protocol or can be revealed. This is, for example the case in elliptic curve cryptography where addition and doubling formulas can be adapted to involve more such patterns [7, 8].
This is also the case in pairing-based cryptography and, more generally, in extension fields. Indeed, when a multiplication in an extension field $\mathbb{F}_{p^k}$ is performed, it is necessary to make only one reduction modulo p for each component of the result instead of one for each $\mathbb{F}_p$ multiplication involved [40]. This means that only k reduction steps are required instead of k^2 for a classical multiplication in $\mathbb{F}_{p^k}$. For example, if $\mathbb{F}_{p^2}$ is defined by $\mathbb{F}_p[X]/(X^2 + 1)$ and $A = a_0 + a_1X$ and $B = b_0 + b_1X \in \mathbb{F}_{p^2}$, we have $AB = a_0b_0 - a_1b_1 + (a_0b_1 + a_1b_0)X$.
We can see that $ab + cd$ patterns naturally occur which makes lazy reduction particularly relevant. Note that, as elements in $\mathbb{F}_{p^2}$ have 2 independent components, it is not possible to have less than 2 reductions in $\mathbb{F}_p$ in this case. The case of 6 and 12 extension degrees is done in detail in [40], showing a significant implementation improvement. These degrees were chosen because of their particular interest in the pairing context at that time, and especially because of the dominance of BN curves [17]. However, other degrees may now be considered (in the pairing context or in other contexts), and the detailed impact of a lazy reduction in these cases should be studied in the future.
Such patterns also naturally appear in lattice-based cryptography because most of the computations in this domain are inner products that are, by definition, well

adapted to the lazy reduction trick. By the way, the reference implementations of Kyber [1] use lazy reduction. This situation is, in fact, even more, favourable in lattice-based cryptography because the reduction module involved is usually much less than a machine word (for example, Kyber prime is only 12-bits long). Consequently, many multiplications and additions may be performed and accumulated before reducing without overflowing a single machine-word [48]. One could even go further in this case by storing 2 integers on the same machine word. It would for sure be interesting to study in detail how this could be done in practice and what would be the expected gain.

6 Conclusion

Finite field arithmetic significantly impacts cryptographic performances but is very diverse and dependent on the cryptosystem. The efficient algorithms are not always the same and must be adapted to the context. They indeed depend on the size of the base field or ring, on the targeted device and its properties (possibilities of parallelism, size of machine words, software/hardware) but also on what is expected for the implementation in terms of performance and security (side-channel resistance ability for example).

Of course, this paper is far from exhaustive about algorithms for finite fields or ring arithmetic, and there are many missing improvements and variants or hidden things in what is presented. However, it gives a good overview of what can be done for efficient finite field arithmetic for cryptographic applications, and it gives some ideas to improve existing methods, extend their range of interest or even discover new ones.

References

1. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber (version 3.02) – Submission to round 3 of the NIST post-quantum project.https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf
2. Azarderakhsh, R., Campagna M., Costello, C., De Feo L., Hess B., Jalali A., Jao D., Koziel B., LaMacchia B., Longa P., Naehrig M., Renes J., Soukharev V., Urbanik D.: Supersingular Isogeny Key Encapsulation, Submission to the NIST's post-quantum cryptography standardization process, https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/submissions/SIKE.zip (2017)
3. Azarderakhsh, R., Campagna M., Costello, C., De Feo L., Hess B., Jalali A., Jao D., Koziel B., LaMacchia B., Longa P., Naehrig M., Renes J., Soukharev V., Urbanik D.: Supersingular Isogeny Key Encapsulation, Submission to the NIST's post-quantum cryptography standardization process, https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/SIKE.zip (2019)

4. Bajard, J.C., Didier, L.S., Kornerup, P.: A RNS Montgomery's Modular Multiplication. IEEE Transactions on Computers, **47:7** (1998)
5. Bajard, J.C., Didier, L.S., Kornerup, P.: Modular multiplication and base extension in residue number systems. 15th IEEE Symposium on Computer Arithmetic, IEEE Computer Society Press pp. 59–65 (2001)
6. Bajard, J. C., Duquesne, S.: Montgomery-friendly primes and applications to cryptography. Journal of Cryptographic Engineering **11** (2021). https://doi.org/10.1007/s13389-021-00260-z
7. Bajard, J. C., Duquesne, S., Ercegovac, M.: Combining leak-resistant arithmetic for elliptic curves de
ned over Fp and RNS representation. Publications Mathématiques de Besancon **1** pp. 67–87 (2013)
8. Bajard, J.C., Duquesne, S., Ercegovac, M., Meloni, N.: Residue systems efficiency for modular products summation: Application to Elliptic Curves Cryptography. SPIE **6313**, 631304 (2006)
9. Bajard, J.C., Imbert, L.: A full RNS implementation of RSA. IEEE Transactions on Computers **53:6** pp. 769–774 (2004)
10. Bajard, J.C., Imbert, L., Plantard, T.: Modular number systems : Beyond the mersenne family. In Selected Areas in Cryptography. SAC 2004. Lecture Notes in Computer Science book series **3357**, pp 159–169. Springer, Heidelberg (2004)
11. Bajard, J.C., Imbert, L., Plantard, T.: Arithmetic operations in the polynomial modular number system. In 17th IEEE Symposium on Computer Arithmetic (ARITH-17), pp. 206–213 (2005)
12. Bajard, J. C., Merkiche, N.: Double Level Montgomery Cox-Rower Architecture, New Bounds. In: Smart Card Research and Advanced Applications. CARDIS 2014. Lecture Notes in Computer Science **8968**. Springer, Heidelberg (2014)
13. Ballet, S.: Curves with Many Points and Multiplication Complexity in Any Extension of Fq. Finite Fields and Their Applications **5**, pp.364–377 (1999)
14. Ballet, S., Rolland, R.: Multiplication algorithm in a finite field and tensor rank of the multiplication. Journal of Algebra **272**(1), pp.173–185 (2004)
15. Ballet, S., Chaumine, J., Pieltant, J., Rambaud, M., Randriambololona, H., Rolland, R.: On the tensor rank of multiplication in finite extensions of finite fields and related issues in algebraic geometry. Uspekhi Mathematichskikh Nauk **76:1**(457), pp. 31–94 (2021)
16. Barbulescu, R., Duquesne, S.: Updating Key Size Estimations for Pairings. Journal of Cryptology **32**(4), pp. 1298–1336 (2019). https://doi.org/10.1007/s00145-018-9280-5
17. Barreto, P., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Selected areas in cryptography–SAC 2005. Lecture Notes in Computer Science **3897**, pp.319–331. Springer, Heidelberg (2006)
18. Barrett, P.: Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In: Odlyzko, A.M. (eds) Advances in Cryptology - CRYPTO' 86. Lecture Notes in Computer Science **263**, pp. 311–323. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_24
19. Bernstein, D. J.: Curve25519: New Diffie-Hellman Speed Records. In: Public Key Cryptography - PKC 2006, Lecture notes in computer science, pp. 207–228 (2006)
20. Bernstein, D. J., Chuengsatiansup, C., Lange, T., Schwabe, P.: Kummer Strikes Back: New DH Speed Records. In: Advances in Cryptology – ASIACRYPT 2014, Lecture notes in computer science **8873**, pp. 317–337 (2014)
21. Bernstein, D.J., Lange, T.: SafeCurves: choosing safe curves for elliptic-curve cryptography. http://safecurves.cr.yp.to

22. Bouvier, C., Imbert, L.: An Alternative Approach for SIDH Arithmetic. In: Garay, J.A. (eds) Public-Key Cryptography – PKC 2021. Lecture Notes in Computer Science **12710**. Springer, Heidelberg (2021).https://doi.org/10.1007/978-3-030-75245-3_2
23. Chudnovsky, D., Chudnovsky, G.: Algebraic complexities and algebraic curves over finite fields. Journal of Complexity **4**, pp. 285–316 (1988)
24. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**(90), pp. 290–301 (1965). https://doi.org/10.2307/2003354
25. Bos, J.W., Costello, C., Hisil, H., Lauter, K: Fas:t Cryptography in Genus 2. In: Johansson, T. and Nguyen, P. Q. (eds) Advances in Cryptology – EUROCRYPT 2013, Lecture notes in computer science , pp. 194–210. Springer Heidelberg (2013)
26. Bos, J., Costello, C., Longa, P., Naehrig, M.: Selecting elliptic curves for cryptography: an efficiency and security analysis. Journal of Cryptographic Engineering **6**(4), pp. 259–286 (2016)
27. Bos, J., Lenstra, A.: Topics in Computational Number Theory Inspired by Peter L. Montgomery, Cambridge University Press (2017). https://doi.org/10.1017/9781316271575
28. Bos, J. Montgomery, P.L.W.: Topics in Computational Number Theory Inspired by Peter L. Montgomery. In: Bos, J. and Lenstra, A. (eds), Cambridge University Press (2017)
29. Bosselaers, A., Govaerts, R., Vandewalle. J.: Comparison of the three modular reduction functions. Lecture Notes in Computer Science **773** pp. 175–186. Springer, Heidelberg (1994)
30. Chen, C., Danba, O., Hoffstein, J., Hulsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z., Saito, T., Yamakawa, T., Xagawa, K.: NTRU – Submission to round 3 of the NIST post-quantum project.https://ntru.org/*
31. Cheung, R., Duquesne, S., Fan, J. , Guillermin, N., Verbauwhede, I., Yao, G.: FPGA Implementation of Pairings Using Residue Number System and Lazy Reduction. In: CHES-2011, Lecture Notes in Computer Science **6917** pp. 421–441. Springer, Heidelberg (2011)
32. Chung, J., Hasan, A.: More generalized Mersenne numbers. In: SAC 2003, Lecture Notes in Computer Science **3006** pp. 335–347. Springer, Heidelberg (2003)
33. Chung, J., Hasan, A.: Asymmetric squaring formulae. In: 18^{th} symposium on Computer Arithmetic. IEEE conference publications pp.113-122 (2017)
34. Clarisse, R. , Duquesne, S., Sanders, O.: Curves with Fast Computations in the First Pairing Group. In: Krenn S., Shulman H. and Vaudenay, S. (eds) CANS 20 : 19th International Conference on Cryptology and Network Security. Lecture Notes in Computer Science **12579** pp. 280–298. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-65411-5_14
35. Crandall, R.: Method and apparatus for public key exchange in a cryptographic system, U.S. Patent #5159632 (1992)
36. Devegili, A.J., O'Eigeartaigh, C., Scott, M., Dahab, R.: Multiplication and Squaring on Pairing-Friendly Fields, IACR Cryptology ePrint Archive **471** (2006), http://eprint.iacr.org/2006/471
37. Didier, L-S., Dosso, F-Y., El Mrabet, N., Marrez, J., Véron, P.: Randomization of arithmetic over polynomial modular number system. In 2019 IEEE 26th Symposium on Computer Arithmetic (ARITH), pp.199–206 (2019)
38. Didier, L-S., Dosso, F-Y., Véron, P.: Efficient modular operations using the Adapted Modular Number System. Journal of Cryptographic Engineering **10**(2), pp. 111–133 (2020)

39. Dosso, F-Y.: Computer arithmetic contribution to side channel attacks resistant implementations. PhD, University of the South, Toulon-Var, France (2020)
40. Duquesne, S.: RNS arithmetic in $\mathbb{F}_{p^k}$ and application to fast pairing computation. Journal of Mathematical Cryptology **5**(1), pp. 51-88 (2011).https://doi.org/10.1515/jmc.2011.006
41. Duquesne, S., El Mrabet, N., Haloui, S., Rondepierre, F.: Choosing and generating parameters for pairing implementation on BN curves. Applicable Algebra in Engineering, Communication and Computing **29**, pp. 113–147 (2018). https://doi.org/10.1007/s00200-017-0334-y
42. El Mrabet, N., Joye, M.: Guide to pairing based cryptography, Chapman & Hall/CRC Cryptography and Network Security (2016). https://doi.org/10.1201/9781315370170
43. Garner, H.L.: The residue number system. IRE Transactions on Electronic Computers, EL **8:6** pp. 140–147(1959)
44. GNU MP, http://gmplib.org
45. Guillermin, N.: A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$.In: CHES 2010, Lecture Notes in Computer Science **6225**, pp. 48–64 Springer, Heidelberg (2010)
46. Guillevic, A.: Comparing the Pairing Efficiency over Composite-Order and Prime-Order Elliptic Curves. In: Jacobson M.J., Locasto, M.E., Mohassel, P., Safavi-Naini R. (eds): ACNS 13 – 11th International Conference on Applied Cryptography and Network Security. Lecture Notes in Computer Science **7954**, pp. 357-372. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38980-1_22
47. Guillevic, A.: A Short-List of Pairing-Friendly Curves Resistant to Special TNFS at the 128-Bit Security Level. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds) PKC 2020 : 23rd International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science **12111**, pp. 535–564. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-45388-6_19
48. Güneysu, T., Oder, T., Pöppelmann, T., Schwabe, P.: Software Speed Records for Lattice-Based Signatures. In: Gaborit, P. (eds) Post-Quantum Cryptography–PQCrypto 2013. Lecture Notes in Computer Science **7932**. Springer, Heidelberg (2013).https://doi.org/10.1007/978-3-642-38616-9_5
49. Halevi, S., Polyakov, Y., Shoup, V.: An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. In: Matsui, M. (eds) Topics in Cryptology - CT-RSA 2019. Lecture Notes in Computer Science **11405**. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-12612-4_5
50. Hamburg, M.: Fast and compact elliptic-curve cryptography. IACR Cryptology ePrint Archive **309**, http://eprint.iacr.org/2012/309 (2012)
51. Hamburg, M.: Ed448-Goldilocks, a new elliptic curve. IACR Cryptology ePrint Archive **625**, https://eprint.iacr.org/2015/625 (2015)
52. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata, Sov Phys Dokl **7** (1963)
53. Kawamura, S., Koike, M., Sano, F., Shimbo, A.: Cox-Rower Architecture for Fast Parallel Montgomery Multiplication. In: EUROCRYPT 2000. Lecture Notes in Computer Science **1807**, pp. 523–538. Springer, Heidelberg (2000)
54. Kim, T., Barbulescu, R.: Extended Tower Number Field Sieve : A New Complexity for the Medium Prime Case. In: Robshaw, M. Katz, J. (eds) Advances in Cryptology - CRYPTO 2016. Lecture Notes in Computer Science **9814**, pp. 543–571. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_20
55. Knuth, D.: Seminumerical Algorithms. The Art of Computer Programming **2**. Addison-Wesley (1981).

56. Koç, Ç. K., Tolga, A., Burton, S.: analysing and comparing Montgomery multiplication algorithms. IEEE Micro **16**(3), pp. 26–33 (1996)
57. Lim, C.H., Hwang, H.S.: Fast Implementation of Elliptic Curve Arithmetic in GF(p^n). In: Public Key Cryptography–PKC 2000. Lecture Notes in Computer Science **1751**, pp.405–421. Springer, Heidelberg (2000)
58. Montgomery, P.: Modular Multiplication without Trial Division. Mathematics of Computation **44**(170), pp. 519–521 (1985)
59. Murty, R.: Prime Numbers and Irreducible Polynomials. The American Mathematical Monthly **109**(5) (2002)
60. Negre, C., Plantard, T.: Efficient modular arithmetic in adapted modular number system using Lagrange representation. In Information Security and Privacy, ACISP 2008, pp. 463–477 (2008)
61. NIST Post-Quantum Cryptography Standardization Process https://csrc.nist.gov/Projects/post-quantum-cryptography
62. NIST ECC standards, https://csrc.nist.gov/publications/detail/fips/186/4/final
63. Plantard, T.: Arithmétique modulaire pour la cryptographie. PhD thesis, Montpellier 2 University, France (2005)
64. Posch, K.C. , Posch, R.: Modulo Reduction in Residue Number Systems. IEEE Trans. Parallel Distrib. Syst. **6**(5), pp. 449–454 (1995)
65. Renes, J., Schwabe, P., Smith, B., Batina, L.: μKummer: efficient hyperelliptic signatures and key exchange on microcontrollers. In: Cryptographic Hardware and Embedded Systems – CHES 2016, Lecture notes in computer science **9813** (2016)
66. Robinson, R. M.: Mersenne and Fermat Numbers. Proc. Amer. Math. Soc. **5** pp. 842–846 (1954)
67. Savaş, E., Koç, Ç. K.: Finite field arithmetic for cryptography, IEEE Circuits and Systems Magazine **10**(2), pp. 40–56 (2010). https://doi.org/10.1109/MCAS.2010.936785
68. Scott, M.: Implementing cryptographic pairings. In: Pairing 2007. Lecture Notes in Computer Science **4575**, pp. 177–196. Springer, Heidelberg (2007)
69. Scott, M.: Missing a trick: Karatsuba variations, Cryptography and Communications **10**, pp. 5–15 (2018). https://doi.org/10.1007/s12095-017-0217-x
70. Solinas, J. A.: Generalized Mersenne Numbers. Technical report Center for Applied Cryptographic Research, University of Waterloo (1999)
71. Svoboda, A., Valach, M.: Operational Circuits. Stroje na Zpracovani Informaci, Sbornik III, Nakl. CSAV, Prague, pp. 247–295 (1955)
72. Szabo, N.S., Tanaka, R.I.: Residue Arithmetic and its Applications to Computer Technology. McGraw-Hill (1967)
73. Van der Hoven, J.: Fast Chinese Remaindering in Practice. In: Mathematical Aspects of Computer and Information Sciences. MACIS 2017. Lecture Notes in Computer Science **10693**, Springer Heidelberg (2017)
74. Weber, D., Denny, T.: The solution of McCurley's discrete log challenge. In: Advances in Cryptology–CRYPTO 98. Lecture Notes in Computer Science **1462**, pp. 458–471. Springer, Heidelberg (1998)
75. Weimerskirch, A., Paar, C.: Generalizations of the Karatsuba Algorithm for Efficient Implementations,IACR Cryptol. ePrint Arch. **224** (2006), http://eprint.iacr.org/2006/224