

Fast Polynomial Arithmetic in the Exponent without Roots of Unity

Diane Ducrocq¹ and Pierrick Gaudry²

¹ ENS Paris-Saclay

² Université de Lorraine, CNRS, Inria

Abstract. In cryptographic algorithms based on groups in which the discrete logarithm is supposed to be hard, it is frequent to perform polynomial arithmetic in the exponent of group elements. This old technique has become a critical operation in many recent advanced protocols, mostly related to commitments and zero-knowledge proofs, where polynomials of high degrees are involved. Number Theoretic Transform (NTT) techniques in the exponent are often implemented to gain speed. However, they require roots of unity to exist in the field $\mathbb{F}_q$ where q is the order of the group.

In this work, we propose algorithms that preserve the quasi-linear complexity of the Fourier Transforms, even when no root of unity are available. For this, we develop a formalism for the algebraic structure of polynomials with coefficients in a group, that allows us to adapt asymptotically fast polynomial multiplication algorithms to the case of arithmetic in the exponent.

In particular, we design an algorithm inspired by the Schönhage-Strassen algorithm, which runs about twice slower than NTT but does not require any arithmetic property on the order of the group. We explain how this can be adapted to batched KZG commitment proofs and the Bayer-Groth verifiable mixnet used in e-voting.

1 Introduction

In cryptography, polynomial arithmetic in the exponent has a long history. It dates back to (at least) Pedersen’s distributed key generation [17], where the Shamir’s secret-sharing scheme is adapted: The scalar values a_i are hidden by using a group G with generator g where the discrete logarithm is difficult, and Pedersen’s protocol manipulates them as g^{a_i} instead of directly.

It has then become a classical tool in advanced cryptographic schemes. In the past decade, it is ubiquitous in zero-knowledge and commitment schemes, where practical applications lead to polynomials in the exponent of very high degrees. See for instance [7, 20, 22]. When the group G has order q , with q a prime such that $q - 1$ is divisible by a large power of 2, then the exponents live in the field $\mathbb{F}_q$ which has many roots of unity, allowing techniques based on the Number Theoretic Transform (NTT). This yields a complexity that is quasi-linear in the degree of the polynomials, while a naive approach would be quadratic.

There are cases where the group is constructed with this NTT constraint in mind. A famous example is the BLS12-381 pairing-friendly curve, whose field of exponents supports NTTs of length up to 2^{32} . On the other hand, the standard, legacy curves that one might wish to use for building a protocol, were not designed with this application in mind and therefore do not support NTT. Other examples are elliptic curves whose group order is equal to the base field of standard curves (typically NIST-P256 and Ed25519), as constructed in [10] and [24]. The goal of these curves which are “companions” to standardized curves is to provide proof or commitment systems that allow a simple arithmetization of the standardized curve arithmetic. These latter examples do not support NTT either.

We therefore emphasize two motivations for raising the arithmetic constraint on the field of exponents and provide fast polynomial arithmetic in the exponent beyond NTT.

First, protocol designers do not always have the freedom to choose the underlying group. It could be for legal / certification reasons, where to be conforming to a regulation, one has only a few standard curves to choose from. This can also be due to technical reasons, with existing equipment or software that can not be easily upgraded and with which the system must be compatible. We have two important examples in mind. The KZG polynomial commitment scheme has a batched version [11] that relies on fast polynomial arithmetic in the exponent, that is currently not possible for legacy elliptic curves (unless one allows a quadratic complexity). Another example comes from electronic voting. The Bayer-Groth verifiable mixnet protocol [3] contains a zero-knowledge sub-protocol that requires the prover to perform a computation that boils down to polynomial multiplication in the exponent. To obtain a quasi-linear complexity in the number of ballots to mix, the authors demand that the group order is NTT-compatible. As a consequence, achieving the advertised complexity requires abandoning standard elliptic curves. One can note that the Bayer-Groth algorithm is currently used in practice for e-voting in Switzerland [21] with an implementation with a naive algorithm for the arithmetic in the exponent that is a bottleneck for the prover.

A second motivation for developing fast polynomial arithmetic in the exponent beyond NTT comes from the parameter generation, namely the construction of elliptic curves with good properties, especially regarding pairings. They are used in several proof systems, and recursive constructions require so-called cycles of elliptic curves (the HyperNova [16] system is an example). It is a challenge to construct such cycles with as many good properties as possible (see [1] for a survey on these curve constructions). Having many roots of unity in the field of exponents is one of the properties that is not always easy to satisfy. For instance, in the Icicle [13] library, the API provides NTT in the exponent for elliptic curve groups (this is called ECNTT in this library), but, unfortunately this fast arithmetic is not available for the Grumpkin curve which forms a 2-cycle with the BN254 curve. Hence, all systems relying on these are designed in such a way that they do not require NTT in the exponent on the second curve.

Removing this constraint may open new design possibilities for cryptographic protocols.

Summary of Contributions

- We develop an *algebraic formalism* of group polynomials, which provides an appropriate model for arithmetic in the exponent. This is presented in a general setting that allows us to later build the algebraic structures required for our algorithms. It includes extending a group of exponent q by a polynomial over $\mathbb{F}_q$, in a way that is similar to field extensions. While this is somewhat elementary from abstract algebra, to our knowledge, the application to polynomial arithmetic in the exponent and associated algorithms is new.
- Thanks to this framework, we transpose several existing polynomial algorithms (Karatsuba, Toom-Cook, NTT, Schönhage-Strassen) to the case of polynomial arithmetic in the exponent. The cost model is quite different from the classical one, and in particular, the $\log \log n$ factor that arises with the classical Schönhage-Strassen algorithm is applied to operations that are negligible. In total, the theoretical slow-down compared with the NTT case is only a factor of about 2. We also cover the case where the roots of unity are present in $\mathbb{F}_{q^2}$, which is the case for the curve of [10] whose order is the base field of NIST-P256. We show that, here, the loss is by a factor $3/2$ compared to the NTT case.
- As an application, we show that the batched KZG proof and Bayer-Groth algorithms keep their quasi-linear complexities, even when instantiated with legacy elliptic curves that are not NTT-compatible.

Related Work The literature of polynomial arithmetic over fields or rings is rich, dating back to the 60’s and 70’s with Karatsuba’s [14], Toom-Cook’s [23] and Schönhage-Strassen’s [18] algorithms. A very good survey of the state-of-the-art at the turn of the century is the book of von zur Gathen and Gerhard [12]. Another rich source of inspiration for this work is the literature specialized on the discrete Fourier Transform, which is often called Number Theoretic Transform (NTT) when the base ring or field is no longer the field of complex numbers. We refer to the survey article [6] for numerous references in this direction. These include many discussions on how to handle the cases where the roots of unity are not directly available in the base ring or field.

There is still recent work on this topic, both from theoretical and practical points of view. We mention in particular the recent work called ECFFT [4, 5], where the authors use the theory of isogenies on elliptic curves to save a $\log \log$ factor in the complexity compared to Schönhage-Strassen’s algorithm, and apply this to Interactive Oracle Proofs (IOP) algorithms. See also the concept of elliptic butterflies, introduced in [9].

As for fast polynomial arithmetic *in the exponent*, the literature is sparse. In [3], Bayer and Groth needed it for claiming a quasi-linear complexity for their

algorithm. They propose the use of NTT, assuming the presence of roots of unity. They also have a practical variant based on a few levels of recursive Toom-Cook algorithm. However, their approach was not purely arithmetic in the exponent, since they had to add an interaction with the verifier at each level of recursion. Therefore, it is not a method that behaves as a black-box replacement in any protocol.

The work of Feist and Khovratovich [11] is perhaps the closest to ours. In their paper, they use classical algorithms for multi-point evaluation and interpolation and restate them in the “in the exponent” case, while still assuming the presence of roots of unity, so that NTT is directly available. We note, for instance, that in their Appendix A.2, they provide a version of our Proposition 2 in a specific case.

In our work, the cost model for evaluating the efficiency of polynomial arithmetic is deeply changed, where linear operations on the coefficients of the two input polynomials can correspond to completely different costs, since they take place in different groups. Also, the cost of the bilinear operations can be different. This is reminiscent of the algorithm by D. V. and G. V. Chudnovsky [2, 8], where only the bilinear complexity is taken into account. While it is an evaluation/interpolation scheme that would fit in the setting of our Theorem 1, this family of algorithms is not efficient for our cost model.

Organization of the Paper In Section 2 we develop our algebraic formalism for group polynomials; this includes an abstract evaluation/interpolation setting that is the prototype of many multiplication algorithms. Section 3 contains our main results, with fast algorithms for polynomial arithmetic in the exponent. We give precise asymptotic costs, counting operations in all the groups involved. In Section 4 we apply these complexity results to the case of KZG and Bayer-Groth algorithms. Finally, in Section 5 we describe our proof-of-concept implementation that validates our claims and provides benchmarks illustrating the quasi-linear complexity.

2 Algebraic Structures with Group Polynomials

All over this section, q is a prime number and all the groups that we consider are abelian groups of exponent q (i.e. all non-neutral elements have order q), for which we use the additive notation. We typically denote such a group by $\mathbb{G}$ and its neutral element by $0_{\mathbb{G}}$.

The formalism introduced in this section is “elementary” in the sense that, while we use some abstractions, the results are intuitively “natural”; the proofs are simple and just follow the definitions and the formulas therein. Most of the definitions and propositions could be stated with tools of pure abstract algebra, we prefer to present them in a rather concrete way, so that the transcription into algorithms becomes transparent. In particular, we wish to state that multiplication of polynomials in the exponent can be done with an evaluation/interpolation scheme, in a more general setting than just the NTT.

This allows us to then combine it with the technique of extension of scalars to incorporate roots of unity when there are not enough of them for NTT. Again, one could just state this using the language of tensor products, but we prefer to construct it with a more concrete approach that makes the algorithmic counterpart more visible.

2.1 Definition and Convolution Pairing

Definition 1. *The set of **group polynomials** over a group $\mathbb{G}$, denoted by $\mathbb{G}[x]$ is the abelian group of sequences $(p_i)_{i \geq 0}$ of elements of $\mathbb{G}$ that have finitely many non-zero coefficients. The **degree** of a group polynomial is the largest index d for which $p_d \neq 0_{\mathbb{G}}$, possibly equal to $-\infty$ for the all- $0_{\mathbb{G}}$ sequence. It is denoted by $p_0 + p_1x + \dots + p_dx^d$.*

The subgroup of elements of degree at most d is denoted by $\mathbb{G}[x]_{\leq d}$. If $\mathbb{G}$ is a finite group, then $\mathbb{G}[x]_{\leq d}$ is also finite.

Remark 1. When $\mathbb{G}$ is cyclic, with a distinguished generator g of order q , all elements can be represented by their discrete logarithms in base g . It is usual in the literature of group polynomials to use the notation $p(x) = [a_0] + [a_1]x + \dots + [a_d]x^d$, where a_i is the discrete logarithm of the i -th coefficient $p_i = g^{a_i}$. This presents the advantage of turning all groups into an additive notation. We will not use this notation in this paper, because in our applications, some groups are not cyclic. Still, using the additive notation for $\mathbb{G}$ makes the polynomial notation more intuitive.

Addition of group polynomials is the natural term-by-term addition in $\mathbb{G}$; however there is no way to multiply two group polynomials without an additional structure, namely a pairing, that we will now define with some generality to cover our needs.

Any finite abelian group of exponent q is a $\mathbb{Z}/q\mathbb{Z}$ -module (and actually an $\mathbb{F}_q$ -vector space, since q is prime). More generally, we generalize this to the context where groups are $\mathbb{A}$ -modules, for a commutative ring $\mathbb{A}$.

Definition 2. *Let $\mathbb{A}$ be a ring and $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ be groups that are $\mathbb{A}$ -modules. An **$\mathbb{A}$ -pairing** is a map $\star : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ that is $\mathbb{A}$ -linear in both components.*

We now introduce the convolution pairing, which is the key ingredient that provides the group polynomials with a product operation close to the one we get for polynomials over a ring.

Definition 3. *Let $\mathbb{A}$ be a ring and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ be groups that are $\mathbb{A}$ -modules, together with an $\mathbb{A}$ -pairing $\star : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$.*

*The **convolution pairing** $\mathbb{G}_1[x] \times \mathbb{G}_2[x] \rightarrow \mathbb{G}_3[x]$, also denoted by $\star$ is defined as follows. If $P(x) = \sum p_i x^i \in \mathbb{G}_1[x]$ and $Q(x) = \sum q_i x^i \in \mathbb{G}_2[x]$, then $P(x) \star Q(x)$ is the polynomial $R(x) = \sum r_i x^i \in \mathbb{G}_3[x]$, with $r_i = \sum_{0 \leq k \leq i} (p_k \star q_{i-k})$.*

There are typical examples covered by this definition.

- **Ring case.** When $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}_3$ is not only a group, but also a ring, the multiplication in $\mathbb{G}_1$ provides a pairing. In that case, the convolution pairing is the classical polynomial multiplication in $\mathbb{G}_1[x]$.
- **Canonical pairing.** When $\mathbb{G}$ is an $\mathbb{A}$ -module, then the module external law provides an $\mathbb{A}$ -pairing $\mathbb{G} \times \mathbb{A} \rightarrow \mathbb{G}$, that we call canonical and which extends to what we call the **canonical convolution pairing** from $\mathbb{G}[x] \times \mathbb{A}[x]$ to $\mathbb{G}[x]$.
- **$\mathbb{Z}_q$ -module case.** If $\mathbb{G}$ is a group of prime exponent q , then it is a $\mathbb{Z}_q$ -module, and the canonical convolution pairing is the product of group polynomials $\mathbb{G}[x] \times \mathbb{F}_q[x] \rightarrow \mathbb{G}[x]$ that is used in most applications.

By analogy with the ring case, we often speak about group polynomial multiplication for the convolution pairing.

The following proposition shows that the convolution pairing is enough to build additional structure on top of it (see Section 2.3).

Proposition 1. *If $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ are $\mathbb{A}$ -modules, then the groups $\mathbb{G}_1[x], \mathbb{G}_2[x], \mathbb{G}_3[x]$ are $\mathbb{A}[x]$ -modules, and the convolution pairing between them is an $\mathbb{A}[x]$ -pairing (and therefore, also an $\mathbb{A}$ -pairing).*

Proof. Let $\mathbb{G}$ be an $\mathbb{A}$ -module. Then the canonical convolution pairing $\star : \mathbb{G}[x] \times \mathbb{A}[x] \rightarrow \mathbb{G}[x]$ gives an external law from $\mathbb{A}[x]$ on $\mathbb{G}[x]$. The formula in its definition follows exactly the one of polynomial multiplication, so that this map endows $\mathbb{G}[x]$ with an $\mathbb{A}[x]$ -module structure.

Now, since $\mathbb{G}_1[x], \mathbb{G}_2[x], \mathbb{G}_3[x]$ are $\mathbb{A}[x]$ -modules, we can check whether the convolution pairing between them is an $\mathbb{A}[x]$ -pairing. Again, this follows directly from the fact that the formula mimics the one from polynomial multiplication. $\square$

2.2 Evaluation and Interpolation

Definition 4. *Let $\mathbb{G}$ be a group that is an $\mathbb{A}$ -module. For any $\omega \in \mathbb{A}$, the **evaluation map** $\text{ev}_\omega : \mathbb{G}[x] \rightarrow \mathbb{G}$ is defined as follows. For $P(x) = \sum p_i x^i \in \mathbb{G}[x]$,*

$$\text{ev}_\omega(P) = P(\omega) = \sum p_i \omega^i.$$

The evaluation map is $\mathbb{A}$ -linear as follows directly from its definition.

Let K be a positive integer, and let $\Omega = (\omega_i)_{0 \leq i < K}$ be a set of K distinct elements of $\mathbb{A}$. We can extend the evaluation map to the $\mathbb{A}$ -linear map $\text{ev}_\Omega : \mathbb{G}[x]_{<K} \rightarrow \mathbb{G}^K$ defined by $\text{ev}_\Omega(P) = (\text{ev}_{\omega_i}(P))_{0 \leq i < K}$. The matrix of this linear map is the Vandermonde matrix of Ω . When its determinant is invertible (which is the case when for all $i \neq j$, the element $(\omega_i - \omega_j)$ is invertible in $\mathbb{A}$), then this evaluation map has an inverse, denoted by interp_Ω and called the **interpolation map**.

These maps allow us to define a general multiplication scheme, upon which the whole asymptotically fast polynomial arithmetic is based.

Theorem 1. Let $\mathbb{A}$ be a ring and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ be groups that are $\mathbb{A}$ -modules, together with an $\mathbb{A}$ -pairing. Let d be a positive integer and Ω be a sequence of K distinct elements of $\mathbb{A}$ for which the Vandermonde determinant is invertible, with $K \geq 2d - 1$. The following diagram (with the solid arrows, not the dotted one) commutes.

$$\begin{array}{ccccc}
 \mathbb{G}_1[x]_{<d} \times \mathbb{G}_2[x]_{<d} & \longrightarrow & \mathbb{G}_1[x]_{<K} \times \mathbb{G}_2[x]_{<K} & \xrightarrow{\text{ev}_\Omega, \text{ev}_\Omega} & \mathbb{G}_1^K \times \mathbb{G}_2^K \\
 & \searrow \star & \downarrow \varphi & \text{pt-wise } \star \downarrow & \\
 & & \mathbb{G}_3[x]_{<K} & \xleftarrow{\text{interp}_\Omega} & \mathbb{G}_3^K
 \end{array}$$

In this diagram, the first horizontal map is just an inclusion, since polynomials of degree less than d are polynomials of degree less than $2d - 1 \leq K$.

The dotted vertical arrow φ is ill-defined, because the resulting polynomial has a degree that can be larger than K . This can be resolved by reducing the result modulo an appropriate polynomial. The notion of modular arithmetic of group polynomials will be given below to this effect.

Proof. What needs to be shown is that for $P(x) \in \mathbb{G}_1[x]$ and $Q(x) \in \mathbb{G}_2[x]$, we have the following equality, in $\mathbb{G}_3$: $\text{ev}_\omega(P) \star \text{ev}_\omega(Q) = \text{ev}_\omega(P \star Q)$. This follows directly from the formulae. $\square$

Following up with the examples above:

- **Ring case.** When $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}_3 = \mathbb{A}$ is not only a group, but also a ring, this is the classical evaluation/interpolation scheme that covers the Toom-Cook algorithm, and, when the ω_i are appropriate roots of unity in $\mathbb{A}$, this covers the multiplication by NTT.
- **$\mathbb{Z}_q$ -module case.** When the ω_i are appropriate roots of unity of $\mathbb{F}_q$, this is the product of group polynomials $\mathbb{G}[x] \times \mathbb{F}_q[x] \rightarrow \mathbb{G}[x]$ done by NTT, that is often used in applications.

Later on, we will need to evaluate group polynomials not only at elements of $\mathbb{A}$, but also at polynomials in $\mathbb{A}$. This is commonly called composition of polynomials. We give a definition that is restricted to composition with monomials, since this is simpler to write and covers our needs.

Definition 5. Let $\mathbb{G}$ be a group that is an $\mathbb{A}$ -module. The **composition with a monomial** of a group polynomials is defined as follows. Let $P(x) = \sum p_i x^i \in \mathbb{G}[x]$ be a group polynomial and $m(x) = \alpha x^k$ be a monomial in $\mathbb{A}[x]$, then the composition of P and m is

$$P(m(x)) = \sum p_i \alpha^i x^{ik}.$$

This is a group polynomial of degree at most $k \deg P$.

For a fixed monomial $m(x)$, the composition with m is an $\mathbb{A}$ -linear map.

2.3 Modular Arithmetic

Let $\mathbb{A}$ be a ring and $\mathbb{G}$ be a group that is an $\mathbb{A}$ -module. The $\mathbb{A}[x]$ -module structure of $\mathbb{G}[x]$ given by Prop. 1 can be further enriched with a notion of Euclidean division.

Proposition 2. *For any group polynomials $A(x) \in \mathbb{G}[x]$ and any monic polynomial $B(x) \in \mathbb{A}[x]$, one can define $(Q(x), R(x)) \in \mathbb{G}[x]^2$, the quotient and remainder of the **Euclidean division** of $A(x)$ by $B(x)$ as the unique polynomials verifying $A(x) = B(x) \star Q(x) + R(x)$, with $\deg R < \deg B$.*

The proof follows the classical Euclidean division of polynomials; we give it here for completeness. This also demonstrates how the fact that while the polynomials $A(x)$ and $B(x)$ do not live in the same algebraic structure, the classical proof carries over easily, and this will therefore be the same for the associated algorithm to divide A by B .

Proof. Let us start with uniqueness. Assume that $A(x) = B(x) \star Q_1(x) + R_1(x) = B(x) \star Q_2(x) + R_2(x)$, with $\deg R_1 < \deg B$ and $\deg R_2 < \deg B$. Then, by linearity of $\star$, we have $B(x) \star (Q_1(x) - Q_2(x)) = R_2(x) - R_1(x)$. On the right-hand-side, the degree of $R_2 - R_1$ is less than $\deg B$. On the left-hand-side, since B is monic, the degree is $\deg B + \deg(Q_1 - Q_2)$, which is at least $\deg B$ as soon as $Q_1 - Q_2 \neq 0$. Combining both leads to $Q_1 = Q_2$, and then it follows that $R_1 = R_2$.

The existence is proven by induction on the degree of A . When $\deg A < \deg B$, then we can take $Q = 0$ and $R = A$. Let us assume that the existence is given for A 's of degree $\leq n - 1$, and let A be a polynomial of degree $n \geq \deg B$. The leading term of A is $\text{lc}(A)x^{\deg A} \in \mathbb{G}[x]$, where $\text{lc}()$ denotes the leading coefficient. Since B is monic, the polynomial $A_0(x) = \text{lc}(A)x^{\deg A - \deg B} \star B(x)$ is of degree $\deg A$ and has the same leading term as A . Therefore $A' = A - A_0$ is of degree $\deg A' < n$ and we can apply the induction hypothesis to it to get Q' and R' . Taking $R = R'$ and $Q(x) = Q'(x) + \text{lc}(A)x^{\deg A - \deg B}$ gives the result. $\square$

Note that this can be extended to the case where $B(x)$ is not monic but its leading coefficient is invertible in $\mathbb{A}$.

Notation. If we are not interested in the quotient but only in the remainder, we can use the standard notation $R(x) = A(x) \bmod B(x)$.

Remark 2. It is now possible to properly define the map φ corresponding to the dotted arrow in Theorem 1 in a way that makes the square part of the diagram commute. It is the convolution pairing of the two input polynomials, followed by a reduction modulo the polynomial $\prod (x - \omega_i) \in \mathbb{A}[x]$.

This example goes in the direction of defining a structure for modular arithmetic. This is what we formalize now.

Proposition 3. *Let $M(x)$ be a monic polynomial of degree d in $\mathbb{A}[x]$. Then the group of polynomials in $\mathbb{G}[x]$ of degree less than d can be endowed with a structure of module over $\mathbb{A}[x]/(M(x))$. This module is denoted by $\mathbb{G}[x]/(M(x))$.*

Proof. Given $P(x) \in \mathbb{G}[x]$ with $\deg P < d$, and $Q(x) \in \mathbb{A}[x]$, the $\mathbb{A}[x]$ -module structure gives $P(x) \star Q(x)$ in $\mathbb{G}[x]$, and we can take its remainder in the division by $M(x)$, giving another polynomial in $\mathbb{G}[x]$ of degree less than d . From the definition of Euclidean division, the result depends only on the class of $Q(x)$ in $\mathbb{A}[x]/(M(x))$. This is linear, and therefore provides a module structure. $\square$

Remark 3. In abstract algebra, this construction is known as extension of scalars of modules, obtained as the tensor product $\mathbb{G} \otimes_{\mathbb{A}} (\mathbb{A}[x]/(M(x)))$.

Proposition 4. *Let $\mathbb{A}$ be a ring and let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ be groups that are $\mathbb{A}$ -modules, together with an $\mathbb{A}$ -pairing. Let $M(x)$ be a monic polynomial in $\mathbb{A}[x]$. Then it is possible to extend the $\mathbb{A}$ -pairing to an $(\mathbb{A}[x]/(M(x)))$ -pairing*

$$\star : \mathbb{G}_1[x]/(M(x)) \times \mathbb{G}_2[x]/(M(x)) \rightarrow \mathbb{G}_3[x]/(M(x)).$$

Proof. The pairing is obtained from the convolution pairing. The reduction modulo $M(x)$ commutes with all the operations, yielding the claimed result. $\square$

Remark 4. We have extended the groups via modular arithmetic, in such a way that the new larger groups fit again in Definition 2, so that we can apply all the results to these new groups. This allows then to construct a stack of extensions, just like what is usual with field extensions.

The main application we have in mind is the case where $\mathbb{F}_q$ does not offer the roots of unity required for NTT-based arithmetic. Then, the groups, which are naturally $\mathbb{F}_q$ -modules can be extended to larger groups over a ring $\mathbb{A}$ that is an extension of $\mathbb{F}_q$ containing the appropriate roots of unity.

3 Asymptotically Fast Group Polynomial Arithmetic

In this section, we revisit well-known polynomial multiplication algorithms to adapt them to the case of group polynomials. The cost model is quite different from the case of polynomials over a ring, and we separate the costs in each of the three group polynomials and the cost of the pairing. In what follows, all the logarithms are taken in base 2.

3.1 Setting and Cost Model

We consider three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$, of prime exponent q , together with a $\mathbb{Z}_q$ -linear pairing $\star : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$. We want to evaluate the cost of computing the convolution pairing between two group polynomials $P(x) \in \mathbb{G}_1[x]$ and $Q(x) \in \mathbb{G}_2[x]$.

We assume that the 3 groups are effective, in the sense that we have efficient algorithms to compute the group laws in them. More precisely, we will take as unit costs scalar multiplications in them: M_1, M_2, M_3 , respectively. We also assume that the pairing is effective, and we denote by B its cost. It will sometimes

be possible to rewrite B in terms of scalar multiplications, but we prefer to keep it separate.

The multiplication algorithms also involve group operations (adding two elements). Such an operation has a negligible cost compared to a scalar multiplication in the same group; unless the number of them is much higher than the number of scalar multiplications, their contribution in the global cost will remain negligible. In the cases where this might be important, we denote the associated costs by A_1 , A_2 , A_3 .

For simplicity, we assume that negating an element is free in the groups. This is true for elliptic curves, but not for multiplicative groups of finite fields. The former are the most important for applications.

The relative values of A_i , M_i and B will vary a lot depending on the applications. We give an important example.

Elliptic Canonical Convolution Pairing This is the case when the group $\mathbb{G}$ is the group (or a subgroup) of points of an elliptic curve of order q , and where the goal is to compute the canonical convolution pairing between $P(x) \in \mathbb{G}[x]$ and $Q(x) \in \mathbb{F}_q[x]$ (the group $\mathbb{G}_2$ is the additive group $\mathbb{Z}_q$, which is also the field $\mathbb{F}_q$). In that case:

- $B = M_1 = M_3$ is the cost of a scalar multiplication in the elliptic curve that we denote M_{ell} .
- M_2 is the cost of an integer multiplication modulo q .
- $A_1 = A_3$ is the cost of adding two points in $\mathbb{G}$ and A_2 is an integer addition modulo q .

Therefore, in that case, we have M_2 , A_1 , A_2 and A_3 that are cheaper than $M_{\text{ell}} = B = M_1 = M_3$ by (typically) two orders of magnitude.

3.2 Schoolbook Algorithm

The schoolbook algorithm follows the formula of Definition 3

$$r_i = \sum_{0 \leq k \leq i} (p_k \star q_{i-k}),$$

where $P(x) = \sum p_i x^i$, $Q(x) = \sum q_i x^i$ and $P(x) \star Q(x) = \sum r_i x^i$. When $\deg P$ and $\deg Q$ have degree at most $d-1$, then there are d^2 pairings to compute and $d^2 - 2d + 1$ operations in $\mathbb{G}_3$ to be done in order to accumulate them in the $2d-1$ resulting values r_i .

We obtain:

Proposition 5. *The multiplication of group polynomials of degree $d-1$ by the schoolbook algorithm has a cost $d^2 B + (d-1)^2 A_3$.*

It is frequent that $A_3 \ll B$, so that the second term is negligible.

3.3 NTT-Based Multiplication

Let K be a power of 2, and assume that there exists a K -th primitive root of unity ω in $\mathbb{F}_q$, which is the case if and only if $q - 1$ is divisible by a power of 2 that is large enough for the polynomials we wish to multiply. We consider $\Omega = (\omega^i)_{0 \leq i < K}$ and apply the scheme of Theorem 1 for polynomials of degree less than $d = K/2$.

The cost of this multiplication algorithm is directly related to the cost of the evaluation and interpolation maps. The fact that the famous Fast Fourier Transform (FFT) algorithm works *mutatis mutandis* in the context of group polynomials is well-known.

We recall here the associated costs, in a slightly more general form (see for instance [12, Thm 8.15]).

Theorem 2. *Let $\mathbb{A}$ be a ring in which 2 is invertible, $\mathbb{G}$ be a group that is also an $\mathbb{A}$ -module, and $\Omega = (\omega^i)_{0 \leq i < K}$, where ω is a K -th primitive root of unity in $\mathbb{A}$, with K a power of 2.*

The map $\text{ev}_\Omega : \mathbb{G}[x]_{<K} \rightarrow \mathbb{G}^K$ can be computed at a cost of $\frac{1}{2}K \log K$ multiplications by powers of ω , and $K \log K$ operations in $\mathbb{G}$.

The inverse map interp_Ω can be computed at a cost of $\frac{1}{2}K \log K$ multiplications by powers of ω , K divisions by a power of 2, and $K \log K$ operations in $\mathbb{G}$.

In the classical case where $\mathbb{A}$ is $\mathbb{F}_q$, the root ω is usually an element that has no special structure, so that a multiplication by a power of ω amounts to a full scalar multiplication in $\mathbb{G}$. This is also the case for division by a power of 2.

We obtain:

Proposition 6. *Let K be a power of 2. Assuming that the exponent q of the groups is such that $K|q - 1$, then the multiplication of group polynomials of degree less than K modulo $x^K - 1$, by the NTT algorithm has a cost $K\mathbf{B} + \frac{1}{2}K \log K(\mathbf{M}_1 + \mathbf{M}_2 + \mathbf{M}_3)$, plus negligible terms.*

In the case of the elliptic canonical convolution pairing, the cost becomes $K \log K \mathbf{M}_{\text{ell}}$ (plus negligible costs).

3.4 Karatsuba and Toom-Cook Algorithms

Let P and Q be two polynomials of degree less than d , where $d = 2d'$ is even. Let $P(x) = P_0(x) + x^{d'}P_1(x)$, where P_0 and P_1 have degrees less than d' , and similarly for Q_0 and Q_1 . By bilinearity of $\star$, we have $P(x) \star Q(x) = (P_0 \star Q_0)(x) + (P_0 \star Q_1 + P_1 \star Q_0)(x)x^{d'} + (P_1 \star Q_1)x^d$. The middle term can be rewritten as $((P_0 + P_1) \star (Q_0 + Q_1) - P_0 \star Q_0 - P_1 \star Q_1)(x)$. Therefore the product of P and Q can be obtained at a cost of 3 subproducts in degree d' , plus a number of additions. Calling this recursively, this yields a cost of $d^{\log(3)}\mathbf{B} + O(d^{\log(3)})(\mathbf{A}_1 + \mathbf{A}_2 + \mathbf{A}_3)$ for Karatsuba's algorithm.

It is possible to interpret this scheme as an evaluation/interpolation at the points 0, 1, ∞ , but this does not fit exactly in Theorem 1, due to the need of a projective point of view.

The Toom-Cook algorithm follows the same idea, but evaluating at more points, thus allowing us to decompose P and Q in k parts of degree less than d/k instead of just 2. The cost is then $2k - 1$ products in size d/k . One has to add the costs of the evaluation and interpolation steps. Choosing evaluation points that are small elements, the evaluations involve only $O(d)(M_1 + M_2)$ (the constant in the $O()$ grows quadratically in k , but for fixed k this is not visible). However, for $k > 2$, it is not possible to have an interpolation map that also involves only small elements, as was the case for Karatsuba. In general, one will have to divide the resulting data by the determinant of the evaluation matrix. This leads to full scalar multiplication in $\mathbb{G}_3$.

The scheme is called recursively, but this normalization can be done at the very end of the computation: one can divide all the coefficients by the determinant raised to the number of recursive calls. It therefore contributes as $2dM_3$ in the total cost.

Summing up the costs leads to $d^{\log(2k-1)/\log(k)} B + O(d^{\log(2k-1)/\log(k)})(A_1 + A_2 + A_3) + 2dM_3$, with the constants within the $O()$ growing quadratically in k . In practice, for polynomial multiplications over fields or integers, only $k = 3$ or $k = 4$ are used.

Proposition 7. *The multiplication of group polynomials of degree $d - 1$ by the Karatsuba algorithm has a cost $d^{\log(3)} B + O(d^{\log(3)})(A_1 + A_2 + A_3)$.*

The multiplication of group polynomials of degree $d - 1$ by the Toom-Cook algorithm for fixed level k has a cost $d^{\log(2k-1)/\log(k)} B + O(d^{\log(2k-1)/\log(k)})(A_1 + A_2 + A_3) + 2dM_3$.

3.5 NTT in $\mathbb{F}_{q^2}$

In the case where $q + 1$ is divisible by a large power of 2 (and therefore, $q - 1$ is not), then we can find roots of unity in an extension of degree 2 of $\mathbb{F}_q$. It is classical to take advantage of this for multiplying polynomials in $\mathbb{F}_q[x]$. We show here that the same strategy works for group polynomials.

Algorithm using Extensions of Group Polynomials Let $K \geq 4$ be a power of 2 that divides $q + 1$, and let ω be a K -th primitive root of unity in $\mathbb{F}_{q^2}$. Let $M(t) = t^2 + 1$; since $q \equiv 3 \pmod{4}$, then -1 is not a square in $\mathbb{F}_q$, and $M(t)$ is irreducible in $\mathbb{F}_q[t]$. We then apply Proposition 3 to construct the groups $\tilde{\mathbb{G}}_i = \mathbb{G}_i[t]/(M(t))$ which are $\mathbb{F}_{q^2}$ -modules, and we extend the pairing using Proposition 4.

We are now in a setting where Theorem 1 can be applied, with $\Omega = (\omega^i)_{0 \leq i < K}$. Let $P(x) \in \mathbb{G}_1[x]$ and $Q(x) \in \mathbb{G}_2[x]$ be two group polynomials of degree less than d , with $K \geq 2d - 1$. Let $\tilde{P}(x)$ be the polynomial obtained from $P(x)$ by sending each coefficient $p_i \in \mathbb{G}_1$ to $p_i + 0_{\mathbb{G}_1} t \in \tilde{\mathbb{G}}_1$, and similarly for $\tilde{Q}(x)$. We can multiply $\tilde{P}$ and $\tilde{Q}$ with K applications of the pairing $\tilde{\mathbb{G}}_1 \times \tilde{\mathbb{G}}_2 \rightarrow \tilde{\mathbb{G}}_3$, and evaluations and interpolation maps at Ω .

The pairing in the extended groups can be computed by first computing a degree-1 product of group polynomials, followed by a reduction modulo $t^2 + 1$,

which is cheap, thanks to the special form. Using Karatsuba scheme, this yields $3B + O(1)(A_1 + A_2 + A_3)$.

As for the evaluations and interpolation maps, we apply Theorem 2 in $\tilde{G}_i$ over $\mathbb{F}_{q^2}$. A multiplication by powers of ω in $\tilde{G}_i$ can, again, be performed using Karatsuba scheme, leading to a cost of $3M_i + O(1)A_i$. A division by a power of 2, which is in $\mathbb{F}_q$ and not $\mathbb{F}_{q^2}$ costs only $2M_i$.

Combining all of these, the overall cost of NTT-based multiplication in $\mathbb{F}_{q^2}$ is $3KB + \frac{3}{2}K \log K (M_1 + M_2 + M_3)$.

Saving an Additional Factor 2 There is an analogy between group polynomials over groups of the form $\tilde{G}_i = \mathbb{G}_i[t]/(t^2 + 1)$ and polynomials over the complex numbers $\mathbb{C}$. In the latter case, the literature on the discrete Fourier transform is rich. In particular, it contains improvements when the polynomials have coefficients in $\mathbb{R}$. In that case, the transforms of the polynomials have symmetries and there exist algorithms to compute them with a transform of half the length. We are now going to describe our adaptation of these algorithms to the case of group polynomials when the roots of unity are in $\mathbb{F}_{q^2}$.

We continue with the same notation as above, where $t \in \mathbb{F}_{q^2}$ plays a role similar to $i \in \mathbb{C}$. For an element $z = z_0 + tz_1$, we denote by $\bar{z} = z_0 - tz_1$ the Galois conjugate of z , and we use the real/imaginary parts terminology with the usual notation $\text{Re}(z) = z_0$ and $\text{Im}(z) = z_1$.

It can be shown that $\bar{\omega} = 1/\omega$. Indeed, since K divides $q+1$, then $2K$ divides q^2-1 , and there exists a primitive $2K$ -th root of unity $\omega_0 \in \mathbb{F}_{q^2}$. The conjugation is an automorphism, so that $\bar{\omega}_0$ is also a $2K$ -th root of unity. Therefore $\omega_0 \bar{\omega}_0$ is a $2K$ -th root of unity. Since this quantity belongs to $\mathbb{F}_q$, it is a 2-nd root of unity, hence ± 1 . Squaring the equation $\omega_0 \bar{\omega}_0 = \pm 1$ gives the result.

Next, we have that $\omega^{K/4} = \pm t$. Without loss of generality, possibly replacing ω by $\bar{\omega}$, we can assume that $\omega^{K/4} = t$.

Proposition 8. *Let $P(x)$ be a group polynomial over $\mathbb{G}$ of degree less than K and let $\omega \in \mathbb{F}_{q^2}$ a K -th root of unity with $\bar{\omega} = 1/\omega$. Then the image $\hat{P} = (\hat{p}_i)_{0 \leq i < K}$ by the ev_Ω map in $\tilde{G}^K$ for $\Omega = (\omega^i)_{0 \leq i < K}$ has the following properties:*

- $\hat{p}_0$ and $\hat{p}_{K/2}$ are “real”, i.e. their imaginary part is 0.
- for all $0 \leq i < K$, we have $\bar{\hat{p}}_i = \hat{p}_{-i \bmod K}$.

The key ingredient is the $\text{DFT}_{\mathbb{R}}$ algorithm, given in Algorithm 1. Given a group polynomial P in $\mathbb{G}[x]$, it defines a group polynomial $\tilde{P}$ in $\tilde{G}[x]$, by just grouping two consecutive coefficients of P to form the real and imaginary parts of a coefficients of $\tilde{P}$. Therefore, the degree of $\tilde{P}$ is halved. Its discrete Fourier transform is computed as the image of the evaluation map $\text{ev}_{\Omega'}$, where $\Omega' = (\omega^{2i})_{0 \leq i < K/2}$. This is a transform of length $K/2$. Then, the evaluation of P at Ω can be deduced by a sparse $\mathbb{F}_q$ -linear map: each entry can be obtained as an $\mathbb{F}_q$ -linear combination of at most two elements of the evaluation of $\tilde{P}$ at Ω' .

Proposition 9. *With the hypothesis that $\bar{\omega} = \omega^{-1}$ and $\omega^{K/4} = t$, Algorithm 1 is correct.*

Proof. (sketch) This algorithm is well-known in the case of real/complex numbers [19]. The proof carries over to finite fields, under the hypothesis that ω indeed behaves like $\exp 2i\pi/K$ with respect to conjugation and its relation to i . It also works *mutatis mutandis* over group polynomials. $\square$

There is an algorithm for the interpolation, or the inverse discrete Fourier transform, that is the inverse of Algorithm 1. First, it does a linear transform that is the inverse of the “Recombine” part; second, it does an interpolation with Ω' , of length $K/2$; and third, it reorganize the real and imaginary parts of the coefficients to give a group polynomial in $\mathbb{G}[x]$. We refer to the function `invDFTreal_Fq2()` in the SageMath artifact provided (see Appendix) for this algorithm that has the same cost as Algorithm 1.

Algorithm 1: $\text{DFT}_{\mathbb{R}}$: Discrete Fourier Transform of a “Real” group polynomial

Input: $P(x)$ in $\mathbb{G}[x]$, with $\deg P < K$ and $\omega \in \mathbb{F}_{q^2}$ a K -th root of unity
Output: $\text{ev}_{\Omega}(P)$, for $\Omega = (\omega^i)_{0 \leq i < K}$

```

// Create a group polynomial  $\tilde{P}$  over  $\tilde{\mathbb{G}}$ ;
Write  $P(x) = \sum_{0 \leq i < K} p_i x^i$ , with  $p_i \in \mathbb{G}$ ;
for  $i \leftarrow 0$  to  $K/2 - 1$  do
    | Set  $\tilde{p}_i = p_{2i} + t p_{2i+1}$ ;
end
Write  $\tilde{P} = \sum_{0 \leq i < K/2} \tilde{p}_i x^i$ ;
// Compute the Fourier transform of  $\tilde{P}$  in length  $K/2$ ;
Compute  $\hat{\tilde{P}} = \text{ev}_{\Omega'}(\tilde{P})$ , for  $\Omega' = (\omega^{2i})_{0 \leq i < K/2}$ ;
Write  $\hat{\tilde{P}} = (\hat{\tilde{p}}_i)_{0 \leq i < K/2}$ ;
// Recombine to the transform of  $P$ ;
 $\hat{p}_0 \leftarrow \text{Re}(\hat{\tilde{p}}_0) + \text{Im}(\hat{\tilde{p}}_0)$ ;  $\hat{p}_{K/2} \leftarrow \text{Re}(\hat{\tilde{p}}_0) - \text{Im}(\hat{\tilde{p}}_0)$ ;
 $\hat{p}_{K/4} \leftarrow \hat{\tilde{p}}_{K/4}$ ;  $\hat{p}_{3K/4} \leftarrow \hat{\tilde{p}}_{K/4}$ ;
for  $i \leftarrow 1$  to  $K/2 - 1$ , with  $i \neq K/4$  do
    |  $a_1 \leftarrow \hat{\tilde{p}}_i \overline{(1 + \omega^{K/4-i})}$ ;
    |  $a_2 \leftarrow \hat{\tilde{p}}_{K/2-i} (1 + \omega^{K/4+i})/2$ ;
    |  $\hat{p}_i \leftarrow a_1 + a_2$ ;  $\hat{p}_{K-i} \leftarrow \overline{\hat{p}_i}$ ;
end
return  $\text{ev}_{\Omega}(P) = (\hat{p}_i)_{0 \leq i < K}$ .

```

Using these two algorithms, the cost of the transforms has been divided by 2. Furthermore, using the symmetries, the number of point-wise multiplications (and therefore the number of calls to the pairing) can also be divided by 2.

We obtain:

Proposition 10. *Let K be a power of 2. Assuming that the exponent q of the groups is such that $K|q + 1$, then the multiplication of group polynomials of*

degree less than K modulo $x^K - 1$, by the NTT algorithm over $\mathbb{F}_{q^2}$ has a cost $\frac{3}{2}KB + K \log K(M_1 + M_2 + M_3)$, plus negligible terms.

This incurs only a factor-1.5 overhead compared with the case where the root of unity lies in $\mathbb{F}_q$.

Remark 5. The approach can be generalized to the case where a large power of 2 divides $\Phi_d(q)$, where Φ_d is the d -th cyclotomic polynomial, by working in an extension of degree d . We did not investigate this approach further, because we did not find any cryptographic group in the literature with such a property, and anyway, if $d \geq 3$, the Schönhage-Strassen algorithm will most likely be faster.

3.6 Schönhage-Strassen Algorithm

The Schönhage-Strassen algorithm for multiplying polynomials over a ring $\mathbb{A}$ relies on an algebraic structure of the form $(\mathbb{A}[x]/(x^{2d} + 1))[y]/(y^\delta + 1)$, where the inner structure is a ring that contains “made-up” roots of unity, so that an FFT approach becomes feasible, even though the ring $\mathbb{A}$ itself does not have any.

Our formalism of Section 2 provides the appropriate notions of modular arithmetic so that it is possible to convert this algorithm into one that works for group polynomials.

Let K be a power of 2. Let $P(x) \in \mathbb{G}_1[x]$ and $Q(x) \in \mathbb{G}_2[x]$ of degree less than K . We are interested in computing the product $(P \star Q)(x) \in \mathbb{G}_3[x]$ modulo $(x^K + 1)$. We will now decompose P and Q into about $\sqrt{K}$ pieces, each having about $\sqrt{K}$ coefficients. Let d be the equal to $\sqrt{K}$ if K is a square and $\sqrt{K}/2$ otherwise. Let $\delta = K/d$. Now, we introduce the ring $\mathbb{A} = \mathbb{F}_q[t]/(t^{2d} + 1)$, and the groups $\tilde{\mathbb{G}}_i = \mathbb{G}_i[t]/(t^{2d} + 1)$, which are $\mathbb{A}$ -modules. The polynomial $P(x) = \sum p_i x^i$ is mapped to a polynomial $\tilde{P}(y) = \sum \tilde{p}_i y^i \in \tilde{\mathbb{G}}_1[y]$ as follows:

$$\begin{aligned} \tilde{p}_0(t) &= p_0 + \dots + p_{d-1}t^{d-1} + 0_{\mathbb{G}_1}t^d + \dots + 0_{\mathbb{G}_1}t^{2d-1} \\ \tilde{p}_1(t) &= p_d + \dots + p_{2d-1}t^{d-1} + 0_{\mathbb{G}_1}t^d + \dots + 0_{\mathbb{G}_1}t^{2d-1} \\ &\dots \end{aligned}$$

In other words, if one lifts $\tilde{P}(t)$ to a bivariate polynomial in $(\mathbb{G}_1[t])[y]$, then the composition of $\tilde{P}$ with the monomial t^d gives back P . The polynomial Q is decomposed in the same way as $\tilde{Q}(y) \in \tilde{\mathbb{G}}_2[y]$.

Let $\omega = t^{4d/\delta} \in \mathbb{A}$ (i.e. $\omega = t^4$ when K is a square and $\omega = t^2$ otherwise). Then, by construction, $\omega^{\delta/2} = t^{2d} = -1$, so that ω is a primitive δ -th root of unity in $\mathbb{A}$. Applying Theorem 1 together with Remark 2, we get a multiplication algorithm that would give $(\tilde{P} \star \tilde{Q})(y) \bmod y^\delta - 1$. In order to get the operation modulo $y^\delta + 1$ (what is usually called a negacyclic convolution), it suffices to do a change of variable $y \leftarrow t^{2d/\delta}y$ before applying Theorem 1 and doing the inverse change thereafter. Indeed, $t^{2d/\delta}$ is a 2δ -th primitive root of unity in $\mathbb{A}$.

We have a polynomial $\widetilde{P \star Q}(y)$ in $\tilde{\mathbb{G}}_3[y]/(y^\delta + 1)$. To get back a polynomial in $\mathbb{G}_3[x]$, we first lift each coefficient in $\mathbb{G}_3[t]$, and get $\widetilde{P \star Q}$ in $\mathbb{G}_3[t][y]$. We can then compose this with x^d , and get $P \star Q$. Since we used $y = x^d$, the result that was modulo $y^\delta + 1$ is, in the end, modulo $x^K + 1$, as expected.

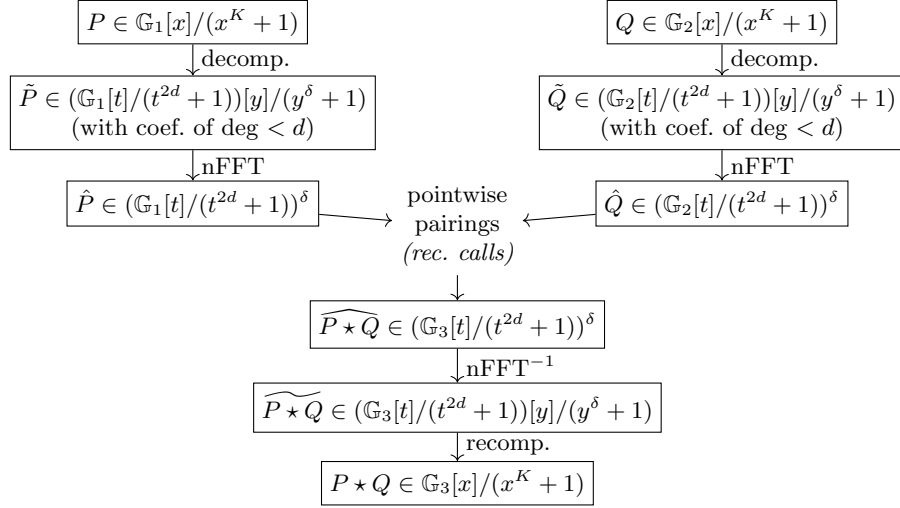


Fig. 1. Schönhage-Strassen's algorithm for multiplying group polynomials modulo $x^K + 1$, where $K = d\delta$ is a power of 2. The decomposition step collates coefficients of P (resp. Q) in δ coefficients of $\tilde{P}$, each of them grouping d of them. The recomposition is the inverse operation, i.e. evaluating at $y = t^d$. The notation nFFT stands for negacyclic Fourier transform, which is a modification of the FFT to compute modulo $x^n + 1$ instead of $x^n - 1$. Finally, the pointwise pairings are in the modular extensions of the $\mathbb{G}_i$ groups, and are done recursively.

One of the key ideas of this algorithm is that the back-forward mapping from $\tilde{\mathbb{G}}_i$ to $\mathbb{G}_i[t]$ does not pose problem because during the initial mapping, the $\tilde{p}_i(t)$'s and $\tilde{q}_i(t)$'s have their top coefficients equal to zero, and the result in $\widetilde{P \star Q}$ involves sums of products of those, and therefore no wrap-around modulo $t^{2d} + 1$ occurs.

Instead of providing pseudo-code for this, we find it more useful to provide the illustrative picture on Figure 1 where all the steps are summarized, together with our SageMath artifact proof-of-concept implementation (see Appendix), where the function `SchStrRec()` is very close to what we would write in pseudo-code.

Complexity analysis. The decomposition step does not involve any computation: this is just a different organization of the data, and putting copies of $0_{\mathbb{G}_1}$ and $0_{\mathbb{G}_2}$ in appropriate places. The recomposition involves adding each coefficients in $\mathbb{G}_3$ in the right place of the result; the cost is $2KM_3$ and is negligible.

The negacyclic Fourier transform (nFFT) of $\tilde{P}$ requires first multiplying the $\tilde{p}_i$ by powers of t for the change of variable. Then, according to Theorem 2, the cost of the Fourier transform is $\frac{1}{2}\delta \log \delta$ multiplications by powers of ω in $\tilde{\mathbb{G}}_1$, and $\delta \log \delta$ operations in $\tilde{\mathbb{G}}_1$. Since ω is a power of t , all the multiplications that have to be done during the negacyclic transform are by powers of t , and due to the form of the modulus, such a multiplication amounts to add or subtract

elements of $\mathbb{G}_1$. The complexity of computing the nFFT of $\tilde{P}$ and $\tilde{Q}$ is therefore $3d\delta \log \delta (A_1 + A_2)$.

The inverse negacyclic Fourier transform has the same cost, except that the coefficients must be divided by a power of 2. This gives $3d\delta \log \delta A_3 + 2d\delta M_3$.

The scheme is called recursively, and, just like in the case of Toom-Cook, the normalization step where we divide by a power of 2 does not need to be done at each recursive step: it is possible to do it at the very end, on the coefficients of the results, leading to an additional cost of $K M_3$.

For the main part of the algorithm, using $K = d\delta$, with $d \approx \delta \approx \sqrt{K}$, we get the following recursive equation for the complexity:

$$C(K) = \frac{3}{2} K \log K (A_1 + A_2 + A_3) + \sqrt{K} C(2\sqrt{K}).$$

This resolves in

$$C(K) = \frac{3}{2} K \log K \log \log K (A_1 + A_2 + A_3) + \alpha K \log K B,$$

where the constant α depends on the algorithm used as a base case, at the end of the recursive calls.

In order to compute the constant α , let us denote by $K_i = 2^{k_i}$ the largest length available with i recursive steps, initialized with $K_1 = 4$ and $k_1 = 2$. We want to analyze the number of calls $\tilde{C}_i$ to the base case multiplication in size K_i . The recursion formulae for these quantities are $k_{i+1} = 2k_i - 1$, and $\tilde{C}_{i+1} = K_i \tilde{C}_i$. This resolves into $k_i = 2^{i-1} + 1$, $K_i = 2 \cdot 2^{2^{i-1}}$, and

$$\tilde{C}_i = \tilde{C}_1 2^{i-1} 2^{2^{i-1}-1}.$$

This can be rewritten as $\tilde{C}_i = \tilde{C}_1 \frac{(2k_i-1)}{2} \frac{1}{4} K_i$. Hence, asymptotically we get

$$\tilde{C}_i = \frac{\tilde{C}_1}{4} K \log K.$$

Similarly, the worst case is obtained for the relation $k_{i+1} = 2k_i - 2$, which leads to a cost that is doubled.

- **With Karatsuba-4.** For $K_1 = 4$, with two steps of Karatsuba, we get a cost $\tilde{C}_1 = 9B$ (without counting negligible costs). The bilinear cost of Schönhage-Strassen becomes between $\frac{9}{4} K \log K B$ and $\frac{9}{2} K \log K B$.
- **With NTT-4.** In the case where $4|q-1$, one can use NTT as a base case, which leads to $\tilde{C}_1 = 4B + (M_1 + M_2 + M_3)$. Here, the costs of the multiplication by $\sqrt{-1}$ in the groups during the transforms are not negligible in the overall cost, and we get between $K \log K (B + \frac{1}{4}(M_1 + M_2 + M_3))$ and twice that value.

We obtain:

Proposition 11. *Let K be a power of 2. The multiplication of group polynomials of degree less than K modulo $x^K + 1$ by the Schönhage-Strassen algorithm has a cost $\frac{3}{2}K \log K \log \log K(A_1 + A_2 + A_3) + K M_3 + \alpha K \log K B$, plus negligible terms, where α is between $\frac{9}{4}$ and $\frac{9}{2}$.*

If, in addition, the group exponent q is such that $4|q - 1$, then, the cost can be reduced to $\frac{3}{2}K \log K \log \log K(A_1 + A_2 + A_3) + K M_3 + \alpha K \log K (B + \frac{1}{4}(M_1 + M_2 + M_3))$, plus negligible terms, where α is between 1 and 2.

In the implementation results of Section 5, we will see that with a better tuning of the threshold for switching to Karatsuba algorithm, the constant α can always be made to be around 2.

On the Cost of Normalization after Inverse Transform In any multiplication scheme based on DFT/NTT, the interpolation map of Theorem 1 is computed in two steps: first an inverse fast Fourier transform is computed, and second a normalization factor is applied to the output. Typically, all the coefficients of the resulting polynomial are divided by the length of the transform, which is usually a power of 2. In classical FFT-based algorithm over rings, multiplication or division by a power of 2 is cheap and negligible compared to other costs. In our case, though, dividing by 2 in $\mathbb{G}$ is implemented as a multiplication by the inverse of 2 in $\mathbb{F}_q$, which is a full-size integer modulo q .

In the case of Schönhage-Strassen, the M_3 component of the complexity comes from the normalization by powers of 2. By bilinearity of the polynomial multiplication, one can apply this normalization to either input polynomial instead of on the output polynomial, choosing the cheapest. One can therefore replace M_3 by $\min(M_1, M_2, M_3)$ in this part of the complexity formula.

The same reasoning applies to Toom-Cook where the normalization comes from the determinant of the Vandermonde matrix corresponding to the evaluation points.

3.7 Summary of the Costs

In Figure 2 we summarize the costs of the algorithms that we have described, choosing the Karatsuba-4 variant for Schönhage-Strassen, because it is always available. Some remarks are in order:

- The $\log \log K$ factor that makes the complexity of the Schönhage-Strassen algorithm hard to compare with other FTT-based algorithms becomes negligible in most cases. Indeed, one has to compare the costs $\log \log K A_i$ and M_i . And in most of the groups, M_i is two orders of magnitude larger than A_i , since M_i represents a double-and-add algorithm that calls A_i hundreds of times. In any foreseeable application, $\log \log K$ will be less than 5.
- In all cases where B has a cost at most comparable with one of the M_i , then the overhead of not having roots of unity is a small constant factor compared with the cost of the NTT algorithm if it were available.

Algorithm	B	M _i	A _i
Schoolbook	d^2	—	$(d-1)^2 A_3$
Karatsuba	$d^{1.59}$	—	$O(d^{1.59})(A_1 + A_2 + A_3)$
Toom-Cook- k	$d^{\log(2k-1)/\log k}$	$d \min(M_1, M_2, 2M_3)$	$O(d^{\log(2k-1)/\log k})(A_1 + A_2 + A_3)$
NTT in $\mathbb{F}_q$	K	$\frac{1}{2} K \log K (M_1 + M_2 + M_3)$	<i>negl.</i>
NTT in $\mathbb{F}_{q^2}$	$\frac{3}{2} K$	$\frac{3}{4} K \log K (M_1 + M_2 + M_3)$	<i>negl.</i>
Sch.-Str.	$\alpha K \log K, \alpha \leq \frac{9}{2}$	$K \min(M_1, M_2, M_3)$	$\frac{3}{2} K \log K \log \log K (A_1 + A_2 + A_3)$

Fig. 2. Summary of the asymptotic costs for the convolution of group polynomials $\mathbb{G}_1[x] \times \mathbb{G}_2[x] \rightarrow \mathbb{G}_3[x]$. For Schoolbook and Karatsuba, d is a bound on the degrees of the input polynomials. For the FFT-based algorithms, K is a power of 2, and the convolution is done modulo $x^K \pm 1$. The costs are expressed in terms of B (the cost of a pairing between elements of $\mathbb{G}_1$ and $\mathbb{G}_2$), M_i (the cost of a scalar multiplication in $\mathbb{G}_i$) and A_i (the cost of a group operation in $\mathbb{G}_i$). Negligible terms are not shown.

Algorithm	Schoolbook	Karatsuba	Toom-Cook- k	NTT in $\mathbb{F}_q$	NTT in $\mathbb{F}_{q^2}$	Schönhage-Strassen
Cost in M_{ell}	d^2	$d^{1.59}$	$d^{\log(2k-1)/\log k}$	$K \log K$	$\frac{3}{2} K \log K$	$\alpha K \log K$, with $\alpha \leq \frac{9}{2}$

Fig. 3. Summary of the asymptotic costs for the Elliptic canonical convolution pairing.

In Figure 3 we give the costs for the special case of Elliptic canonical convolution pairing, which covers the important application to batched KZG proofs. We recall that in that case, $B = M_1 = M_3 = M_{\text{ell}}$ is the cost of a scalar multiplication in the elliptic curve, while M_2 is a multiplication in $\mathbb{F}_q$ which is 2 orders of magnitude cheaper. Similarly all the A_i 's are negligible.

4 Applications to Cryptographic Algorithms

As mentioned in the introduction, we focus on two important applications, batched KZG proofs and the Bayer-Groth verifiable mixnet. While the first one involves the canonical convolution pairing, thus justifying that it has been a running example in the previous sections, the second one is a far less standard example. Indeed, the groups themselves depend on the number of ciphertexts to mix, and the pairing is unusual, leading to atypical relative costs of basic operations.

4.1 Batched KZG Proofs

The KZG protocol [15] is a widely-used polynomial commitment scheme. It works in a pairing-based cryptography setting. Let $\mathbb{G}$ be the (additive) first group of the domain of the pairing, with generator g of order q . During the setup, a group polynomial $S(x) = \sum_{0 \leq i < d} (s^{d-i} \cdot g)x^i$ is computed and published, where s is a (secret) element of $\mathbb{Z}_q$. Let $f(x) \in \mathbb{Z}_q[x]$ be a polynomial; in the KZG scheme, the commitment of f is obtained as the group element $C_f = f(s) \cdot g$, which can be computed from the public coefficients of S , without knowing s . It is then possible for the entity which knows f to prove that it verifies $f(y) = z$, for

public values y and z . For this, let us define $T_y(x) = (f(x) - z)/(x - y)$, which is a polynomial in $\mathbb{Z}_q[x]$. The proof is the commitment C_{T_y} . Checking the proof amounts to computing two pairings and verifying that they match. This basic KZG proof does not require fast arithmetic in the exponent.

However, it is possible to also define a multi-point proof, where the prover guarantees that $f(y_i) = z_i$ for a set of public values y_i and z_i . These batched KZG proofs are increasingly used in advanced protocols, like scalable threshold cryptosystems [22], bilinear accumulators [20], or memory pool privacy [7].

In [11], Feist and Khovratovich show that one can efficiently compute these KZG proofs, using asymptotically fast group polynomial arithmetic. They provide algorithms for both settings, either in the case where many individual proofs have to be computed for the same polynomial f , or in the case where a multi-point proof is needed.

The key idea of their algorithms is to reinterpret the construction of the KZG proofs with convolutions of group polynomials in $\mathbb{G}[x]$ and $\mathbb{F}_q[x]$. For instance, in [11, Proposition 1], we have to compute a group polynomial $h(x)$, and evaluate it at a multiset of points. And in [11, Proposition 2] the authors show that $h(x)$ can be interpreted as a product of other group polynomials, modulo $x^d - 1$.

Assuming that $\mathbb{F}_q$ contains appropriate roots of unity, they use the NTT to obtain a quasi-linear complexity.

With our work, we have FFT support, even when $\mathbb{F}_q$ is not NTT-friendly. The additional cost can be deduced from Figures 2 and 3. Here, we have $\mathbb{G}_1$ and $\mathbb{G}_3$ that are elliptic curve groups, and $\mathbb{G}_2 = \mathbb{F}_q$. Therefore, $\mathbf{B}$, $\mathbf{M}_1$ and $\mathbf{M}_3$ are scalar multiplications in the elliptic curves and the other costs are negligible. For NTT, the cost is $K \log K$, while the cost of Schönhage-Strassen is bounded by $\alpha K \log K$, where α is a constant. In our experiments that we describe in the next Section, we show that α can be made close to 2. Therefore, the absence of roots of unity only incurs an approximately twofold slowdown.

4.2 Bayer-Groth Verifiable Mixnets

In [3], Bayer and Groth proposed a verifiable mixnet protocol. It takes as input a list of ElGamal ciphertexts, and returns a list of re-encryptions of these ciphertexts, in a shuffled order, together with a zero-knowledge proof that the operation has been done honestly, without revealing the permutation. This building block is used in e-voting, as a way to cut the link between voters and their ballot, before decryption. It is used in the Swiss Post Voting System [21], which is deployed in several Swiss cantons for legally binding elections.

As described in the article, the complexity of the mixnet is quasi-linear in the number of ciphertexts. To reach this complexity, the prover has to perform asymptotically fast polynomial arithmetic in the exponent, and therefore, the quasi-linear complexity is available only for ciphertexts constructed on a group $\mathbb{G}$ of order q such that $q-1$ is divisible by a large power of 2. We remark, however, that the implementation of [21] does not include any NTT implementation, and therefore does not provide the quasi-linear complexity.

Contrary to the two previous applications, the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ have a more involved structure. Let $N = n^2$ be the number of ciphertexts, that is assumed to be a perfect square. We denote by $\mathbb{H} = \mathbb{G} \times \mathbb{G}$ the group of pairs of elements of $\mathbb{G}$, which is the domain of the ElGamal ciphertexts. The three groups are then $\mathbb{G}_1 = \mathbb{H}^n$, $\mathbb{G}_2 = \mathbb{F}_q^n$ (i.e. the n -th Cartesian power of the additive group of the finite field $\mathbb{F}_q$), and $\mathbb{G}_3 = \mathbb{H}$. The corresponding pairing (in the sense of Definition 2) combines n ciphertexts with n scalars into a single ciphertext as follows:

$$\begin{aligned} \star : \quad & \mathbb{H}^n \times \mathbb{F}_q^n \rightarrow \mathbb{H} \\ & ((c_1, \dots, c_n), (a_1, \dots, a_n)) \mapsto \sum a_i c_i \end{aligned}$$

(Note: we keep the additive notation to be consistent with the rest of the paper, but for ElGamal ciphertexts a more usual notation as used in [3] would be multiplicative; hence the image of the pairing would read $\prod c_i^{a_i}$.)

This pairing $\star$ is extended as a convolution pairing between the group polynomials $\mathbb{G}_1[x]$ and $\mathbb{G}_2[x]$, with image in $\mathbb{G}_3[x]$. Then, the most costly step performed by the prover can be re-interpreted as the convolution of two group polynomials of degree n . From Section 3, this can be done with a number of operations that is quasi-linear in n . Given the complex structures of $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_3$, this has to be reinterpreted in terms of scalar multiplications in $\mathbb{G}$.

For $\mathbb{G}_1 = \mathbb{H}^n$, the cost of one scalar multiplication is $2n$ scalar multiplications in $\mathbb{G}$. For $\mathbb{G}_2 = \mathbb{F}_q^n$, the scalar multiplication means multiplication in each of the n components, so this is negligible. For $\mathbb{G}_3 = \mathbb{H}$, the cost of a scalar multiplication is 2 scalar multiplications in $\mathbb{G}$. And finally, for the cost of the pairing, this is $2n$ scalar multiplications in $\mathbb{G}$. Hence, in total, depending on the availability of roots of unity we get the following total cost for the prover:

- NTT in $\mathbb{F}_q$: $N \log n$ scalar multiplications in $\mathbb{G}$;
- NTT in $\mathbb{F}_{q^2}$: $\frac{3}{2} N \log n$ scalar multiplications in $\mathbb{G}$;
- Schönhage-Strassen: $2\alpha N \log n$ scalar multiplications in $\mathbb{G}$.

The first case is just a restatement of the algorithm in [3]. We see that with our algorithms, even if the roots of unity are not present, we still have a quasi-linear complexity, and lose only a small constant factor close to 4.

If one wants to use for $\mathbb{G}$ an elliptic curve taken from the standards, before our work, the alternative approach would lead to a complexity of $N^{1.5}$ scalar multiplications in $\mathbb{G}$ for the prover. To mitigate this, in [3], it is proposed to use Toom-Cook polynomial multiplication, but without our framework, the authors had to modify the zero-knowledge proof and resort to additional interactions with the verifier at each recursion level.

5 Implementation and Experimental Validation

We have implemented all the algorithms of Section 3 (except the Toom-Cook algorithm) in a proof-of-concept artifact written in SageMath, whose main language is Python. Our goal is threefold:

- Validate the correctness of our algorithms, and therefore, the correctness of the algebraic framework we developed for the group polynomials.
- Validate experimentally the costs in operation counts that we claim.
- Provide a code base that should be easy to adapt to any other language and underlying library for the group and field arithmetic.

In particular, for the third point, we did not use any advanced feature of SageMath: just finite field and elliptic curve arithmetic.

Our implementation of the multiplication algorithms is generic and can work on any group structure $\mathbb{G}$ with the appropriate interface, giving in particular access to a ring $\mathbb{A}$ for which the group $\mathbb{G}$ is an $\mathbb{A}$ -module, and the corresponding action.

Due to the fully generic nature of this SageMath implementation, it allows us to test and validate essentially all our claims, except those concerning some part of the cost formulae that should be negligible. This is not the case in this implementation, and this is not so much of a surprise with a high level language.

We therefore wrote another implementation in Rust, in order to validate our approximations in the cost estimates. We chose to keep a moderate level of generics, but this is not as flexible as the SageMath one. Due to the strong typing, heavy use of generic types would require either to make the code too complex or to sacrifice compile-time awareness that would impact efficiency.

For our test, we used a laptop running Linux, with an Intel Core Ultra 7 255U processor and 32 GB of RAM. Our implementations use a single core; in this situation the frequency can in theory go up to 5.2 GHz. The version of SageMath is 10.8. For Rust, we used the nightly version; we rely on curve25519-dalek 4.1.3 and bls12_381 0.8.0 for the underlying curve arithmetic.

We make our implementations available as a free software artifact, so that all the data from this section are reproducible.

5.1 Tests of Correctness

In our SageMath implementation, we provide the following instances of groups: multiplicative groups and additive groups of prime finite fields, elliptic curves BLS12-381 (which allows NTT), T-256 of [10] (which allows NTT in $\mathbb{F}_{q^2}$), NIST-P256 and Curve25519 (which both allow only generic algorithms). In order to implement our NTT in $\mathbb{F}_{q^2}$ and Schönhage-Strassen algorithms, we also provide ways to construct the appropriate extensions of groups. For this, we make use of Proposition 3 to define the corresponding module action by the extended ring. We also defined the canonical pairings for these groups.

Finally, we have implemented the groups required in the Bayer-Groth algorithm, namely tuples of ElGamal ciphers and of finite field elements, together with the pairing they used.

Note that when we call Schönhage-Strassen or NTT in $\mathbb{F}_{q^2}$ in the Bayer-Groth setting, most of these abstractions are combined into fairly complex structures.

In order to validate the algorithms of Section 3, we compare the results obtained with various algorithms, making the assumption that the basic schoolbook

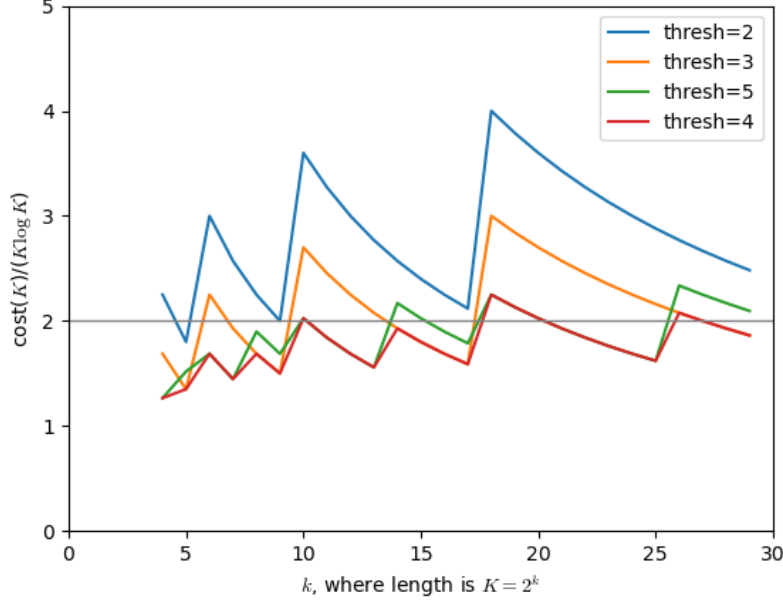


Fig. 4. Cost of Schönhage-Strassen algorithm as a function of the length of the input, for various values of the threshold with Karatsuba. In abscissa, k is the logarithm of the length $K = 2^k$, and in ordinate, the cost divided by $K \log K$.

implementation is correct. We performed such tests for groups of all the kinds listed above. They can be reproduced by running our SageMath artifact.

5.2 Checking the Operation Count

Still in the SageMath artifact, for each implementation of a group $\mathbb{G}$ that is an $\mathbb{A}$ -module, we have added a counter for the number of calls to the module actions. In the typical case where $\mathbb{G}$ is a multiplicative group of a finite field, this is modular exponentiation, while for an elliptic curve, this is scalar multiplication. The count does not depend on the group. We therefore run the algorithms on the computationally cheapest group, namely the additive group of a finite field.

For Schönhage-Strassen, however, the formula in Figure 2 is not precise regarding the constant in front of $K \log K B$, and our experiments allow to measure the quotient between the number of pairings and $K \log K$ for various sizes. In Figure 4 we plot this quantity. As claimed, with Karatsuba-4, the value stays below $\frac{9}{2}$. We experimented with several thresholds for the length at which the algorithm switches to Karatsuba during recursive calls. The optimal is obtained for a threshold of 4, i.e. that for a length of size less or equal to $2^4 = 16$, we call Karatsuba. With this parameter choice, the cost is $\approx 2K \log K B$ for all

Algorithm \ k		6	8	10
Schoolbook	theory	4096	65536	1048576
	measured	4096 (+0.00 %)	65536 (+0.00 %)	1048576 (+0.00 %)
Karatsuba	theory	729	6561	59049
	measured	729 (+0.00 %)	6561 (+0.00 %)	59049 (+0.00 %)
NTT	theory	384	2048	10240
	measured	322 (-16.15 %)	1794 (-12.40 %)	9218 (-9.98 %)
NTT in $\mathbb{F}_{q^2}$	theory	576	3072	15360
	measured	742 (+28.82 %)	3766 (+22.59 %)	18166 (+18.27 %)
SchStr	theory	768	4096	20480
	measured	712 (-8.29 %)	3712 (-9.38 %)	21760 (+6.25 %)

Algorithm \ k		12	14	16
Karatsuba	theory	531441	4782969	
	measured	531441 (+0.00 %)	4782969 (+0.00 %)	
NTT	theory	49152	229376	1048576
	measured	45058 (-8.33 %)	212994 (-8.14 %)	983042 (-6.25 %)
NTT in $\mathbb{F}_{q^2}$	theory	73728	344064	1572864
	measured	84982 (+15.26 %)	389110 (+13.09 %)	1753078 (+11.46 %)
SchStr	theory	98304	458752	2097152
	measured	87040 (-11.46 %)	458752 (+0.00 %)	1835008 (-12.50 %)

Fig. 5. Comparison between the asymptotic theoretical cost of Figure 3 of the elliptic canonical convolution pairing, in length 2^k , in number of scalar multiplications, and the cost measured in our implementation. For Schönhage-Strassen, we took $\alpha = 2$ in the asymptotic formula.

practical lengths. Note that for large sizes, we did not use a real computation, but a function that computes just the cost (this function has been checked to be correct on small to medium values).

In Figure 5, we report on how the theoretical asymptotic costs of the elliptic canonical convolution pairing in Figure 3 fit with the measured one (we measure the number of operations, here, not the running time). For Schoolbook and Karatsuba, measurement is equal to the theoretical cost. For NTT, the measured cost is slightly below the asymptotic estimate of Figure 5. This could look surprising, because the linear cost $K\mathbb{B}$ should add to the $K \log K$ contribution. However, at the first stage of the DFT, the roots of unity are ± 1 , and multiplying by them is for free. This is a saving in $O(K)$, which more than compensates the bilinear cost. When K grows, as expected, the difference with the theory becomes smaller. For NTT in $\mathbb{F}_{q^2}$, on the other hand, the linear contribution is much larger, so that the measured cost is higher than the one given by the asymptotic formula. Again, the difference diminishes for larger sizes. Finally, for Schönhage-Strassen, we recover the oscillating part of Figure 4.

Algorithm \ k		8	9	10	11	12	13	14	15	16	17
NTT	time (s)	0.48	0.95	2.15	5.20	10.61	22.85	50.00	107.40	232.42	492.18
	cost (M_{ell})	1794	4098	9218	20482	45058	98306	212994	458754	983042	2097154
	ratio (ms / M_{ell})	0.27	0.23	0.23	0.25	0.24	0.23	0.23	0.23	0.24	0.23
SchStr	time (s)	0.89	1.75	5.28	10.28	20.91	41.16	109.87	218.82	438.77	895.12
	cost (M_{ell})	3712	7424	21760	43520	87040	174080	458752	917504	1835008	3670016
	ratio (ms / M_{ell})	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24	0.24
ratio SchStr/NTT		1.86	1.84	2.45	1.98	1.97	1.80	2.20	2.04	1.89	1.82

Fig. 6. Benchmark of the Rust implementation, with the BLS12-381 group. For NTT and Schönhage-Strassen algorithms, we report the running time for each length 2^k , the number of scalar multiplications, and the ratio between both. We also give the ratio of the running times between both algorithms.

Algorithm \ k	4	5	6	7	8	9	10	11
Schoolbook (s)	0.17	1.34	10.70	84.58				
Karatsuba (s)	0.05	0.32	1.90	11.44	68.71	410.36		
SchStr (s)	0.05	0.29	1.67	6.78	36.34	146.04	880.63	3535.00

Fig. 7. Running times, in seconds (Rust implementation) of the main operation for the prover, in the Bayer-Groth verifiable mixnet, for various k , where the number of ciphertexts to mix is 2^{2k} .

5.3 Running Times Experiments

Our first focus is to validate the claim that, in the Schönhage-Strassen algorithm, the part of the complexity with the $\log \log K$ factor is negligible, so that the running times is proportional to $K \log K$. We mention that this is not the case with our SageMath implementation, which is arguably too high-level a language, so that even basic operations like copies have a non-negligible cost, when the structures are complex.

The Rust implementation is better suited for this analysis, and we report on Figure 6 our experiments, where we compare the measured running times of Schönhage-Strassen algorithm with the number of scalar multiplications (which grows approximately like $2K \log K$, as seen above). In the result, we see that the running times grow perfectly linearly with the number of scalar multiplications, within measurement precision. This provides strong evidence that the $\log \log K$ factor is negligible in practice.

The data also confirms the claim that Schönhage-Strassen takes a time that is about twice that of NTT (with some variations).

Finally we benchmarked the operation that is required in the Bayer-Groth mixnet, in the case where the base curve is the Curve25519, with the Ristretto group. This is a realistic setting, in the sense that for a highly regulated context as e-voting, it is often mandatory to use a curve from a standard, hence without NTT. According to Section 4.2, the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are $\mathbb{H}^n$ and $\mathbb{F}_q^n$, where $\mathbb{H}$ is the group of ElGamal ciphertexts and n and k must be such that $n = 2^k$ and we have n^2 ballots to mix. We therefore benchmark the cost of the multiplication of

polynomials in the exponent for the pairing of Bayer-Groth, for various values of k , and keeping $n = 2^k$. The size of the input is therefore in 2^{2k} , which explains why we can not go very far. The result of this experiment is shown in Figure 7 and confirms that Schönhage-Strassen provides a quasi-linear complexity in the number of ballots to mix. This also shows that mixing millions of ballots is entirely feasible, even if practical security reasons impose that it runs on an air-gapped laptop.

This benchmark also provided an opportunity to verify that the complex structures involved in the Schönhage-Strassen algorithm did not incur excessive memory overhead. The largest experiment, with $k = 11$ corresponding to 4 million ciphertexts, used about 8 GB of memory. While this is not negligible, we do not expect it to be a practical bottleneck.

6 Conclusion

In this work, we have shown how the most popular polynomial multiplication algorithms can be adapted to computations performed “in the exponent”, as required by some advanced cryptographic protocols. In particular, the Karatsuba algorithm for moderate input sizes, and the Schönhage-Strassen algorithm for asymptotically quasi-linear complexity are available provide practical alternatives to the NTT algorithm, when the groups in which the operations take place have a group order that is not NTT-friendly.

In this setting, the Schönhage-Strassen algorithm effectively avoids the practical impact of the additional $\log \log$ factor in the complexity, which is often seen as a major drawback, compared to NTT. Therefore, it proves to be quite competitive, and much simpler to deploy than, for instance, ECFFT. Our practical experiments demonstrate this good behaviour. In situations where imposing the presence of many roots of unity in the group would lead to increased costs in other places of the protocol, then it is perhaps preferable to use our technique while incurring only about a factor-two slowdown in the polynomial arithmetic in the exponent part.

Finally, we mention that the literature on discrete Fourier transform is rich, and it is a huge task to explore it thoroughly to check if some of the known techniques could lead to optimizations in our setting. However, we consider that beforehand, this would require refining the cost model and taking into account parallelism and the multi-scalar-multiplication (MSM) primitive. This opens new questions, since some elliptic curve libraries provide parallelism within MSM, while others don’t.

Artifacts

The artifact described in Section 5 is available under a free software license as a companion material to this article at <https://inria.hal.science/hal-05785708>.

This should allow to reproduce our results. We refer to the README file in this repository for software requirements and instructions on how to use them.

References

1. Aranha, D.F., El Housni, Y., Guillevic, A.: A survey of elliptic curves for proof systems. *Designs, Codes and Cryptography* **91**, 3333–3378 (2022), special Issue: Mathematics of Zero-Knowledge
2. Ballet, S., Pielant, J., Rambaud, M., Randriambololona, H., Rolland, R., Chaumine, J.: On the tensor rank of multiplication in finite extensions of finite fields and related issues in algebraic geometry. *Russian Mathematical Surveys* **76** (2021)
3. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: *EUROCRYPT 2012*. pp. 263–280. Springer (2012)
4. Ben-Sasson, E., Carmon, D., Kopparty, S., Levit, D.: Scalable and transparent proofs over all large fields, via elliptic curves - (ECFFT Part II). In: *TCC 2022*. pp. 467–496. LNCS, Springer (2022)
5. Ben-Sasson, E., Carmon, D., Kopparty, S., Levit, D.: Elliptic curve fast Fourier transform (ECFFT) Part I: Low-degree extension in time $o(n \log n)$ over all finite fields. In: *SODA 2023*. pp. 700–737. SIAM (2023)
6. Bernstein, D.J.: Multidigit multiplication for mathematicians (2001), preprint available from <https://cr.yp.to/papers/m3.pdf>
7. Choudhuri, A.R., Garg, S., Policharla, G., Wang, M.: Practical mempool privacy via one-time setup batched threshold encryption. In: *USENIX Security 2025*. pp. 3477–3495. USENIX (2025)
8. Chudnovsky, D.V., Chudnovsky, G.V.: Algebraic complexities and algebraic curves over finite fields. *J. Complex.* **4**, 285–316 (1988)
9. Couveignes, J.M., Lercier, R.: Elliptic butterflies (2026), <https://arxiv.org/abs/2510.27375>
10. Faz-Hernández, A., Ladd, W., Maram, D.: ZKAttest: Ring and group signatures for existing ECDSA keys. In: *Selected Areas in Cryptography*. Springer (2022)
11. Feist, D., Khovratovich, D.: Fast amortized KZG proofs. *Cryptology ePrint Archive*, Paper 2023/033 (2023), <https://eprint.iacr.org/2023/033>
12. von zur Gathen, J., Gerhard, J.: *Modern computer algebra*. Cambridge University Press (1999)
13. Ingonyama: ICICLE: GPU library for ZK acceleration (2025), <https://github.com/ingonyama-zk/icicle>, version 4.0.0
14. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata. *Soviet Physics Doklady* **7** (1962)
15. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: *ASIACRYPT 2010*. pp. 177–194. Springer (2010)
16. Kothapalli, A., Setty, S.: HyperNova: Recursive arguments for customizable constraint systems. In: *CRYPTO 2024*. pp. 345–379. Springer (2024)
17. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: *CRYPTO ’91*. pp. 129–140. Springer (1992)
18. Schönhage, A., Strassen, V.: Schnelle Multiplikation großer Zahlen. *Computing* **7**, 281–292 (1971)
19. Sorensen, H., Jones, D., Heideman, M., Burrus, C.: Real-valued fast fourier transform algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing* (1987)
20. Srinivasan, S., Karantaidou, I., Baldimtsi, F., Papamanthou, C.: Batching, aggregation, and zero-knowledge proofs in bilinear accumulators. In: *CCS 2022*. p. 2719–2733. ACM (2022)

21. Swiss Post: Cryptographic primitives of the Swiss Post Voting System (2025), version 1.5.1. Available at <https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives/-/blob/master/Crypto-Primitives-Specification.pdf>
22. Tomescu, A., Chen, R., Zheng, Y., Abraham, I., Pinkas, B., Gueta, G.G., Devadas, S.: Towards scalable threshold cryptosystems. In: IEEE S&P 2020. pp. 877–893 (2020)
23. Toom, A.: The complexity of a scheme of functional elements realizing the multiplication of integers. Doklady Akademii Nauk SSSR **3** (1963)
24. Woo, A.P.Y., Ozdemir, A., Sharp, C., Pornin, T., Grubbs, P.: Efficient proofs of possession for legacy signatures. In: IEEE S&P. pp. 3291–3308 (2025)