Delegating Elliptic-Curve Operations with Homomorphic Encryption

Carlos Aguilar-Melchor
ISAE-Supaéro, Toulouse, France
Email: carlos.aguilar@isae-supaero.fr

Jean-Christophe Deneuville
INSA-CVL, LIFO, Bourges, France
jean-christophe.deneuville@insa-cvl.fr

Philippe Gaborit
University of Limoges, Limoges, France
philippe.gaborit@unilim.fr

Tancrède Lepoint
SRI International, New York, NY, USA
Email: tancrede.lepoint@sri.com

Thomas Ricosset
Thales Communications & Security, Gennevilliers, France
and INP-ENSEEIHT, CNRS, IRIT, Toulouse, France
thomas.ricosset@thalesgroup.com

Abstract—**The landscape of Fully Homomorphic Encryption (FHE) has known great changes these past six years. As computational costs drop, libraries are developed and new applications become possible. Prototypes demonstrating private health diagnosis, signal processing, genome statistics and database queries spur hope on the practical deployment of FHE in the near future. However, in most of these applications, increasing the privacy, security or enabling new functionalities comes in pair with some significant computation and/or communication burden.**

In this work, we depart from the latter paradigm and demonstrate that FHE might be useful to increase the *practical performance* of elliptic-curve cryptography. More precisely, we show how to reduce the computational burden of computing elliptic-curve points from a generator through delegation. We show that using homomorphic encryption it is possible to reduce in practice computational costs even with respect to traditional, not based on homomorphic-encryption, delegation protocols. We demonstrate the feasibility of our protocols with proof-of-concept implementations using HElib (adapted to handle multiprecision plaintext space moduli).

I. Introduction

Regarded as cryptography's "Holy Grail", fully homomorphic encryption (FHE) enables computation of (almost) arbitrary functions on encrypted data [16, 8]. Secure delegation of computationally costly operations is one of the most intuitive applications of homomorphic encryption. Indeed, instead of computing a given function for some secret inputs the delegator can encrypt the inputs, send them to a delegatee that computes the function over the encrypted data using FHE, and finally the delegator can retrieve the result and decrypt it.

The usual approach to compute a function through FHE is to find a binary circuit with AND and XOR gates allowing to evaluate the function. As a XOR gate is equivalent to an addition modulo two and AND gate to a multiplication modulo two, if an FHE scheme allows to do enough plaintext additions and multiplications modulo two, it can be used to evaluate this circuit over bit-by-bit encrypted data.

In practice, the computational cost of such an approach, both for the delegator and the delegatee, grows rapidly with the multiplicative-depth of the FHE computation, which is equal to the AND-depth (i.e. the maximal amount of successive AND gates an input must go through) of the circuit. Therefore, it is important to obtain low AND-depth circuits for the functions that must be evaluated.

Another interesting class of circuits is *arithmetic circuits*. In the context of FHE, these circuits are defined as circuits in which gates are exclusively integer additions or multiplications modulo a given fixed modulus (no division, inversions, bit operations, floating-point representation, etc.). When the function to be evaluated is a modular arithmetic polynomial, we can reduce significantly the multiplicative depth by replacing the binary circuit with an arithmetic circuit. In this case, we can directly use an FHE's natural addition and multiplication to evaluate the circuit by setting its plaintext modulus as the circuit modulus.

Point Computation. The considered points on elliptic-curve cryptography belong to a cyclic group, and therefore every point can be obtained through a scalar multiplication with a given generator. This is generally the most costly operation that is required on a protocol based on elliptic-curve cryptography. Hence, as in previous works (see Section I-A), we focus on delegating this operation.

Computing a scalar multiplication can be reduced, through a classical double-and-add [23] algorithm, to computing additions of points. Point addition algorithms depend a lot on the coordinate system used. In this sense, the most popular are projective coordinates which allow to make scalar multiplication of elliptic-curve points with a shallow arithmetic circuit. Combining these two facts, we deduce that it is possible to compute a scalar multiplication in an elliptic-curve through shallow arithmetic circuits.

This work. Practical implementations of FHE are not well suited for large plaintext moduli which are essential in elliptic-curve cryptography. In order to evaluate the efficiency of homomorphic encryption in this setting we have modified some existing libraries (see Section I-A) and proposed some new homomorphic encryption implementations that are adapted to it. We expect those libraries, to be released with an open-source license, to be useful in other contexts too.

Our delegation protocol is generic and it is possible to delegate the computation of a scalar operation over generic groups. Thus it is also possible to use the delegation techniques described here over other groups such as RSA multiplicative groups, but even with optimizations such as RNS representation *and* Montgomery Reductions (such as [3]), our tests demonstrate that the computational cost is too high to consider practical benefits in a foreseeable future. Given this fact, and space constraints, we focus on the groups used on elliptic-curve cryptography. More precisely we will use as an example computations over P-256 [4] which grants 128 bits security for cryptographic protocols.

In order to provide an efficient delegation protocol in the elliptic-curve setting we have revisited the literature on elliptic-curve point additions. Usually these algorithms try to reduce the number of field operations done, whereas in our setting we want to have the lowest possible multiplicative depth. We show that a recent algorithm, by Renes *et al.* [32] gives an algorithm that fits perfectly the homomorphic encryption setting with only two levels of multiplications.

A straightforward delegation protocol of the double-and-add protocol would result in a circuit with a very large multiplicative depth. For example for P-256 we would have 512 point addition depth (to hide through 256 iterations whether we do an add or a double-and-add). Using the point addition described above this will result in a multiplicative depth of 1024 which is absolutely unreachable with current homomorphic encryption schemes.

We propose an improved protocol, which relies on pre-computation and windowing, leading to delegation protocols such that the arithmetic circuit to be evaluated is extremely shallow. The exact depth depends on the pre-computation memory / performance trade-off. For example for a client with 64 MBytes of pre-computed data the delegatee only needs to evaluate an arithmetic circuit with multiplicative depth 8 in the P-256 setting.

The main bottleneck for our delegation protocol is the amount of data that needs to be sent to the delegatee. FHE encryption has a low transmission rate and encrypting and sending large amounts of data is therefore the most costly operation for the client. Again for the example of P-256 with 64MBytes of pre-computed data, the client has to send 200Kbits for every scalar multiplication done. Thus even with a gigabit Ethernet connection he cannot delegate more than five thousand multiplications per second. From a computational point of view the protocol is realistic and requires a few milliseconds of computation from the server per scalar multiplication.

Note that if we reduce the computational constraints on the delegatee and reduce to the maximum the delegator costs we can use a trans-encryption technique [28]. With such an approach the client only needs to send 16Kbits per scalar multiplication and can delegate a much larger amount of operations.

The practicality of our technique is validated through proof-of-concept implementations of the delegator and delegatee. Our implementations are based on an adaptation of HElib [19, 20] so as to handle large plaintext spaces (HElib is restricted to single precision prime powers). The delegatee implementation has been deployed on an Amazon EC2 instance (`c4.8xlarge`) to allow reproducibility and verification of our results. Our implementations will be made fully available under GNU General Public License (GPL).

A. Related Work

HE practical implementation. The most popular library, HELib [19, 20] was developed by Halevi and Shoup. It relies on Shoup's Number Theory Library [35] (NTL) and implements a variant of the BGV [8] scheme of Brakerski, Gentry and Vaikuntanathan. It also features many optimizations proposed by Gentry, Halevi and Smart [17]. HElib restricts the modulus to $t = p^r$ with p and r being simple-precision integers. We modified this library to accept such moduli but without size restrictions on p.

Precomputations and delegated Computations. Many works in the early 90's proposed to speed up protocols either based on factoring [34, 9, 26, 14, 7] or discrete logs [7] by using precomputations. While we also use precomputations in our protocol (see Fig. 1 for an overview and 2 for more details), most of speed-up is achieved through the delegation of some computations to an external, untrusted source of computational power, using additional techniques such as windowing [23].

Additionally, several protocols to delegate computation have been proposed in the literature, especially regarding to the delegation of scalar operations [15, 27, 31, 22, 38, 13]. In [13], Chevalier *et al.* present various delegation protocols and a proof that the attained performance is optimal. However, the proof they give considers that the only operation that the delegatee can do is a set of scalar multiplications for scalars given in the clear. Using homomorphic encryption we can make the delegatee compute a scalar multiplication without knowing which scalar is being used and therefore we can go beyond their definition of optimality.

Prototypes using Homomorphic Encryption. Open-source libraries for homomorphic encryption are available online [25, 19, 20, 29], and were used as building blocks in several prototypes in the recent years, such as statistics computations [30], machine learning [18], signal processing [1], database queries [5, 11], private health diagnosis [6], genome statistics [24], and edit distance [12]. To the best of our knowledge, no prototype tackled the delegation of scalar multiplications in the elliptic curve setting (nor exponentiations in the RSA setting).

B. Outline

The paper is organized as follows. Sec. II introduces the notation used throughout the paper, together with some mathematical background. Sec. III introduces the general approach to scalar multiplication delegation, and describes our approach. Experimental results for our protocol are presented in Sec. IV, before concluding the paper in the last section.

II. Preliminaries

Let $\mathbb{Z}_q$ be the set of integers modulo q, $\mathbb{F}_q$ be the finite field of q elements, and denote $E(\mathbb{F}_q)$ the group of points $(x, y) \in \mathbb{F}_q^2$ belonging to an elliptic curve E and G an element of $E(\mathbb{F}_q)$ (which in practice will often be a generator). Sampling uniformly an element x from a set S is denoted $x \overset{\$}{\leftarrow} S$. If q is an integer, denote $\ell_{w,q}$ the length of its representation using a basis of w elements (*e.g.* $\ell_{2,q}$ is its bit length).

Homomorphic Encryption and HElib. An homomorphic encryption (HE) scheme [16] allows to publicly process encrypted data without knowing the secret key. In this section we recall the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme [8] — implemented in the software library HElib [19, 20] — that we use in our prototypes. This description is mostly taken from [17].

BGV. BGV is defined over polynomial rings of the form $R = \mathbb{Z}[x]/(\Phi_m(x))$ where m is a parameter and Φ_m the m-th cyclotomic polynomial. The plaintext space is usually the ring $R_p = R/pR$ for an integer p.

A plaintext polynomial $a(x) \in R_p$ is encrypted as a vector over $R_q = R/qR$, where q is an odd public modulus. More specifically, BGV contains a chain of moduli of decreasing size $q_0 > q_1 > \cdots > q_L$ and freshly encrypted ciphertexts are defined modulo q_0. During homomorphic evaluation, we keep switching to smaller moduli after each multiplication until we get ciphertexts modulo q_L, which cannot be multiplied anymore — L is therefore an upper bound on the multiplicative depth of the circuit we can compute.

Homomorphic Operations. The plaintext space of BGV are elements of R_p, and homomorphic additions (resp. multiplications) correspond to additions (resp. multiplications) over the ring R_p.[1]

Batching. Rather than encrypting elements of $R_p = \mathbb{Z}_p[x]/(\Phi_m(x))$, the BGV cryptosystem can be slightly modified to encrypt vectors of elements of $\mathbb{Z}_p$ [36, 37, 10] in an SIMD fashion: homomorphic operations implicitly perform the componentwise operations over the plaintext vectors (and rotations are easy to perform via the Frobenius endomorphism). Such a feature is called *batching* and essentially allows, for the cost of an homomorphic evaluation, to evaluate the function independently on several inputs [8]. Let us recall briefly the batching technique for BGV from [36, 37]. If the cyclotomic polynomial $\Phi_m(x)$ factors modulo the plaintext space p into a product of irreducible factors $\Phi_m(x) = \prod_{j=0}^{\ell-1} F_j(x) \pmod{p}$, then a plaintext polynomial $a(x) \in R_p$ can be viewed as encoding ℓ different small polynomials, $a_j = a \bmod F_j$, and each constant coefficient of the a_j can be set to an element of $\mathbb{Z}_p$. Unfortunately, not every tuple (p, m) yield an efficient batching — we will discuss how we selected (p, m) in Sec. IV.

[1]One can easily obtain a scheme with plaintext space $\mathbb{Z}_p$ by embedding $\mathbb{Z}_p$ into R_p via $a \in \mathbb{Z}_p \mapsto a \in R_p$. Hence homomorphic operations correspond to arithmetic operations over the ring $\mathbb{Z}_p$.

HElib. HElib [19, 20] is, as of today, a standard library for HE prototypes. HElib is a C++ library that implements the BGV scheme (with batching) for arbitrary plaintext space modulus p^r, with p and r in simple-precision. This software library uses NTL [35] and includes numerous optimizations described in [17, 19, 20]. HElib supports multi-threading and is distributed under the terms of the GNU GPL version 2.

III. Delegating Elliptic-Curve Point Computation

Recall that we want to delegate the computation of kG to the UC for a secret k and a public or secret G. The high-level idea of the protocol is that we will send k and G (actually a representation thereof) encrypted under an HE scheme $\mathcal{E}$ so that the Cloud can homomorphically compute kG without knowing k nor G. Then the client will be able to decrypt and obtain kG. Note that our protocol, described in this section, does not reveal G but this point can be publicly known by other means (e.g. if using a standard which specifies a generator that should be used in a protocol). Assume the client C is connected to an external source of computational power (*e.g.* the Cloud, a LAN server or a Cryptographic Coprocessor in a PCI-e card) that we denote UC (for Untrusted Cloud). In Sec. III, we give a high-level overview of our protocol; in the rest of the section, we detail each step of the protocol, and in Sec. III we give the full protocol for completeness.

Throughput Limitation. Potentially, the computational power of the UC could be as large as needed. In such a setting the bottleneck can be two-fold: (1) the client communication bandwidth and (2) the HE scheme encryption/decryption throughput and the client post-processing. In practice, both potential bottlenecks have a similar impact, and give an upper bound on the amount of encrypted data that can enter/exit the client.

If the UC is in a LAN, or even better in a PCI-e card, available bandwidths can be very high. In the PCI-e setting, bandwidth can theoretically reach 252.064 Gbit/s (PCI-e x32 v4.0 with 16-lane slot). However, the client will not be able to do homomorphic encryptions or decryptions at such speeds. In practice a throughput limitation around some Gigabits will always exist, and will be the main performance bottleneck.

High-Level Description. The high-level description of this delegation protocol is provided in Fig. 1. In a first step, some precomputation will be performed on G (see Sec. III — essentially it will compute a set of windowed scalar multiplications of G), then in Step 2), some of the precomputed values, useful to compute kG, are encrypted under the HE scheme $\mathcal{E}$ and sent to the UC. The UC will homomorphically compute kG in Step 3), and will send back the computation to the client (Step 4)). Finally, the client will decrypt the result.

In the rest of this section, we will specify in detail every step of this protocol, and namely: (i) what precomputations need to be performed on G (Sec. III), (ii) which server computation, elliptic curve representation, and point addition should be used (Sec. III).

Step 1): Initialization and Windowing. Each doubling or point addition to compute kG from k will increase the

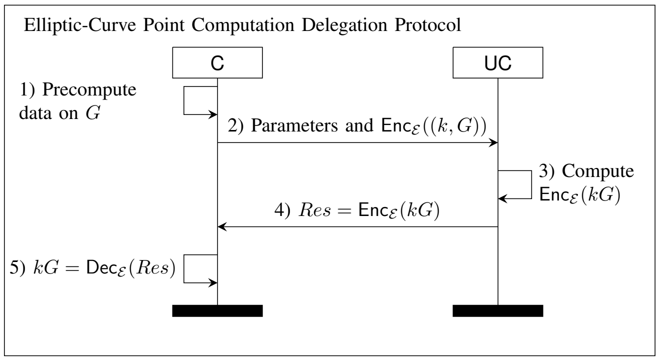

Elliptic-Curve Point Computation Delegation Protocol
C
UC
1) Precompute data on G
2) Parameters and $\mathsf{Enc}_{\mathcal{E}}((k, G))$
3) Compute $\mathsf{Enc}_{\mathcal{E}}(kG)$
4) $Res = \mathsf{Enc}_{\mathcal{E}}(kG)$
5) $kG = \mathsf{Dec}_{\mathcal{E}}(Res)$

Fig. 1: High-level description of the delegation protocol: delegation of the elliptic-curve point computation.

multiplicative depth of the UC computation. Therefore, we use a classical window technique to trade computation complexity for memory.[2] In Sec. III, we will detail how to represent the elliptic curve points to obtain an addition formula of multiplicative depth 2.

More precisely during Step 1), for a ω bits window size, the client precomputes all $2^\omega \cdot \ell_{2,n}/\omega$ points of the form $P_{i,j} = (i \cdot 2^{\omega(j-1)})G$ for i from 0 to $2^\omega - 1$, and for j from 1 to $\ell_{2,n}/\omega$ (we discuss the memory implications of this windowing technique in Sec. IV). Thus, for every scalar $k = \sum_{j=1}^{\ell_{2,n}/\omega} k^{(j)} \cdot 2^{\omega j}$, we have that

$$kG = \left(\sum_{j=1}^{\ell_{2,n}/\omega} k^{(j)} 2^{\omega j} \right) G = \sum_{j=1}^{\ell_{2,n}/\omega} P_{k^{(j)},j} \, ,$$

and computing the scalar multiplication only costs $\ell_{2,n}/\omega$ point additions.

The whole process is described in Fig. 2 for a single delegated elliptic-curve point computation. Note that the main cost for the client is to send the ordered set $(\mathsf{Enc}_{\mathcal{E}}(x_j), \mathsf{Enc}_{\mathcal{E}}(y_j))_j$ for $P_{k_j,j} = (x_j, y_j)$ and $j \in [0..\ell_{2,n}/\omega)$.

Using HE Batching. Instead of encrypting only one x_j (resp. y_j) per ciphertext, we will use HE batching to encrypt several (say m of them) x_j's in parallel for *different inputs scalars* k's (and the same j). Thus, for the same communication and UC computation complexities, we will be able to compute several kG's in parallel.

Steps 2) and 3): Revisiting Elliptic Curve Point Additions. In elliptic curve cryptography there are dozens of ways to perform additions and point doubling (see [21] for instance). Most algorithms were designed to minimize the number of multiplications on $\mathbb{F}_q$ one has to perform.[3] In practice the existing algorithms are not a priori optimized in terms of multiplicative circuit depth and we must revisit the existing work to get the best possible algorithm given this new optimization target.

[2]We leave as an interesting open problem a fine-grained analysis of the homomorphic evaluations of the best time-memory trade-offs for regular implementation of scalar multiplication over prime-field elliptic curves [33].

[3]Also they were designed to resist side-channel attacks by using the same "unified" formula for addition and point doubling, but we do not have to worry about such attacks because in homomorphic encryption, the operation flow is independent of the input scalar.

In the rest of the paper, we focus on elliptic curves with short Weierstraß equation of the form $E(\mathbb{F}_q) : y^2 = x^3 - 3x + b$, for $q \geq 5$ (the ones used in NIST standards). Recently Renes *et al.* proposed an efficient complete addition law that is valid not only for curves of composite order, but also for all prime order NIST curves [32], using standard projective coordinates. In such a representation, points $P(x,y) \in E(\mathbb{F}_q)$ can be written with triple (X, Y, Z) where $x = X/Z$ and $y = Y/Z$ [21]. Such coordinates were designed to speed up the point addition and doubling computation by avoiding field inversions for the benefit of multiplications.

Renes *et al.* presented an addition law [32, Algorithm 4] point addition circuits with a multiplicative depth of 2; we briefly review these formulae. Let $P_1(X_1, Y_1, Z_1)$ and $P_2(X_2, Y_2, Z_2)$ be points in the projective embedding of $E(\mathbb{F}_q)$, and denote by $P_3(X_3, Y_3, Z_3)$ their sum. Notice that there are no requirements on P_1 and P_2 being different, nor on being distinct from $\mathcal{O}(0, 1, 0)$.

For sake of clarity, we introduce some intermediate variables that can be computed with a multiplicative depth of one:

$$\begin{aligned}
T_0 &= (X_1Y_2 + X_2Y_1), \\
T_1 &= Y_1Y_2, \\
T_2 &= 3(X_1Z_2 + X_2Z_1 - bZ_1Z_2), \\
T_3 &= Y_1Z_2 + Y_2Z_1, \\
T_4 &= b(X_1Z_2 + X_2Z_1) - X_1X_2 - 3Z_1Z_2, \\
T_5 &= 3(X_1X_2 - Z_1Z_2).
\end{aligned}$$

Then the complete addition law on the projective embedding is given by the formula:

$$\begin{aligned}
X_3 &= T_0 \cdot (T_1 + T_2) - 3T_3 \cdot T_4, \\
Y_3 &= (T_1 - T_2)(T_1 + T_2) + 3T_5T_4, \\
Z_3 &= T_3(T_1 - T_2) + T_0T_5.
\end{aligned}$$

As mentioned in [32], the "plaintext" cost of this formula is $12\mathbf{M} + 2\mathbf{m}_b + 29\mathbf{a}$, where $\mathbf{M}$ (resp. $\mathbf{m}_b$ resp. $\mathbf{a}$) is the cost of a multiplication of two field elements (resp. multiplication by a constant, resp. addition). Therefore, the cost of evaluating this formula in the encrypted domain is 12 homomorphic multiplications, plus 2 multiplications by a constant, plus 29 homomorphic additions. The main advantage of this formula is that it can be evaluated as a depth 2 circuit, as shown in Fig. 3 (in the Appendix).

On a different but non negligible side, this addition law is also optimal in terms of communication complexity. Indeed, as it requires at least three coordinates to avoid field inversions — which are problematic for homomorphic computations — the formula of Renes *et al.* reaches the optimal lower bound. Moreover, the points we start our additions with are on standard coordinates which means that when putting them into projective coordinates, the third coordinate is always 1. We can therefore send just two coordinates per point.

For our proof-of-concept implementation of the protocol using a modified HElib BGV, [32, Algorithm 4] turned out to be the addition law yielding best performances. This can be explained by the relatively low number of multiplications required by this formula, in addition to the optimal depth and representation.

A reasonable question is whether one can reduce significantly the amount of operations by using more coordinates or increasing depth. The short answer proved to be no given the literature on elliptic curve point addition formulas[4]. Roughly, a deeper circuit can sometimes save one coordinate in the point representation, but the FHE parameters become too large (see [2] for more details). Vice-versa an additional coordinate either implies additional multiplications. Nevertheless, tradeoffs for constrained devices are possible and finding the best tuple (elliptic curve, set of coordinates, addition algorithm, homomorphic encryption scheme) for computing over encrypted elliptic curve points remains an interesting open question.

Full Protocol. For completeness, we provide in Fig. 2 the full description of the protocol. For the delegator C, the main effort is on Steps 6 and 7. They define his maximum capacity to delegate the point computations.

Security. As usual in delegation protocols we consider the delegatee honest but curious. In this setting the IND-CPA property of our scheme combined with a standard hybrid argument ensures the attacker learns nothing about the plaintexts sent to him nor about the result of the computation.

IV. Implementation Details

We implemented prototypes of our delegation protocol. The code was written in C++ using the open-source HElib library[5] [19, 20], which in turn uses NTL [35]. Our implementation will be made available under the GNU GPL license. We now describe implementation details.

The P-256 Curve. The commercially available ECC solutions usually implement NIST's P-256 Elliptic Curve. Since our goal is to demonstrate the feasibility of our delegation protocol, we chose to focus on the same curve. However, we would like to emphasize that this does not help us: the P-256 curve is quite bad for our setting! On the contrary, curves over binary fields, Koblitz curves or Edwards curves might yield much faster protocols given their small characteristic and their point addition formulae. The study of the fastest elliptic curve to be used in combination with homomorphic encryption is certainly an interesting theoretical open problem orthogonal to this work.

Recall that the P-256 curve is the elliptic curve

$$(E_{\textsf{P-256}})\colon y^2 = x^3 - 3x + b \in \mathbb{F}_{p_{\textsf{P-256}}}$$

where

$$\begin{aligned} p_{\textsf{P-256}} &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1, \text{ and} \\ b &= 41058363725152142129326129780047268409 1 \\ &\quad\; 14441015993725554835256314039467401291 \end{aligned}$$

Extension of HElib to Large p's. The HElib library only handles plaintext space modulus p^r with p and r in single precision. For our protocol however, since we considered the elliptic curve P-256, we have to work with plaintext space modulus $p = p_{\textsf{P-256}}$; we therefore modified HElib to handle that case.[6]

Selection of BGV Parameters. HElib automatically selects all the parameters for the BGV scheme from the tuple (m, p, L) where m is the index of the cyclotomic polynomial $\Phi_m(x)$, the plaintext space modulus p and L the multiplicative depth of the circuit to be homomorphically evaluated. As described in Sec. II, our protocol needs to perform batching, *i.e.* needs to embed several plaintext elements per ciphertext. Now, for every (m, p), the cyclotomic polynomial $\Phi_m(x)$ factors modulo p into a product of irreducible factors $\Phi_m(x) = \prod_{j=1}^{b} F_j(x) \pmod p$ for a $b = b(m, p) \in \{1, \ldots, \deg(\Phi_m(x)) = \phi(m)\}$ where ϕ is Euler's totient function. Unfortunately, not every tuple (p, m) yield an efficient batching; for example $p = p_{\textsf{P-256}}$ and $m = 1031$ do not allow batching at all — cf. Tab. I.

m	1031	1531	2048
$\phi(m)$	1030	1024	1024
b	1	1024	512

TABLE I: Degree $\phi(m)$ of the cyclotomic polynomial $\Phi_m(x)$, and number b of factors thereof modulo $p = p_{\textsf{P-256}}$.

In particular, full batching will be possible when $b = \phi(m)$, *i.e.* when all the roots of $\Phi_m(x)$ are in $\mathbb{Z}_p$. Since $\Phi_m(x)$ divides $x^m - 1$, all the roots of $\Phi_m(x)$ will be in $\mathbb{Z}_p$ when $m \mid p - 1$. Now we have that $p_{\textsf{P-256}} - 1 = 2 \cdot 3 \cdot 5^2 \cdot 17 \cdot 257 \cdot 641 \cdot 1531 \cdot 65537 \cdot 490463 \cdot 6700417 \cdot p'$ for $p' = 83594504224[...]3916927241$ a large prime. We select $m = 1531$ and by Tab. I, this yields a scheme that works with polynomials of degree $1024 - 1 = 1023$, and that can batch 1024 elements.

Windowing. In our delegation protocol, one can select different window sizes ω to encrypt the scalars k_i's. A larger window size will increase the memory used but will decrease the communication cost — cf. Tab. II. Also with larger windows, the UC will have to perform less homomorphic computations. The limiting factor is therefore the memory used by the client. Note that in the case the client is very limited in memory (e.g. a smartcard), this can be stored in unprotected memory with a MAC, or even outsourced (e.g. to the phone storage).

More precisely, the k_i's are decomposed in basis 2^ω, and give $2\lambda/\omega$ elliptic curve points represented by the two

[4]See `hyperelliptic.org/EFD/` for instance.

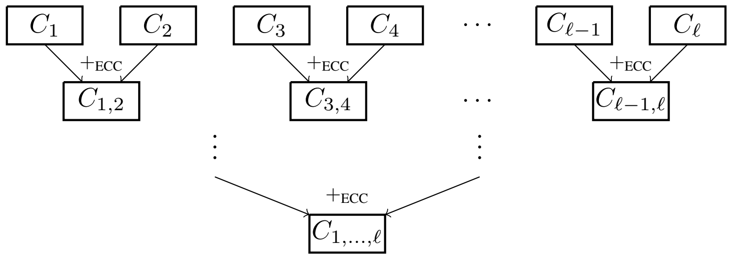
[5]`github.com/shaih/HElib`.

[6]A similar modification had been performed in `github.com/dwu4/fhe-si`, but was based on an earlier version of HElib without many of the recent improvements (special primes, etc.).

ECC point computation outsourcing protocol

Input: The elliptic-curve parameters, a value $\eta \in \mathbb{N}^*$, η scalars k_i, and a FHE scheme $\mathcal{E}$, a batching size b for the FHE scheme, a window size ω, and an ECC point representation.
Output: A set of η elliptic-curve points k_iG.

1) **Only once:** C computes all the $2^\omega \ell_{2,n}/\omega$ points $\{P_{i,j}\} = \{(i \cdot 2^{\omega(j-1)})\, G\}_{i=0..2^\omega-1}^{j=1..\ell_{2,n}/\omega}$. Noting $\ell = \ell_{2,n}/\omega$, for $k \in \{1,\ldots,n-1\}$ and $(k^{(1)},\ldots k^{(\ell)})$ the decomposition of k in base 2^ω we have $kG = P_{k^{(1)},1} + \ldots + P_{k^{(\ell)},\ell}$.
2) C draws $k_1,\ldots,k_b \xleftarrow{\$} \{1,\ldots,n-1\}$.
3) For each k_i define $(k_i^{(1)},\ldots,k_i^{(\ell)})$ as its decomposition in basis 2^ω.
4) For each $j \in \{1,\ldots,\ell\}$, C defines $MX_j = (\text{proj}_X(P_{k_1^{(j)},j}),\ldots,\text{proj}_X(P_{k_b^{(j)},j}))$.
5) For each $j \in \{1,\ldots,\ell\}$, C defines $MY_j = (\text{proj}_Y(P_{k_1^{(j)},j}),\ldots,\text{proj}_Y(P_{k_b^{(j)},j}))$.
6) C computes for each $j \in \{1,\ldots,\ell\}$, $CX_j = \text{Enc}_\mathcal{E}(MX_j)$ and $CY_j = \text{Enc}_\mathcal{E}(MY_j)$.
7) C sends $(CX_1, CY_1, \ldots, CX_\ell, CY_\ell)$ to UC.
8) UC generates $CZ_1 = \ldots = CZ_\ell = \text{Enc}_\mathcal{E}((1,\ldots,1))$, note $C_j = (CX_j, CY_j, CZ_j)$.
9) UC computes homomorphically $C = \textbf{HE.ECC.Add}(C_{j_1}, C_{j_2})$ taking two by two the vectors of encrypted coordinate sets and iterating recursively until there is only one result.



10) UC sends the encrypted result $C_{1,\ldots,l}$ to C.
11) C decrypts the result and for each $i \in \{1,\ldots b\}$ the coordinates X_i, Y_i, Z_i are used to define a point (x_i, y_i) with $x_i = X_i/Z_i$ and $y_i = Y_i/Z_i$.

Fig. 2: Full ECC point computation delegation protocol with batching

Parameters	$\lambda = 128$		
	$\omega = 8$	$\omega = 16$	$\omega = 32$
Enc. points sent	32	16	8
Com. cost (bits)	$64 \cdot \zeta_{\mathcal{E},q}$	$32 \cdot \zeta_{\mathcal{E},q}$	$16 \cdot \zeta_{\mathcal{E},q}$
Outsourced Mem. (Client)	512KB	64MB	2TB
Mult. depth	$5d_{add}$	$4d_{add}$	$3d_{add}$

TABLE II: Impact of Windowing over storage and communication costs. λ is the bit-security level, ω the bit-size of the window, $\zeta_{\mathcal{E},q}$ is the number of bits required to encrypt an element of $\mathbb{F}_q$, and d_{add} the multiplicative depth of the point addition circuit.

coordinates (x, y) in $\mathbb{F}_{p_{\text{P-256}}}$. Note that the decomposition is easy to compute: it simply consists of the $2\lambda/\omega$ successive sequences of ω bits of the k_i's binary decompositions. The precomputation costs are as follow:

- $(2\lambda)^2/\omega \cdot 2^\omega \cdot 2$ bits in (unprotected) memory storage for the client, and
- $2\lambda/\omega \cdot 2 \cdot \zeta_{\mathcal{E},p_{\text{P-256}}}$ bits in communication between the client and the UC, where $\zeta_{\mathcal{E},p_{\text{P-256}}}$ denotes the bits needed to encrypt a coordinate in $\mathbb{F}_{p_{\text{P-256}}}$ with the BGV scheme $\mathcal{E}$.

Moreover, the addition of the encrypted coordinates (which we denote $\text{Enc}_\mathcal{E}(x)$ and $\text{Enc}_\mathcal{E}(y)$) of the elliptic points received by the UC has a multiplicative depth linear in d_{add} and a number of multiplications that is closely related to the representation of those points (in our case, we have $d_{add} = 2$, cf. Sec. III). It is precisely this depth that determines the size of the BGV parameters $params_\mathcal{E}$ and hence, the size of data that is sent. Tab. II shows the impacts of different window sizes over communications and memory usage.

As the UC receives $\ell_{2,n}/\omega$ points to add, his addition circuit has depth $d = \log_2(\ell_{2,n}/\omega)$, which implies that our homomorphic encryption scheme $\mathcal{E}$ must be able to handle d times the depth d_{add} of an homomorphic elliptic curve point-addition.

Finally, we can reduce UC's computational cost of the point computation at the expense of an additional work factor for the client by allowing UC to abort the computation prematurely

Window Size	Mult. depth	# Elliptic Curve Additions	UC Time (s)	C → UC	UC → C	Client Outsourced Memory
				Com. Cost	Com. Cost	
8	2	16	24.8s	156.3 Mb	82.4 Mb	512 KB
8	4	$16+8$	61.2s	236.2 Mb	36.1 Mb	512 KB
8	6	$16+8+4$	102s	314.9 Mb	14.8 Mb	512 KB
8	8	$16+8+4+2$	148s	391.4 Mb	5.82 Mb	512 KB
8	10	$16+8+4+2+1$	208s	471.9 Mb	2.25 Mb	512 KB
16	2	8	12.4s	78.2 Mb	41.2 Mb	64 MB
16	4	$8+4$	30.1s	118.1 Mb	18.0 Mb	64 MB
16	6	$8+4+2$	51.6s	157.4 Mb	7.4 Mb	64 MB
16	8	$8+4+2+1$	74.4s	195.7 Mb	2.9 Mb	64 MB
32	2	4	6.2s	39.1 Mb	20.6 Mb	2 TB
32	4	$4+2$	15.4s	59.1 Mb	9.0 Mb	2 TB
32	6	$4+2+1$	25.9s	78.7 Mb	3.7 Mb	2 TB

TABLE III: Benchmarkings of the delegation of 1024 ECC point computations. Tests are run on a `c4.8xlarge` AWS instance for different window sizes and evaluations of different multiplicative depths — "UC Time (s)" denotes the benchmarking of the homomorphic evaluation consisting of "# Elliptic Curve Additions" additions of points over the curve P-256, we also include the communication and storage costs.

and let the client finish the job. This optimization is discussed in the next Section.

Early Aborts. In our delegation protocol, the UC compute homomorphically the coordinates of

$$kG = \left(\sum_{j=1}^{\ell_{2,n}/\omega} k^{(j)} 2^{\omega j} \right) G = \sum_{j=1}^{\ell_{2,n}/\omega} P_{k^{(j)},j} \,,$$

given the encryptions of the coordinates of the $P_{k^{(j)},j}$. In order to reduce the total depth of the point additions computation (namely $\log_2(\ell_{2,n}/\omega) \cdot d_{add}$ using a binary tree), the UC can perform the addition up to a specific multiplicative depth and send back the results (the coordinates of the partial sums) to the client. The latter will have to finish the point addition computation over the plaintexts. This in terms allows to lower the multiplicative depth capability of the BGV scheme at the expense of increasing the number of encrypted coordinates sent to the client. The resulting trade-off depends mainly on the ciphertext expansion. Notice that the more computation does the UC, the larger is the multiplicative depth and the ciphertext expansion, but the less ciphertexts are sent.

In our final evaluation, we will consider different early abortions trade-off — cf. Sec. IV.

Full Evaluation. Our prototype was deployed on a commercially available cloud computing service — namely a `c4.8xlarge` AWS instance, with turboboost turned off and running on a single thread, while the client was on a mid-range laptop. We benchmarked on different window sizes $\omega = 8, 16, 32$ and different early aborts (corresponding to resp. multiplicative depths $1 \cdot d_{add}$, $2 \cdot d_{add}$, $\ldots$, $5 \cdot d_{add}$).

We consider communications costs from the client to the UC (encrypted coordinates of the points to be added), and from the UC to the client (encrypted results). Of course we also consider computational costs for the client and UC. All the results are given in Tab. III. Note that the communication and computational costs are for a batch of 1024 point computations, so to obtain the costs per point the figures should be divided by 1024.

One of the most interesting compromises is a windowing size of 16 bits and full additions (no early abort) which can be executed in 74 seconds on a single thread and results in 200Mbits of uploaded data and 3Mbits of downloaded data. In order to use this window size the client also needs to access an internal or locally outsourced memory cache of 32MBytes.

Note that all the operations done by the UC are coordinate wise over vectors of 1024 coordinates given the encryption scheme parameters chosen. Therefore we can suppose that the computation can benefit from a linear gain in a multi-threaded environment with a large amount of threads. It would be possible to send the computation to two `c4.8xlarge` servers which can handle each 36 threads and thus get a reply in roughly a second.

For the client the main computational limit is the throughput at which it can produce the encrypted flow to be sent to the UC and decrypt the flow that comes from the UC. Our client could produce an encryption flow of 800Mbits/s and decrypt an incoming flow of 1.1Gbits/s. Such processing speeds would allow using 8 `c4.8xlarge` instances and retrieve thus up to $4 \cdot 1024$ encrypted points per second.

Note that if the UC is in the cloud and the client requires using it at its maximum throughput, having a constant upload bandwidth usage of 800Mbits/s can be quite costly. The cloud would be more adapted for a sporadic usage. If the processing power is required often, installing computing blades locally (through a LAN or PCI connection) would be a much more interesting strategy.

V. Conclusion and Perspectives

In this paper, we described a protocol that allows a constrained device processing ECC point computations to benefit from an outside source of computational power, in a secure way. It remains an open question to find new techniques to efficiently delegate RSA-like operations. Also, besides the optimizations proposed above, we believe it possible to further speed-up the constrained device using dedicated hardware, for elliptic curve point addition but our experiments do not cover that range.

References

[1] Aguilar-Melchor, C., Fau, S., Fontaine, C., Gogniat, G., and Sirdey, R. Recent advances in homomorphic encryption: A possible future for signal processing in the encrypted domain. *Signal Processing Magazine, IEEE 30*, 2 (2013), 108–117.

[2] Albrecht, M.R.: On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2017) 103–129

[3] Bajard, J.-C., and Imbert, L. A full rns implementation of rsa. *Computers, IEEE Transactions on 53*, 6 (2004), 769–774.

[4] Barker, E., Barker, W., Burr, W., Polk, W., and Smid, M. Nist special publication 800-57. *NIST Special Publication 800*, 57, 1–142.

[5] Boneh, D., Gentry, C., Halevi, S., Wang, F., and Wu, D. J. Private database queries using somewhat homomorphic encryption. In *ACNS 13* (June 2013), M. J. Jacobson Jr., M. E. Locasto, P. Mohassel, and R. Safavi-Naini, Eds., vol. 7954 of *LNCS*, Springer, Heidelberg, pp. 102–118.

[6] Bos, J. W., Lauter, K., and Naehrig, M. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics 50* (2014), 234–243.

[7] Boyko, V., Peinado, M., and Venkatesan, R. Speeding up discrete log and factoring based schemes via precomputations. In *EUROCRYPT'98* (May / June 1998), K. Nyberg, Ed., vol. 1403 of *LNCS*, Springer, Heidelberg, pp. 221–235.

[8] Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012* (Jan. 2012), S. Goldwasser, Ed., ACM, pp. 309–325.

[9] Brickell, E. F., Gordon, D. M., McCurley, K. S., and Wilson, D. B. Fast exponentiation with precomputation (extended abstract). In *EUROCRYPT'92* (May 1993), R. A. Rueppel, Ed., vol. 658 of *LNCS*, Springer, Heidelberg, pp. 200–207.

[10] Cheon, J. H., Coron, J.-S., Kim, J., Lee, M. S., Lepoint, T., Tibouchi, M., and Yun, A. Batch fully homomorphic encryption over the integers. In *EUROCRYPT 2013* (May 2013), T. Johansson and P. Q. Nguyen, Eds., vol. 7881 of *LNCS*, Springer, Heidelberg, pp. 315–335.

[11] Cheon, J. H., Kim, M., and Kim, M. Search-and-compute on encrypted data. In *FC 2015 Workshops* (Jan. 2015), M. Brenner, N. Christin, B. Johnson, and K. Rohloff, Eds., vol. 8976 of *LNCS*, Springer, Heidelberg, pp. 142–159.

[12] Cheon, J. H., Kim, M., and Lauter, K. E. Homomorphic computation of edit distance. In *FC 2015 Workshops* (Jan. 2015), M. Brenner, N. Christin, B. Johnson, and K. Rohloff, Eds., vol. 8976 of *LNCS*, Springer, Heidelberg, pp. 194–212.

[13] Chevalier, C., Laguillaumie, F., and Vergnaud, D. *Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions*. Springer International Publishing, Cham, 2016, pp. 261–278.

[14] de Rooij, P. Efficient exponentiation using procomputation and vector addition chains. In *EUROCRYPT'94* (May 1995), A. D. Santis, Ed., vol. 950 of *LNCS*, Springer, Heidelberg, pp. 389–399.

[15] Feigenbaum, J. Encrypting problem instances: Or ..., can you take advantage of someone without having to trust him? In *CRYPTO'85* (Aug. 1986), H. C. Williams, Ed., vol. 218 of *LNCS*, Springer, Heidelberg, pp. 477–488.

[16] Gentry, C. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC* (May / June 2009), M. Mitzenmacher, Ed., ACM Press, pp. 169–178.

[17] Gentry, C., Halevi, S., and Smart, N. P. Homomorphic evaluation of the AES circuit. In *CRYPTO 2012* (Aug. 2012), R. Safavi-Naini and R. Canetti, Eds., vol. 7417 of *LNCS*, Springer, Heidelberg, pp. 850–867.

[18] Graepel, T., Lauter, K., and Naehrig, M. ML confidential: Machine learning on encrypted data. In *ICISC 12* (Nov. 2013), T. Kwon, M. Lee, and D. Kwon, Eds., vol. 7839 of *LNCS*, Springer, Heidelberg, pp. 1–21.

[19] Halevi, S., and Shoup, V. Algorithms in HElib. In *CRYPTO 2014, Part I* (Aug. 2014), J. A. Garay and R. Gennaro, Eds., vol. 8616 of *LNCS*, Springer, Heidelberg, pp. 554–571.

[20] Halevi, S., and Shoup, V. Bootstrapping for HElib. In *EUROCRYPT 2015, Part I* (Apr. 2015), E. Oswald and M. Fischlin, Eds., vol. 9056 of *LNCS*, Springer, Heidelberg, pp. 641–670.

[21] Hankerson, D., Menezes, A. J., and Vanstone, S. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[22] Hohenberger, S., and Lysyanskaya, A. How to securely outsource cryptographic computations. In *TCC 2005* (Feb. 2005), J. Kilian, Ed., vol. 3378 of *LNCS*, Springer, Heidelberg, pp. 264–282.

[23] Knuth, D. The art of computer programming: Semi-numerical algorithms, volume vol. 2, 1997.

[24] Lauter, K. E., López-Alt, A., and Naehrig, M. Private computation on encrypted genomic data. In *LATINCRYPT 2014* (Sept. 2015), D. F. Aranha and A. Menezes, Eds., vol. 8895 of *LNCS*, Springer, Heidelberg, pp. 3–27.

[25] Lepoint, T., and Naehrig, M. A comparison of the homomorphic encryption schemes FV and YASHE. In *AFRICACRYPT 14* (May 2014), D. Pointcheval and D. Vergnaud, Eds., vol. 8469 of *LNCS*, Springer, Heidelberg, pp. 318–335.

[26] Lim, C. H., and Lee, P. J. More flexible exponentiation with precomputation. In *CRYPTO'94* (Aug. 1994), Y. Desmedt, Ed., vol. 839 of *LNCS*, Springer, Heidelberg, pp. 95–107.

[27] Matsumoto, T., Kato, K., and Imai, H. Speeding up secret computations with insecure auxiliary devices. In *CRYPTO'88* (Aug. 1990), S. Goldwasser, Ed., vol. 403 of *LNCS*, Springer, Heidelberg, pp. 497–506.

[28] Méaux, P., Journault, A., Standaert, F.-X., and Carlet, C. Towards stream ciphers for efficient fhe with low-noise ciphertexts. In *Proceedings of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9665* (New York, NY, USA, 2016), Springer-Verlag New York, Inc., pp. 311–343.

[29] Melchor, C. A., Barrier, J., Guelton, S., Guinet, A., Killijian, M.-O., and Lepoint, T. NFLlib: NTT-based fast lattice library. In *CT-RSA 2016* (Feb. / Mar. 2016), K. Sako, Ed., vol. 9610 of *LNCS*, Springer, Heidelberg, pp. 341–356.

[30] Naehrig, M., Lauter, K. E., and Vaikuntanathan, V. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011* (2011), pp. 113–124.

[31] Nguyen, P. Q., and Shparlinski, I. On the insecurity of a server-aided RSA protocol. In *ASIACRYPT 2001* (Dec. 2001), C. Boyd, Ed., vol. 2248 of *LNCS*, Springer, Heidelberg, pp. 21–35.

[32] Renes, J., Costello, C., and Batina, L. Complete addition formulas for prime order elliptic curves. Cryptology ePrint Archive, Report 2015/1060, 2015.

[33] Rivain, M. Fast and regular algorithms for scalar multiplication over elliptic curves. Cryptology ePrint Archive, Report 2011/338, 2011.

[34] Schnorr, C.-P. Efficient signature generation by smart cards. *Journal of Cryptology 4*, 3 (1991), 161–174.

[35] Shoup, V. NTL: A library for doing number theory, 2015. Version 9.4.0.

[36] Smart, N. P., and Vercauteren, F. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *PKC 2010* (May 2010), P. Q. Nguyen and D. Pointcheval, Eds., vol. 6056 of *LNCS*, Springer, Heidelberg, pp. 420–443.

[37] Smart, N. P., and Vercauteren, F. Fully homomorphic SIMD operations. *Des. Codes Cryptography 71*, 1 (2014), 57–81.

[38] Van Dijk, M., Clarke, D., Gassend, B., Suh, G. E., and Devadas, S. Speeding up exponentiation using an untrusted computational resource. *Designs, Codes and Cryptography 39*, 2 (2006), 253–273.

Appendix

Fig. 3 below shows that the complete addition formula of [32, Algorithm 4] has multiplicative depth 2.

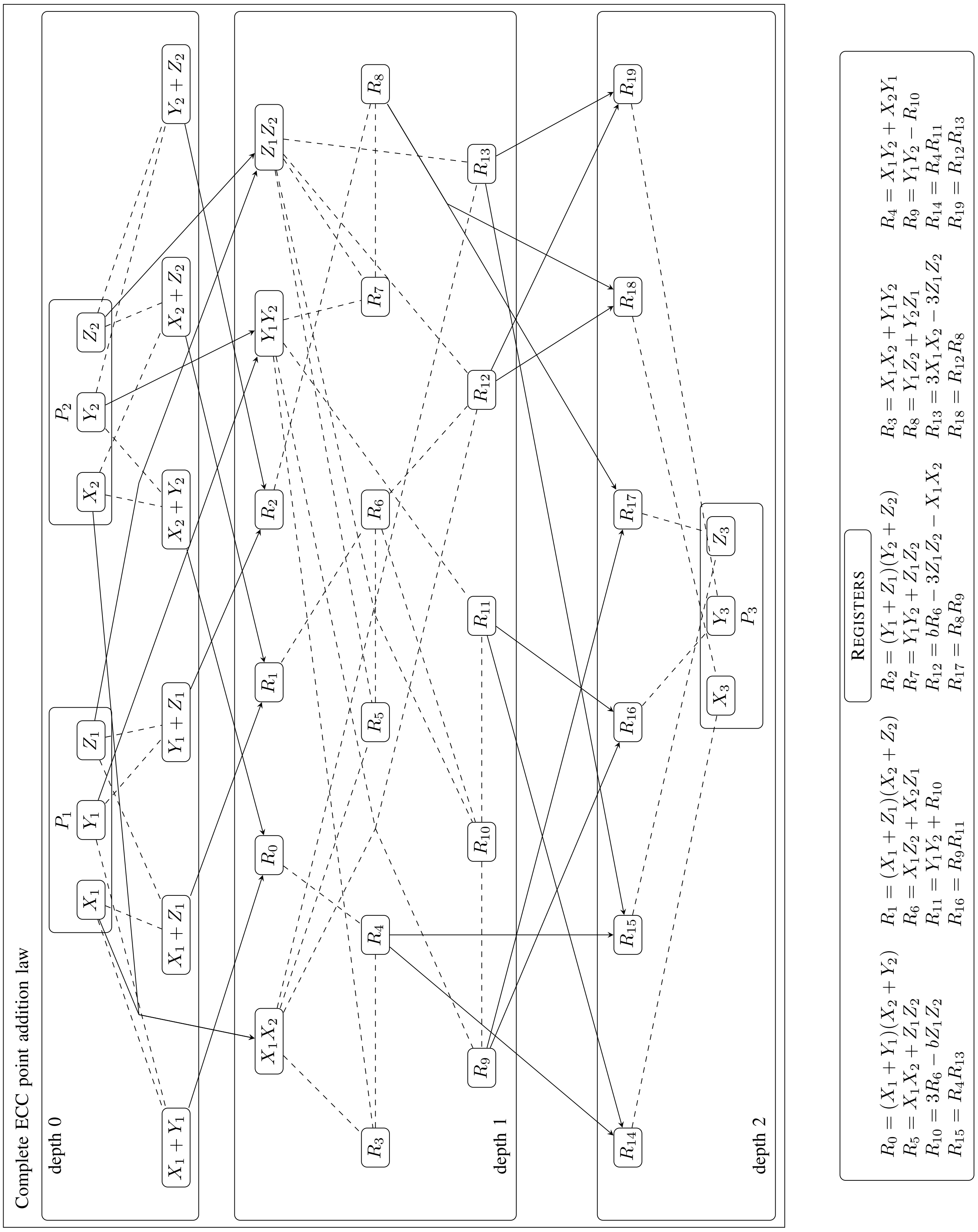


Fig. 3: Complete addition using [32, Algorithm 4]: only 3 coordinates, multiplicative depth 2, and 12 multiplications (see original algorithm for this). Dashed lines correspond to additions, filled ones to multiplications. We note this algorithm **ECC.Add** and its homomorphic encryption version **HE.ECC.Add**.