I want to ride my BICYCL: BICYCL Implements CryptographY in CLass groups

Cyril Bouvier[1], Guilhem Castagnos[2], Laurent Imbert[1], and Fabien Laguillaumie[1]

[1] Université de Montpellier, CNRS, LIRMM, Montpellier, France.
`{cyril.bouvier,laurent.imbert,fabien.laguillaumie}@lirmm.fr`
[2] Université de Bordeaux, CNRS, INRIA, IMB, UMR 5251, F-33400 Talence, France.
`guilhem.castagnos@math.u-bordeaux.fr`

Abstract. We introduce `BICYCL` an Open Source C++ library that implements arithmetic in the ideal class groups of imaginary quadratic fields, together with a set of cryptographic primitives based on class groups. It is available at `https://gite.lirmm.fr/crypto/bicycl` under GNU General Public License version 3 or any later version. `BICYCL` provides significant speed-ups on the implementation of the arithmetic of class groups. Concerning cryptographic applications, `BICYCL` is orders of magnitude faster than any previous pilot implementation of the `CL` linearly encryption scheme, making it faster than Paillier's encryption scheme at any security level. Linearly homomorphic encryption is the core of many multi-party computation protocols, sometimes involving a huge number of encryptions and homomorphic evaluations: class group-based protocols become the best solution in terms of bandwidth and computational efficiency to rely upon.

Keywords: class group cryptography, quadratic form arithmetic, implementation library, linearly homomorphic encryption, multi-party computation

1 Introduction

Class group cryptography was introduced in the late 80s by Buchmann and Williams [10,11] and concurrently by McCurley [45] through variants of the Diffie-Hellman key agreement [29] and their extensions to public-key cryptosystems using constructions similar to that of Elgamal [31]. More specifically, Buchmann and Williams [10] suggested that the class group of ideals of maximal orders of imaginary quadratic fields may offer a better security than the multiplicative group of finite fields. Indeed, at the time, the best algorithms for computing discrete logarithms in class groups were exponential in the group order, whereas index calculus methods with subexponential complexity already existed for the equivalent problem in (multiplicative subgroups of) finite fields. The current best algorithms for computing the structure of the class group of a quadratic imaginary number field [39,5,41] are all improvements of Hafner and

McCurley's algorithm [34]. The costs of these algorithms increase with Δ, the discriminant of the number field. The current largest computation involved a 512 bits discriminant [4].

With a subexponential complexity of $L_{1/2}(\Delta)$, computing class groups is still asymptotically slower than factoring integers, whose complexity is $L_{1/3}(N)$, where N is the integer to be factored. From a cryptographic perspective, it essentially means that any given security level may be achieved with shorter keys than those required by schemes based on integer factorization. Nevertheless, the arithmetic of class groups is quite intricate. Therefore, the associated cryptographic protocols are potentially slower than cryptosystems based on elliptic curves or those involving arithmetic modulo a prime number or an RSA modulus.

A nice attempt to improve the efficiency of cryptosystem based on class group was achieved by Hühnlein, Jacobson, Paulus and Takagi [37]. Unfortunately, a critical attack [18,17] against this family of cryptosystems consigned class group cryptography to oblivion. Over the last decade, class groups have regained a lot of interest thanks to their usefulness in designing advanced cryptosystems and secure multi-party computation protocols. In particular, ideas from the original attack [18] were positively turned into the Castagnos-Laguillaumie (CL) linearly homomorphic encryption scheme (LHE) [19] which involves a subgroup of a class group where discrete logarithms are easy to compute. CL is at the heart of a construction of projective hash functions, which in turn allows to design efficient inner-product functional encryption [20], two-party and fully-threshold ECDSA signatures [14,15,56,28], as well as coin mixing protocols [32].

A crucial advantage of class group cryptography is that it is well-suited when multi-party protocols require a one-time transparent (or public-coin) setup with minimal interaction among parties. In addition, it is also interesting for time-released cryptography since protocols can take advantage of an exponentiation over a group of unknown order. For instance, [53] presented secure timed commitments and a scalable distributed randomness generation with enhanced security, transparent setup, that relies on a variant of the CL encryption scheme. Besides, class group cryptography have recently led to many advanced protocols such as verifiable random functions without trusted setup [55], accumulators [44,7], encryption switching protocols [16], efficient designated-verifier non-interactive zero-knowledge proofs of knowledge [22], succinct non-interactive argument of knowledge [12,43], homomorphic secret sharing and pseudorandom correlation functions for generating oblivious transfer [1], range proofs [25,24] or vector commitments [3].

When available, the timings reported in these contributions come from vanilla implementations of the protocols, mainly relying on PARI/GP [48] functions for the arithmetic of class group. Very few dedicated libraries exist for class group cryptography: we can cite ZenGo-X's `Class` library written in Rust [57] (used to benchmark threshold ECDSA signatures), which makes calls to PARI/GP, Sayles' `libqform` library [49], which is an optimized C library for ideal arithmetic in imaginary quadratic number fields but which does not implement any cryptographic protocols, and the implementation that won the Chia VDF competition [23] whose goal was to evaluate a VDF by performing successive squarings as fast as possible[3].

In this paper, we introduce `BICYCL`, an optimized open source library that implements arithmetic in ideal class groups of imaginary quadratic fields together with class group-based cryptographic primitives and protocols. We show that our library outperforms any implementation that we were aware of, and demonstrate that class group cryptography can be more efficient than classical cryptography in certain cases, particularly for linearly homomorphic encryption. This is interesting for applications to multi-party computation as LHEs based on class groups offer at the same time: transparent setup, low bandwidth, efficient thresholdization due to their Elgamal structure, and now computational efficiency.

Our main contributions:

– From a theoretical point of view, we unify several frameworks to construct linearly homomorphic encryption schemes from [19,21], as well as compact (*i.e.,* with shorter ciphertexts) and large message variants of these cryptosystems and provide the security proofs.
– We optimize the implementation of the arithmetic of binary quadratic forms at a low level, namely the squaring (NUDUPL) and the composition (NUCOMP) of quadratic forms, by carefully implementing partial extended GCD variants. At a higher level, we also improve exponentiation in the class group arising from practical cryptographic applications, such as CL encryption which requires two fixed-basis exponentiations.
– We explicit some isomorphisms involving some subgroups of the class groups (in particular those where computing discrete logarithms is easily) that allow very fast exponentiations and discrete logarithm computations needed by the CL framework. More precisely, these explicit isomorphisms make it possible to avoid quadratic form arithmetic in these subgroups, allowing instead the use of more efficient arithmetic of quadratic integers.
– We provide a suite of cryptographic protocols: not only we have implemented the CL encryption and its variants, but we also provide an implementation of Paillier and Camenisch-Shoup, that take benefit, when possible, from our improvements for CL, as well as a zero-knowledge proof of well-formedness of a CL ciphertext.

`BICYCL` is orders of magnitude faster than any previous pilot implementation of the CL protocol. At the lower level, these significant speed-ups come from our implementation of the arithmetic of binary quadratic forms, namely the NUDUPL and NUCOMP algorithms. As an illustration, for 6000-bit discriminant, our version of NUDUPL is 15% faster than `libqform` [49], which was the fastest implementation available that we were aware of, and 68% faster than PARI/GP that is probably the most widely used. The cryptographic schemes that we have implemented also benefit from tailored exponentiations in the class

[3] `https://github.com/Chia-Network/vdftrack1results`

group, in particular when the base if fixed. In Section 6, we present more extensive comparisons with other libraries for the low level arithmetic primitives, as well as timings and comparisons with various LHE schemes. In particular, we show that for common applications, CL's encryption is faster than Paillier's at *any* security level. In addition, in multi-party computation protocols or when data are encrypted, homomorphic arithmetic modulo an RSA number is not well-suited if the message space is of order a prime q, and requires subtleties to deal with potential wraparounds or to hide the number of modular reductions modulo q during the process. On top of that, to reach adaptive security, range proofs must be added. CL provides a solution which is simpler and more efficient.

Our library will be augmented with other flavors of CL (like the DDH-f based variants, threshold decryption or Gaussian distributions for randomness sampling), as well as more complex cryptographic protocols that rely on class groups. It has been widely tested and can be used to implement class group cryptography, and more generally for computations in ideal class groups of imaginary binary quadratic fields.

Detailed Roadmap and Results: In Section 2, we give some background on class groups of imaginary quadratic fields. In Section 3 we present an unified view of the CL framework for designing linearly homomorphic encryption schemes from groups of unknown order that contain a subgroup where the discrete logarithm problem is easy. We provide generic schemes and their security, in particular a generic compact variant that was previously only described in a particular case. In Section 4, we review instantiations of the CL framework within class groups with some improvements compared to previous works. In particular, we simplify the handling of large messages and apply the compact variant to more cases as implemented in our library. We also give a guide to select the best variant of CL if one wants a linearly encryption scheme modulo an integer M of a certain bitsize without any particular property on the form of M. Then, we give specific theorems that explicit the representation of elements of the subgroups where discrete logarithms are easy, giving the mathematical foundations of some optimizations of our implementation. In Section 5, we focus on specific points of the implementation. First, the speed up of the arithmetic of class groups, then we show how to combine some known exponentiation methods from the context of elliptic curves in order to obtain the highest speed-ups for the instantiations of CL in class groups. Finally, we give algorithms specific to the subgroup where discrete logarithms are easy, in particular we give important improvements when the order of this group is a power of a prime, and more generally when the order is large by showing that we can work with quadratic integers instead of quadratic forms. In the last section, Section 6, we give benchmarks of the implementation of several linearly homomorphic encryption schemes: two from the CL framework for class groups, homomorphic modulo a prime q and 2^k, and the Paillier and Camenisch-Shoup encryption, homomorphic modulo an RSA integer N. These benchmarks highlight the improvements previously mentioned.

2 Class Group and Cryptography

We give here a high level overview of class groups of imaginary quadratic fields. See for instance [9,26] for a comprehensive treatment.

2.1 Quadratic Fields, Orders and Class Groups

Imaginary quadratic fields are extensions of degree 2 of $\mathbf{Q}$ that can be written as $K = \mathbf{Q}(\sqrt{D})$ where $D < 0$ is a square free integer. To such a field is attached an integer, called *fundamental discriminant* and denoted Δ_K, that is defined as $\Delta_K := D$ if $D \equiv 1 \pmod 4$ and $\Delta_K := 4D$ otherwise.

The ring $\mathcal{O}_{\Delta_K}$ of algebraic integers in K, also called the *maximal order*, can be written as $\mathbf{Z}[\omega_K]$ where $\omega_K = (\Delta_K + \sqrt{\Delta_K})/2$. One can define special subrings of $\mathcal{O}_{\Delta_K}$, called *orders*, associated to a non fundamental discriminant $\Delta_\ell := \ell^2 \Delta_K$ where ℓ is called a *conductor*. The order $\mathcal{O}_{\Delta_\ell}$ of conductor ℓ is the ring $\mathbf{Z}[\ell\omega_K]$, it has index ℓ in $\mathcal{O}_{\Delta_K}$.

To each discriminant Δ (fundamental or not), one can associate a finite abelian group, called the *ideal class group* and denoted $\mathrm{Cl}(\Delta)$ defined as the quotient of the group of (invertible fractional) ideals of $\mathcal{O}_\Delta$ by the subgroup of principal ideals. The order of this group is called the *class number* and denoted $h(\Delta)$.

One has $h(\Delta) \approx \sqrt{|\Delta|}$ in general and one can compute its number of bits in polynomial time using the analytic class number formula (cf [45]). However, the full value of $h(\Delta)$ is only known to be computable from Δ in sub-exponential time $L_{1/2}(\Delta)$. As a result, these groups have been extensively used in the past decade to implement cryptographic protocols based on groups of unknown order, as they can be generated with a public coin setup: It is sufficient to generate the integer Δ to use these groups. This is in contrast to the group of invertible elements of $\mathbf{Z}/N\mathbf{Z}$ for which one needs a trusted setup to generate an RSA integer N, in order to keep its factorization secret, and thus the order of the group unknown. Another feature used by cryptographic applications is that discrete logarithms in $\mathrm{Cl}(\Delta)$ are also hard to compute (again only sub-exponential time algorithms with complexity $L_{1/2}(\Delta)$ are known, see [6] for instance).

2.2 Elements and Group Law

Elements of $\mathrm{Cl}(\Delta)$ can be represented in a compact form. One can define a system of representatives of the classes with the notion of reduced ideals. Ideals are of the form $a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\mathbf{Z}$, where $a, b \in \mathbf{N}$ and a and b are smaller than $\sqrt{|\Delta|}$ when the ideal is reduced. As a result, one needs $\log_2(|\Delta|)$ bits to represent an element of the class group and this can be reduced to $3/4 \log_2(|\Delta|)$ using a recent technique proposed in [30]. Combined with the fact that the best known attacks against cryptographic problems in class groups are of complexity $L_{1/2}(\Delta)$ instead of $L_{1/3}(N)$ for factoring based schemes, this leads to bandwidth improvement for cryptographic applications.

The group law corresponds to product of ideals followed by a reduction operation to find the unique reduced ideal of the class. In practice, the group law is computed using the language of positive definite binary quadratic forms. Let a, b, c be three relatively prime integers such that $a > 0$ and $\Delta = b^2 - 4ac$, we will denote $f := (a, b, c)$ the primitive positive definite binary quadratic form over the integers, $f(X, Y) = aX^2 + bXY + cY^2$ of discriminant Δ. One can define an equivalence relation on forms from the action of $SL_2(\mathbf{Z})$. The class group $\mathrm{Cl}(\Delta)$ is actually isomorphic to the set of forms modulo this equivalence relation. Moreover there is an explicit correspondence between ideals and forms: the class of the form (a, b, c) corresponds to the ideal $a\mathbf{Z} + \frac{-b+\sqrt{\Delta}}{2}\mathbf{Z}$. The definition of the unique representative of the class is more natural when working with forms: it is the reduced form (a, b, c), which satisfies $-a < b \leqslant a$, $a \leqslant c$ and if $a = c$ then $b \geqslant 0$.

The group law is implemented using Gauss' composition of forms and a reduction algorithm for forms devised by Lagrange. More efficient algorithms have been proposed by Shanks to compute the group law, NUCOMP and NUDUPL, where a partial reduction if applied during composition ([50]). Our library represents elements of the class group $\mathrm{Cl}(\Delta)$ as triplets of integers corresponding to reduced binary quadratic forms and implements Shank's algorithms (see Section 5).

2.3 Maps between Class Groups

We have seen that from a fundamental (square free) discriminant Δ_K, one can define a class group $\mathrm{Cl}(\Delta_K)$ associated with the ring of integers $\mathcal{O}_{\Delta_K}$. Moreover, from Δ_K one can define a non fundamental discriminant $\Delta_\ell := \ell^2 \Delta_K$ and a class group $\mathrm{Cl}(\Delta_\ell)$ associated with the suborder $\mathcal{O}_{\Delta_\ell} \subset \mathcal{O}_{\Delta_K}$. This inclusion of the orders actually translates into a relation on the class groups $\mathrm{Cl}(\Delta_f)$ and $\mathrm{Cl}(\Delta_K)$. First one defines a map on the level of the ideals. A nonzero ideal I of $\mathcal{O}_{\Delta_\ell}$ is said to be prime to ℓ if $I + f\mathcal{O}_{\Delta_\ell} = \mathcal{O}_{\Delta_\ell}$. Then the map $I \mapsto I\mathcal{O}_{\Delta_K}$ is an isomorphism from the group of ideals of $\mathcal{O}_{\Delta_\ell}$ prime to ℓ to the group of ideals of $\mathcal{O}_{\Delta_K}$ prime to ℓ. This map and the inverse map are explicit and can be computed efficiently knowing the conductor ℓ (see [37, Algo. 2,3]). By passing to the quotient, we obtain a surjection $\varphi_\ell : \mathrm{Cl}(\Delta_\ell) \twoheadrightarrow \mathrm{Cl}(\Delta_K)$. Studying the kernel of this map, one obtains, for $\Delta_K < -4$, ([26, Theorem 7.24]) the formula

$$h(\mathcal{O}_{\Delta_\ell}) = h(\mathcal{O}_{\Delta_K}) \cdot \ell \cdot \prod_{p \mid \ell} \left(1 - \left(\frac{\Delta_K}{p}\right)\frac{1}{p}\right).$$

The instantiation of the CL framework to design linearly homomorphic encryption schemes in class groups heavily relies on this formula and on the fact that the discrete logarithm problem can be easy in $\ker \varphi_\ell$ for appropriate choices of Δ_K and ℓ. We will see this in more details in Section 4 with a precise characterization of the elements of this kernel.

2.4 Squares and Square Roots

A last important property of class groups for cryptography concerns squares and square root computations. Knowing the factorization of the discriminant Δ, it is possible to determine in polynomial time if an element of $\mathrm{Cl}(\Delta)$ is a square and compute its square roots ([42]). The situation is somewhat similar to $\mathbf{Z}/N\mathbf{Z}$ for an RSA integer $N = pq$ where one can compute the values of the Legendre symbol relative to p and q to determine if an element is a square. For class groups, genus theory defines various characters that play the role of these symbols, depending on the specific form of the discriminant. Genus theory is of crucial importance in the instantiation of the CL framework modulo 2^k. We refer the interested reader to [21, Subsection 2.3] for more details.

3 The CL Framework

In this section, we present the generic framework introduced in [19] and refined in [54] to design linearly homomorphic encryption schemes.

Let M be an integer. In the CL framework, a setup generates a large cyclic group of unknown order that contains a subgroup of order M where the discrete logarithm problem is easy. The index of that subgroup in the larger group is dictated by the security level. This construction makes it possible to instantiate various linearly homomorphic variants of the Elgamal cryptosystem where the plaintexts space is the additive group $(\mathbf{Z}/M\mathbf{Z}, +)$.

Compared to previous works which restricts the generic presentation to prime order groups for the plaintexts space, we expose at the same time the CL framework, the design of a generic linearly homomorphic encryption scheme and its security, with minimal hypotheses on the integer M.

This allows to capture many more concrete instantiations based on class groups: for instance the $\mathsf{CL\text{-}HSM_q}$ scheme introduced in [20], homomorphic modulo an odd prime q, but also variants modulo q^k as analysed in [27], and the recent scheme of [21] homomorphic modulo 2^k. As we shall see, this generalized framework is not limited to class groups. It also encompasses derivatives of the Paillier cryptosystem such as the Camenisch and Shoup's encryption scheme, homomorphic modulo an RSA integer N (see Remark 1 below).

Compare to the original exposition of the framework of [19] based solely on the DDH assumption, we take advantage of the HSM assumption from [20] which provides more efficient and versatile schemes (notably, these schemes are related to hash proof systems as exposed in [54]). Moreover, in the last subsection, we generalized to any message space, the so-called compact variant introduced in [19] and analyzed in [53] that produces more compact ciphertexts.

3.1 Generic definition and properties

Let us now introduce the algebraic structure and the properties required to build a CL scheme.

Definition 1 **($\mathsf{Setup}_{\mathsf{CL}}$ – Adapted from Def. 3.1 of [54]).** *Let λ be a security parameter and M be a positive integer. Let $\widehat{G}$ a group of unknown order $M \cdot \widehat{s}$, for some $\widehat{s}$ that depends on λ, and which admits a cyclic subgroup $F = \langle f \rangle$ of order M. We require $\gcd(M, \widehat{s}) = 1$, although the index $[\widehat{G} : F] = \widehat{s}$ should be hard to compute. Yet, we need to know an upper bound $\tilde{s} \geq \widehat{s}$. Let $h = x^M$ for some random element $x \in \widehat{G}$.*

A setup for the CL framework for M is a pair of algorithms $(\mathsf{Gen}_{\mathsf{CL}}, \mathsf{Solve}_{\mathsf{DL}})$ such that:

- $\mathsf{Gen}_{\mathsf{CL}}$ *takes as input 1^λ and outputs public parameters $\mathsf{pp} := (\tilde{s}, f, h, \mathsf{extra})$ where extra encodes necessary information to efficiently compute in $\widehat{G}$,*
- $\mathsf{Solve}_{\mathsf{DL}}$ *is a deterministic polynomial time algorithm for computing discrete logarithms in F.*

The following proposition clarifies the structure of the group used for cryptographic purposes.

Proposition 1. *Let $g = f \cdot h$, and let G be the cyclic subgroup of $\widehat{G}$ generated by g. Then, denoting $H = \{x^M ; x \in G\}$, we have $\langle h \rangle = H$ and $G \simeq F \times H$.*

Proof. Let $y \in H$. We have $y = (g^\alpha)^M = f^{\alpha M} \cdot h^{\alpha M} = h^{\alpha M}$ since f has order M. Thus $H \subset \langle h \rangle$. Conversely, denoting s the order of h, $\beta :\equiv M^{-1} \pmod{s}$ and $x := g^\beta$, one as $x^M = g^{\beta M} = h^{\beta M} = h$, so $h \in H$ and $\langle h \rangle = H$. Finally, since the orders of f and h are coprime, we have $G \simeq F \times H$. $\square$

The encryption schemes that we present in the following sections exploit the cyclic subgroup G by hiding with a random element of H the plaintext messages encoded in F, using a technique *à la* Elgamal. However, contrary to the standard Elgamal scheme defined in a group of known prime order, in the instantiations of our framework the order of G will remain unknown (to anyone), and there will be no way to efficiently recognize valid encodings of elements of this cyclic group. This can be problematic for applications involving malicious participants (like CCA secure encryption schemes, zero knowledge proofs and multi-party computation). However, in general, elements of the group $\widehat{G}$ will be easy to recognize, which will be sufficient for applications.

3.2 Assumption

The generic encryption scheme that follows the framework from Def. 1 relies on the hard subgroup membership assumption related to the direct product of Proposition 1. We next give the definition of this generic assumption for any integer M. Note that this generalizes the HSM assumption from [20], which was made for $M = q$, an odd prime (we take $u \xleftarrow{\$} (\mathbf{Z}/q\mathbf{Z})^\times$ instead of $u \xleftarrow{\$} (\mathbf{Z}/q\mathbf{Z})$ in [20], but it infers a negligible difference $1/q$ as the bitsize of q is larger than the security parameter to ensures that q is prime to $\hat{s}$ in concrete instantiations). This generic assumption also generalises the HSM_{2^k} assumption from [21] made for $M = 2^k$. This is also a generalisation of the DCR assumption (see Remark 1 below).

Notation 1 (Distribution $\mathcal{D}_H$) *In the following, given a generator h of a cyclic group H, we will denote $\mathcal{D}_H$ a distribution over the integers such that the distribution $\{h^x,\ x \hookleftarrow \mathcal{D}_H\}$ is at distance less than $\delta_{\mathcal{D}_H}(\lambda)$ from the uniform distribution in H for some negligible function $\delta_{\mathcal{D}_H}$.*

Knowing an upper bound $\tilde{s}$ on the order of H, this distribution can be instantiated efficiently from a folded uniform distribution or from a folded discrete Gaussian distribution (see [54, Subsection 2.7.1]).

Definition 2 ($\mathsf{HSM}_{\mathsf{CL,M}}$ assumption). *Let λ be a security parameter, M be a positive integer and $\mathsf{Setup}_{\mathsf{CL}} = (\mathsf{Gen}_{\mathsf{CL}}, \mathsf{Solve}_{\mathsf{DL}})$ be a setup for the CL framework for M. Let $\mathcal{A}$ be a PPT algorithm, its advantage for the $\mathsf{HSM}_{\mathsf{CL,M}}$ problem is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}_{\mathsf{CL,M}}}(\lambda) := \Big| \Pr\big[b = b^* \ : \ \mathsf{pp} = (\tilde{s}, f, h, \mathsf{extra}) \leftarrow \mathsf{Gen}_{\mathsf{CL}}(1^\lambda),$$
$$x \hookleftarrow \mathcal{D}_H, u \xleftarrow{\$} (\mathbf{Z}/M\mathbf{Z})^\times, b \xleftarrow{\$} \{0,1\}, z_0 := f^u h^x, z_1 := h^x,$$
$$b^* \leftarrow \mathcal{A}(\mathsf{pp}, z_b, \mathsf{Solve}_{\mathsf{DL}}(\cdot))\big] - 1/2 \Big|$$

For a positive integer M, The $\mathsf{HSM}_{\mathsf{CL,M}}$ problem is $\delta_{\mathsf{HSM}_{\mathsf{CL,M}}}-$hard for $\mathsf{Setup}_{\mathsf{CL}}$ if for all PPT algorithm $\mathcal{A}$, its advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{HSM}_{\mathsf{CL,M}}}(\lambda) \leqslant \delta_{\mathsf{HSM}_{\mathsf{CL,M}}}(\lambda)$. The $\mathsf{HSM}_{\mathsf{CL,M}}$ assumption holds for $\mathsf{Setup}_{\mathsf{CL}}$ if the $\mathsf{HSM}_{\mathsf{CL,M}}$ problem for $\mathsf{Setup}_{\mathsf{CL}}$ is $\delta_{\mathsf{HSM}_{\mathsf{CL,M}}}-$hard and $\delta_{\mathsf{HSM}_{\mathsf{CL,M}}}$ is negligible.

3.3 LHE under $\mathsf{HSM}_{\mathsf{CL,M}}$

We describe in Figure 1 the generic linearly homomorphic encryption scheme modulo M that follows from the CL framework, and prove its security in Theorem 2.

Theorem 2. *Let $\mathsf{Setup}_{\mathsf{CL}} = (\mathsf{Gen}_{\mathsf{CL}}, \mathsf{Solve}_{\mathsf{DL}})$ be a setup for the CL framework for an integer M. The resulting linearly homomorphic encryption scheme modulo M of Figure 1 is $\mathsf{ind-cpa}$ under the $\mathsf{HSM}_{\mathsf{CL,M}}$ assumption for $\mathsf{Setup}_{\mathsf{CL}}$.*

Proof. Let $\mathcal{A}$ be an adversary against the indistinguishability of the scheme. We describe a sequence of indistinguishable games which starts from the $\mathsf{ind\text{-}cpa}$ experiment $\mathsf{Game}\ 0$ and finishes with a game where the ciphertext statistically hides the challenge bit $b^\star$. In the following, we denote by S_i the event "adversary $\mathcal{A}$ outputs $b = b^\star$ in $\mathsf{Game}\ i$".

In the first game $\mathsf{Game}\ 1$, the only modification is the distribution of secret key sk: It is sampled according to the distribution $\mathcal{D}_G$ instead of $\mathcal{D}_H$.

The difference from $\mathcal{A}$'s view is therefore the distribution of $\mathsf{pk} = h^{\mathsf{sk}}$. With the notation of Proposition 1, $\mathcal{D}_G$ is a distribution over the integers such that the distribution $\{x \mod Ms : x \leftarrow \mathcal{D}_G\}$ is $\delta_{\mathcal{D}_G}-$close to the uniform distribution in $\{1, \dots, Ms\}$, where s is the order of H. Since s divides Ms, sampling sk according

```
Algorithm 1: KeyGen
    Input: $\mathsf{pp} = (\tilde{s}, f, h, \mathsf{extra})$
           generated by $\mathsf{Gen}_{\mathsf{CL}}$
    Result: $(\mathsf{pk}, \mathsf{sk})$
    —
1   sample $\mathsf{sk} \xleftarrow{\$} \mathcal{D}_H$
2   $\mathsf{pk} := h^{\mathsf{sk}}$
3   return $(\mathsf{pk}, \mathsf{sk})$
```

```
Algorithm 2: Encrypt
    Input: $\mathsf{pp}, \mathsf{pk}, m \in \mathbf{Z}/M\mathbf{Z}$
    Result: ciphertext $(c_1, c_2)$
    —
1   sample $r \xleftarrow{\$} \mathcal{D}_H$
2   $c_1 := h^r$
3   $c_2 := f^m \mathsf{pk}^r$
4   return $(c_1, c_2)$
```

```
Algorithm 3: Decrypt
    Input: $\mathsf{pp}, \mathsf{sk}, (c_1, c_2)$
    Result: $m \in \mathbf{Z}/M\mathbf{Z}$
    —
1   $a := c_2 \cdot c_1^{-\mathsf{sk}}$
2   return $\mathsf{Solve}_{\mathsf{DL}}(\mathsf{pp}, a)$
```

```
Algorithm 4: EvalAdd
    Input: $\mathsf{pp}, \mathsf{pk}, (c_1, c_2), (c_1', c_2')$
    Result: ciphertext $(c_1'', c_2'')$
    —
1   $c_1'' := c_1 \cdot c_1'$
2   $c_2'' := c_2 \cdot c_2'$
3   sample $r \xleftarrow{\$} \mathcal{D}_H$
4   return $(c_1'' \cdot h^r, c_2'' \cdot \mathsf{pk}^r)$
```

```
Algorithm 5: EvalScal
    Input: $\mathsf{pp}, \mathsf{pk}, (c_1, c_2), \alpha$
    Result: ciphertext $(c_1', c_2')$
    —
1   $c_1' := c_1^{\alpha}$
2   $c_2' := c_2^{\alpha}$
3   sample $r \xleftarrow{\$} \mathcal{D}_H$
4   return $(c_1' \cdot h^r, c_2' \cdot \mathsf{pk}^r)$
```

Fig. 1. Generic linearly homomorphic encryption scheme modulo M

to $\mathcal{D}_G$ yields also a distribution $\delta_{\mathcal{D}_G}$−close to the uniform in $\{1, \ldots, s\}$, and by the triangular inequality, we have

$$\left|\Pr[S_1] - \Pr[S_0]\right| \leq 2\delta_{\mathcal{D}_G}.$$

In the second game Game 2, the challenge ciphertext is computed using the secret key. Namely, $c_1 := h^r$ and $c_2 := f^{m_{b^\star}} c_1^{\mathsf{sk}}$ where $r \leftarrow \mathcal{D}_H$. Though the challenge ciphertext is computed differently, its distribution is in fact unchanged as $c_1^{\mathsf{sk}} = h^{r\mathsf{sk}} = \mathsf{pk}^r$. Hence,

$$\Pr[S_1] = \Pr[S_2].$$

In the third game Game 3, an integer u is sampled uniformly at random in $(\mathbf{Z}/M\mathbf{Z})^\times$ and c_1 is computed as $c_1 = f^u h^r$. By reduction to the $\mathsf{HSM}_{\mathsf{CL},\mathsf{M}}$ assumption, we have

$$\left|\Pr[S_3] - \Pr[S_2]\right| \leq \delta_{\mathsf{HSM}_{\mathsf{CL},\mathsf{M}}}.$$

In Game 3, we have $c_2 = f^{m_{b^\star}} c_1^{\mathsf{sk}} = f^{m_{b^\star}+u\cdot\mathsf{sk}}\mathsf{pk}^r$. From an information theoretic point of view, using the direct product of Proposition 1, the adversary learns $m_{b^\star} + u \cdot \mathsf{sk} \mod M$ from c_2. Moreover, it gets $u \mod M$ from c_1, but only $\mathsf{sk} \mod s$ from pk. Since sk is close to uniform modulo Ms, and M and s are coprime, $\mathsf{sk} \mod M$ is uniform and independent from $\mathsf{sk} \mod s$. Therefore, as u is invertible modulo M, $u\cdot\mathsf{sk}$ acts as a perfect one time pad for the message $m_{b^\star}$ and we get

$$\Pr[S_3] = \frac{1}{2}\cdot$$

Combining the above probability equations concludes the proof. $\square$

Remark 1. This generic construction of a linearly homomorphic encryption introduced to cover the CL family of cryptosystems actually also captures Camenisch and Shoup's encryption scheme from [13] (a linearly homomorphic encryption can be obtained from their encryption protocol by removing the last component of the ciphertext, which brings non-malleability). It involves an RSA modulus $N = pq$, for two safe primes p and q, and the message space is $\mathbf{Z}/N\mathbf{Z}$ ($M = N$ with our notations). The group $\widehat{G}$ is the subgroup of $(\mathbf{Z}/N^2\mathbf{Z})^\times$ of squares, H is the subgroup of the N-th powers, and F is the subgroup of order N generated by $1 + N$. In this case the $\mathsf{HSM}_{\mathsf{CL},\mathsf{M}}$ assumption corresponds to the Paillier's DCR assumption from [47]. We provide in our library an implementation of this encryption scheme (`CamenischShoup` class) and give comparison in Section 6. Our implementation benefits from all improvements of Section 5.2 that also apply to this setting.

3.4 A compact variant

The ciphertexts of the generic linearly homomorphic encryption scheme of Figure 1 use two elements of the group $\widehat{G}$. The element $c_2 = f^m\mathsf{pk}^r$ needs to be in $G \subset \widehat{G}$ in order to use the direct product $G \simeq F \times H$ of Proposition 1. However, c_1 is an element of the subgroup H, and H is isomorphic to the quotient group G/F. Moreover, in all known implementations of the CL framework, this quotient group is isomorphic to a group whose elements can be represented with a more compact representation than the elements of $\widehat{G}$.

In this subsection, we present a compact variant of the scheme of Figure 1 that benefits from this observation. To this end, we first enrich the setup of Definition 1 in order to add some properties: We consider a group $\widehat{\Gamma}$ and a surjection $\widehat{G} \to \widehat{\Gamma}$ of kernel F. These objects always exist by taking $\widehat{\Gamma} = \widehat{G}/F$ and the canonical surjection, be we further require that $\widehat{\Gamma}$ has an explicit and compact representation of elements.

These extra properties and this variant was originally proposed in [19] for the specific $M = q$ an odd prime, and presented as a *fast variant*. The security of this variant was then analyzed in [53], still presented as a *fast variant*. However, if this variant used to be fast compared to an encryption scheme that follows the original CL framework based on the DDH assumption, it is no longer the case compared to optimized implementation of schemes based on the $\mathsf{HSM}_{\mathsf{CL},\mathsf{M}}$

assumptions, as suggested by our benchmarks of Section 6. But this variant is still interesting for applications where the bandwidth is a critical issue as it leads to schemes with only a slight computational overhead due to the exponentiation to the power M and significantly reduces the size of ciphertexts. As a consequence we prefer the term *compact variant* in this work.

Definition 3 ($\mathsf{Setup}_{\mathsf{CL}_C}$)**.** *Let λ be a security parameter and M be a positive integer and* $\mathsf{Setup}_{\mathsf{CL}} = (\mathsf{Gen}_{\mathsf{CL}}, \mathsf{Solve}_{\mathsf{DL}})$ *a setup for the* CL *framework for M. Let $\widehat{\Gamma}$ be a group and $\pi : \widehat{G} \to \widehat{\Gamma}$ be a surjective map with kernel F.*

We say that $\mathsf{Setup}_{\mathsf{CL}_C}$ *is a setup for the compact* CL *framework, if we have the following additional properties:*

- *one can compute in $\widehat{\Gamma}$ in deterministic polynomial time;*
- *the representation of elements of $\widehat{\Gamma}$ is more compact than the representation of elements of $\widehat{G}$;*
- *one can evaluate π in polynomial time;*
- *given an element $\omega \in \widehat{\Gamma}$, one can efficiently lift it in $\widehat{G}$,* i.e.*, compute an element $\omega_\ell \in \pi^{-1}(\omega)$.*

The algorithm $\mathsf{Gen}_{\mathsf{CL_c}}$ *still outputs* $\mathsf{pp} := (\tilde{s}, f, h, \mathsf{extra})$*, where* extra *encodes both the necessary information to efficiently compute in $\widehat{G}$ and $\widehat{\Gamma}$.*

Proposition 2. *With the notation of Definitions 1 and 3, the map $\psi : \widehat{\Gamma} \to \widehat{G}$ s.t. $\omega \mapsto \omega_\ell^M$, where $\omega_\ell \in \pi^{-1}(\omega)$, is an injective morphism, computable in polynomial time. Let $\gamma = \pi(h)^M$ and $\Gamma := \langle\gamma\rangle$. With the notation of Proposition 1, ψ gives an isomorphism $\Gamma \simeq H$.*

Proof. This is a generalization of the proof of [19, Lemma 3] that we include for completeness. First ψ is well defined: if $\omega_\ell^{(1)}, \omega_\ell^{(2)} \in \pi^{-1}(\omega)$ are two distinct pre-images of ω then there exists an element $x \in F$ such that $\omega_\ell^{(1)} = x\omega_\ell^{(2)}$, and $(\omega_\ell^{(1)})^M = (\omega_\ell^{(2)})^M$ as F is of order M. Moreover it is easy to see that ψ is a morphism. Consider ω in $\widehat{\Gamma}$ such that $\psi(\omega) = \omega_\ell^M = 1$ in $\widehat{G}$, with $\omega_\ell \in \pi^{-1}(\omega)$. Applying π gives $\pi(\omega_\ell)^M = \omega^M = 1$ in $\widehat{\Gamma}$. Now observe that $\widehat{\Gamma}$ is of order $\hat{s}$ (as $\widehat{G}$ has order $M \cdot \widehat{s}$ and π is a surjection with kernel of order M). Moreover, $\hat{s}$ is prime to M so $\omega = 1 \in \widehat{\Gamma}$, so the map is injective. Eventually, ψ is computable in polynomial time as composition of operations computable in polynomial time. Concerning the map $\psi : \Gamma \simeq H$: firstly the restriction of π to H is injective. Moreover, denoting s the order of h, s is prime to M so $\gamma = \pi(h)^M = \pi(h^M)$ generates a group Γ of the same order s. Moreover, as $\psi(\gamma) = h^{M^2} \in H$ and ψ is injective, we get that $\psi : \Gamma \simeq H$. □

We describe in Figure 2 the compact variant of the scheme of Figure 1. The public key and the first element of the ciphertext are now elements of Γ instead of $\widehat{G}$. We then use the map ψ of Proposition 2 in order to generate the element of H that hides f^m in the second element of the ciphertext. The unnatural choice of $\gamma = \pi(h)^M$ instead of $\pi(h)$ is a generalisation of a trick introduced in [53] that makes the security proof goes trough.

Algorithm 6: KeyGen

```
    Input: $\mathsf{pp} = (\tilde{s}, f, h, \mathsf{extra})$
           generated by $\mathsf{Gen}_{\mathsf{CL_C}}$
    Result: $(\mathsf{pk}, \mathsf{sk})$
    —
1   $\gamma := \pi(h)^M$; append $\gamma$ to $\mathsf{pp}$
2   sample $\mathsf{sk} \xleftarrow{\$} \mathcal{D}_H$
3   $\mathsf{pk} := \gamma^{\mathsf{sk}}$
4   return $(\mathsf{pk}, \mathsf{sk})$
```

Algorithm 7: Encrypt

```
    Input: $\mathsf{pp}, \mathsf{pk}, m \in \mathbf{Z}/M\mathbf{Z}$
    Result: ciphertext $(c_1, c_2)$
    —
1   sample $r \xleftarrow{\$} \mathcal{D}_H$
2   $c_1 := \gamma^r$
3   $c_2 := f^m \psi(\mathsf{pk}^r)$
4   return $(c_1, c_2)$
```

Algorithm 8: Decrypt

```
    Input: $\mathsf{pp}, \mathsf{sk}, (c_1, c_2)$
    Result: $m \in \mathbf{Z}/M\mathbf{Z}$
    —
1   $a := c_2 \cdot \psi(c_1^{-\mathsf{sk}})$
2   return $\mathsf{Solve}_{\mathsf{DL}}(\mathsf{pp}, a)$
```

Algorithm 9: EvalAdd

```
    Input: $\mathsf{pp}, \mathsf{pk}, (c_1, c_2), (c_1', c_2')$
    Result: ciphertext $(c_1'', c_2'')$
    —
1   $c_1'' := c_1 \cdot c_1'$
2   $c_2'' := c_2 \cdot c_2'$
3   sample $r \xleftarrow{\$} \mathcal{D}_H$
4   return $(c_1'' \cdot \gamma^r, c_2'' \cdot \psi(\mathsf{pk}^r))$
```

Algorithm 10: EvalScal

```
    Input: $\mathsf{pp}, \mathsf{pk}, (c_1, c_2), \alpha$
    Result: ciphertext $(c_1', c_2')$
    —
1   $c_1' := c_1^{\alpha}$
2   $c_2' := c_2^{\alpha}$
3   sample $r \xleftarrow{\$} \mathcal{D}_H$
4   return $(c_1' \cdot \gamma^r, c_2' \cdot \psi(\mathsf{pk}^r))$
```

Fig. 2. Generic compact linearly homomorphic encryption scheme modulo M

Theorem 3. *Let* $\mathsf{Setup}_{\mathsf{CL}_C} = (\mathsf{Gen}_{\mathsf{CL_C}}, \mathsf{Solve}_{\mathsf{DL}})$ *be a setup for the compact* CL *framework for an integer* M*. The resulting linearly homomorphic encryption scheme modulo* M *of Figure 2 is* $\mathsf{ind-cpa}$ *under the* $\mathsf{HSM}_{\mathsf{CL,M}}$ *assumption for* $\mathsf{Setup}_{\mathsf{CL}_C}$*.*

Proof. We proceed with a sequence of games as in the proof of Theorem 2, starting with the ind-cpa experiment in Game 0.

In Game 1, the public key is computed as follows: first sample $\mathsf{sk}' \xleftarrow{\$} \mathcal{D}_H$ and set $\mathsf{pk} = \pi(h)^{\mathsf{sk}'}$.

We still have $\mathsf{pk} = \pi(h)^{M\mathsf{sk}} = \gamma^{\mathsf{sk}}$ for some sk defined modulo s, the order of h, since $\gcd(s, M) = 1$. Moreover, sk and sk' follow the same distribution modulo s. Therefore, the distribution of the public key is unchanged,

$$\Pr[S_0] = \Pr[S_1].$$

In $\mathsf{Game}\ 2$, we change the distribution of sk': it is sampled from $\mathcal{D}_G$ instead of $\mathcal{D}_H$. As in the first step of the proof of Theorem 2, one has

$$\left|\Pr[S_2] - \Pr[S_1]\right| \leq 2\delta_{\mathcal{D}_G}.$$

In $\mathsf{Game}\ 3$, we set $z := h^{r'}$ for $r' \overset{\$}{\leftarrow} \mathcal{D}_H$ and compute the challenge ciphertext as follows: $c_1 = \pi(z)$ and $c_2 = f^{m_{b^\star}} z^{\mathsf{sk}'}$.

Again, there exists r such that $c_1 = \pi(h)^{r'} = \pi(h)^{Mr} = \gamma^r$. Moreover, $c_2 = f^{m_{b^\star}} h^{\mathsf{sk}' r'}$. But $\psi(\mathsf{pk}^r) = \psi(\pi(h)^{\mathsf{sk}' r}) = h^{\mathsf{sk}' M r} = h^{\mathsf{sk}' r'} = z^{\mathsf{sk}'}$ as $Mr \equiv r' \pmod{s}$. Therefore, $c = (\gamma^r, f^{m_{b^\star}} \psi(\mathsf{pk}^r))$ is a genuine ciphertext of $m_{b^\star}$ for the public key pk with the correct distribution:

$$\Pr[S_3] = \Pr[S_2].$$

In $\mathsf{Game}\ 4$, we now set $z := f^u h^{r'}$ with u sampled uniformly at random in $(\mathbf{Z}/M\mathbf{Z})^\times$. By reduction to the $\mathsf{HSM}_{\mathsf{CL},\mathsf{M}}$ assumption, we have

$$\left|\Pr[S_4] - \Pr[S_3]\right| \leq \delta_{\mathsf{HSM}_{\mathsf{CL},\mathsf{M}}}.$$

In $\mathsf{Game}\ 4$, we have $c_2 = f^{m_{b^\star}} z^{\mathsf{sk}'} = f^{m_{b^\star} + u \cdot \mathsf{sk}'} h^{\mathsf{sk}' r'}$ and the only information on sk' known by the adversary is sk' modulo s (from pk). As in the last step of the proof of Theorem 2, one has $\mathsf{sk}' \mod M$ uniform and as u is invertible modulo M, $u \cdot \mathsf{sk}$ acts as a perfect one time pad for the message $m_{b^\star}$ and we get

$$\Pr[S_4] = \frac{1}{2}.$$

Combining the above probability equations concludes the proof. □

4 Instantiation in Class Groups

In this section, we present instantiations of the CL framework in class groups, for plaintexts spaces $\mathbf{Z}/M\mathbf{Z}$ where $M = 2^k$ and $M = q^k$ where q is an odd prime, of bitsize larger than the security parameter. The case 2^k was recently proposed in [21]. For the case q^k, the generator for $k = 1$ was originally proposed in [19] and a generalization for $k > 1$ was only suggested in that work. This generalization was then developed in [27]. These two works where focused on the DDH version of the CL framework, whereas we are interested in schemes based on the $\mathsf{HSM}_{\mathsf{CL},\mathsf{M}}$ assumption. Moreover [27] also consider the case of small odd primes q and product of different odd primes.

We do not consider the case of small primes, as it is required that q is large in order to have q prime to the class number with overwhelming probability for the $\mathsf{ind\text{-}cpa}$ proof to go through when working with the $\mathsf{HSM}_{\mathsf{CL},\mathsf{M}}$ assumption. Note that this property is also needed in many security proofs of applications of the DDH variant. Products of odd primes could also be obtained in our setting, by using Chinese remaindering during the computation of the discrete logarithm for decryption. It may also be possible to consider cases of even numbers such as

$2^k q$ but the generation process would be very technical for limited application interest.

In the next subsection we describe the generators for $M = 2^k$ and $M = q^k$ already given in [19,27,21] with some changes in order to follow the framework of the previous section. Moreover we unify the presentation of several variants. Firstly we show that these generators are all compatible with the compact variant of Subsection 3.4. This variant was previously only known to be compatible with message space $M = q$. Secondly, we simplify the exposition by presenting generators independent of the size of M, where as previous works imposed condition on the size of M or treated differently the case of large M's.

Then, in Subsections 4.2 and 4.3, we show that it is possible to optimize computations of exponentiations and discrete logarithms in the subgroup F. As we will see in Section 5, this gives speedup, particularly in the context of plaintexts spaces of orders q^k and 2^k with large k's. This will also improve dramatically the computation of discrete logarithms in the case of large M's.

4.1 Generators for the CL framework in class groups

Case $M = q^k$ Algorithm 11 implements the generator $\mathsf{Gen}_{\mathsf{CL}}$ for $\mathsf{Setup}_{\mathsf{CL}}$ for $M = q^k$ where q is an odd prime, of bitsize larger than the security parameter. We denote $\eta(\lambda)$ the bitsize of a fundamental discriminant Δ_K such that the computation of the class number $h(\Delta_K)$ and computation of discrete logarithms in $\mathrm{Cl}(\Delta_K)$ takes at least 2^λ operations (see Table 2 for concrete sizes). A bound B such that $B \leqslant h(\Delta_K) < 2B$ can be computed in polynomial time using the analytic class number formula (cf [45]). The bitsize of B is expected to be $\eta(\lambda)/2$.

Algorithm 11: $\mathsf{Gen}_{\mathsf{CL\text{-}HSM}_{q^k}}$

```
Input: $1^\lambda, k, q$ a $\mu$ bits prime with $\mu \geqslant \lambda$
Result: pp
—
1  if $\mu \geq \eta(\lambda)$ then
2      $p := 1$ if $q \equiv 3 \pmod 4$ or else take the smallest prime $p$
         such that $pq \equiv 3 \pmod 4$ and $\left(\frac{q}{p}\right) = -1$
3  else
4      sample $p$ a random $\eta(\lambda) - \mu$ bits prime such that
5      $pq \equiv 3 \pmod 4$ and $\left(\frac{q}{p}\right) = -1$
6  end
7  $\Delta_K := -pq$, $\Delta_{q^k} := q^{2k}\Delta_K$ and $\widehat{G} := \mathrm{Cl}(\Delta_{q^k})^2$
8  compute $\tilde{s}$ an upper bound on the order of $\widehat{\Gamma} := \mathrm{Cl}(\Delta_K)^2$
9  sample $t$ in $\widehat{G}$ and set $h := t^{q^k}$
10 $f := (q^{2k}, q^k, \frac{1-\Delta_K}{4})$ in $\widehat{G}$
11 return pp $:= (\tilde{s}, f, h, \mathsf{extra} := (\Delta_K, q, k))$
```

The algorithm first computes a fundamental discriminant $\Delta_K = -q$ or $\Delta_K = -pq$ where p is another prime, with $\Delta_K \equiv 1 \pmod 4$ and Δ_K of bitsize at least $\eta(\lambda)$. The additional requirement on the Legendre symbol ensures that the group of squares $\widehat{\Gamma} := \mathrm{Cl}(\Delta_K)^2$ has odd order $\hat{s}$, where $\hat{s} = h(\Delta_K)$ or $h(\Delta_K)/2$ depending if $\Delta_K = -q$ or $\Delta_K = -pq$ (see [19]). From the Cohen-Lenstra heuristics, $\gcd(q, \hat{s}) = 1$ with overwhelming probability as q is a μ-bit prime with $\mu \geqslant \lambda$.

This computation of Δ_K is done in order to have Δ_K of the minimal size with the constraints that $q|\Delta_K$ and Δ_K is large enough to meet the security level. Moreover, we do not set an upper bound on the plaintexts space size: For instance, for $k = 1$, [19] supposed that q was not to large to ensure an easy computation of discrete logarithms in F, and proposed another process, called the "large message variant", for large q's. We here proposed a unified view of these two settings (see Subsection 4.3 for the consequences on the computation of discrete logarithms in F).

We then denote $\widehat{G} = \mathrm{Cl}(\Delta_{q^k})^2$ the group of squares of $\mathrm{Cl}(\Delta_{q^k})$. It has order $\hat{s}q^k$ and $\gcd(q^k, \hat{s}) = 1$ with overwhelming probability. The subgroup F generated by $f := (q^{2k}, q^k, \frac{1-\Delta_K}{4})$ has order q^k (we prove this in Theorem 5 as this does not seem to have been formally proven elsewhere). This subgroup is the kernel of the surjection φ_{q^k} from $\mathrm{Cl}(\Delta_{q^k})$ to $\mathrm{Cl}(\Delta_K)$. Note that F is contained in $\mathrm{Cl}(\Delta_{q^k})^2$ as F has odd order.

The description of the polynomial time algorithm that computes discrete logarithms in F is deferred in Subsection 5.4: Algorithm 13. It contains several new optimizations to compute in F, introduced in the next subsections.

One can recognize elements of $\widehat{G}$ as it is possible to recognize squares given the factorization of the discriminant. Note that Algorithm 11 is a public coin algorithm: we can publish the randomness used to generate the parameters.

Algorithm 11 also implements a generator for $\mathsf{Setup}_{\mathsf{CL}_C}$ by setting $\widehat{\Gamma} := \mathrm{Cl}(\Delta_K)^2$. Indeed, elements of $\mathrm{Cl}(\Delta_K)^2$ have shorter representations than those of $\mathrm{Cl}(\Delta_{q^k})^2$: this saves $3/2 \cdot k\log_2(q)$ bits using the compression of [30]. The homomorphism φ_{q^k} between the two class groups can be restricted from $\mathrm{Cl}(\Delta_{q^k})^2$ to $\mathrm{Cl}(\Delta_K)^2$ and its kernel is still F, so it plays the role of the map π in $\mathsf{Setup}_{\mathsf{CL}_C}$. If $\omega \in \mathrm{Cl}(\Delta_K)^2$, one can find a pre-image by φ_{q^k} by computing $[I \cap \mathcal{O}_{\Delta_{q^k}}]$ for I an ideal prime to q in the class ω. This can be done efficiently using [37, Algo. 2].

Case $\boldsymbol{M = 2^k}$ Algorithm 12 implements a generator for $\mathsf{Setup}_{\mathsf{CL}}$ for $M = 2^k$. This is the original algorithm proposed in [21], with a restriction on the choice of the primes p and q to ease both exposition and implementation. The overall idea of the generator is similar to the case q^k with the notable change that the factorisation of Δ_K must be kept secret in order to prevent an adversary from computing square roots and recognizing squares, which would break the ind-cpa security of the encryption scheme. We thus choose primes p and q of bitsize $\eta'(\lambda)$ where $\eta'(\lambda)$ denotes the bitsize of N such that the best algorithms for factoring $N := pq$ take 2^λ operations (again, see Table 2 for concrete sizes). As a result the coins of the algorithm can not be given to an adversary. The algorithm that computes discrete logarithms in F is similar to the q^k case, as we will see in Subsection 5.4.

Algorithm 12: $\mathsf{Gen}_{\mathsf{CL\text{-}HSM}_{2^k}}$

```
  Input: $1^\lambda$, $k \geqslant 1$
  Result: $\mathsf{pp}$
  —
1 sample two random distinct $\eta'(\lambda)$ bits primes $p$, $q$ such that
    $(p \mod 8, q \mod 8) \in \{(3,5),(5,3)\}$
2 $N := pq$
3 $\Delta_K := -8N$, $\Delta_{2^{k+1}} := 2^{2k+2}\Delta_K$ and $\widehat{G} := \mathrm{Cl}(\Delta_{2^{k+1}})^2$
4 compute $\tilde{s}$ an upper bound on the order $\widehat{\Gamma} := \mathrm{Cl}(\Delta_K)^2$
5 sample $t$ in $\widehat{G}$ and set $h := t^{2^k}$
6 $f := (2^{2k}, 2^{k+1}, 1-\Delta_K) \in \widehat{G}$
7 return $\mathsf{pp} := (\tilde{s}, f, h, \mathsf{extra} := (\Delta_K, k))$
```

We now prove that this algorithm also implements a generator for $\mathsf{Setup}_{\mathsf{CL}_C}$, which was not shown in [21]. The situation is however a little bit intricate compared to the case q^k. We still consider for π the restriction of $\varphi_{2^{k+1}}$ from $\widehat{G} := \mathrm{Cl}(\Delta_{2^{k+1}})^2$ to $\widehat{\Gamma} := \mathrm{Cl}(\Delta_K)^2$. However the kernel of $\varphi_{2^{k+1}}$ is now larger than F, it is of order 2^{k+1} and $\ker\varphi_{2^{k+1}} \not\subset \widehat{G}$. Actually, $F = \ker\varphi_{2^{k+1}} \cap \widehat{G}$. If $\omega \in \mathrm{Cl}(\Delta_K)^2$, computing a pre-image ρ by $\varphi_{2^{k+1}}$: $\rho := [I \cap \mathcal{O}_{\Delta_{q^k}}]$ for I an ideal prime to 2 may give an element which is not in $\widehat{G}$, as $\varphi_{2^{k+1}}^{-1}(\{\omega\}) = x \cdot \ker\varphi_{2^{k+1}}$ where $x \in \widehat{G}$.

This can be checked and solved thanks to genus theory (we refer the reader to [21] for background on genus theory for this specific class group). As in the proof of [21, Theorem 1], one can prove that the $\tilde{f} = (2^{2k+2}, 2^{k+2}, \frac{4-\Delta_K}{4})$ is in $\ker\varphi_{2^{k+1}} \setminus F$ (actually, this is a generator of $\ker\varphi_{2^{k+1}}$). This element $\tilde{f}$ has genus $(1,1,-1,1)$ for the respective characters $(\chi_p, \chi_q, \chi_{-4}, \chi_8)$. Moreover, elements of $\ker\varphi_{2^{k+1}} \setminus F$ are in the same genus. Elements of F are squares, so their genus is $(1,1,1,1)$.

Now let us see how to finish our lift. We have similarly half the elements of $\varphi_{2^{k+1}}^{-1}(\{\omega\})$ in $\widehat{G}$, and half of genus $(1,1,-1,1)$. Back to our element ρ, one can check the value of χ_{-4} efficiently without knowing the factorization of N. If this gives 1, ρ is a square and we are done. Otherwise, we multiply it by $\tilde{f}$ in order to get a square. In both case, we have obtained an element of $\varphi_{2^{k+1}}^{-1}(\{\omega\})$ which is a square, so an element of $\pi^{-1}(\{\omega\})$.

Note that one can not recognize elements of $\widehat{G}$ nor of $\widehat{\Gamma}$. The situation is similar to elements of Jacobi symbol 1 and squares of $\mathbf{Z}/N\mathbf{Z}$. Here we can only recognize if elements have genus $(1,1,1,1)$ or $(-1,-1,1,1)$.

Concrete linearly homomorphic encryption schemes When using the generator $\mathsf{Gen}_{\mathsf{CL\text{-}HSM}_{q^k}}$ with the generic constructions of Figure 1 (resp. Figure 2), one gets an encryption scheme linearly homomorphic modulo q^k for an odd prime q, denoted $\mathsf{CL\text{-}HSM}_{q^k}$ (resp. $\mathsf{CL_C\text{-}HSM}_{q^k}$ for the compact variant). Likewise, the generator $\mathsf{Gen}_{\mathsf{CL\text{-}HSM}_{2^k}}$ gives a scheme homomorphic modulo 2^k, denoted $\mathsf{CL\text{-}HSM}_{2^k}$ (resp. $\mathsf{CL_C\text{-}HSM}_{2^k}$ for the compact variant).

On the selection of the plaintext space size In most of the cases, the choice of the generator will be guided by the application, as a specific form of M will be needed. However, it might be the case that one needs a linearly encryption scheme modulo an integer M of a certain bitsize ν, without any particular property on M.

We discuss here on the best choice of the generator in this context in order to optimize ciphertexts size. First, we summarize in Table 1, the bitsize of the ciphertexts for these schemes using the compression technique of [30]. The different sizes of ciphertexts for the q^k variants comes from the fact that when $\log(q) \leqslant \eta(\lambda)$, Algorithm 11 chooses a fundamental discriminant Δ_K of minimal size $\eta(\lambda)$ to ensure security, where as when $\log(q) > \eta(\lambda)$, Δ_K will have approximately the same bitsize as q.

Scheme	Size of plaintexts	Size of ciphertexts	Condition
$\mathsf{CL\text{-}HSM}_{q^k}$	$k \log q$	$\frac{3}{2}\eta(\lambda) + 3k\log(q)$	$\lambda \leqslant \log(q) \leqslant \eta(\lambda)$
$\mathsf{CL_C\text{-}HSM}_{q^k}$	$k \log q$	$\frac{3}{2}\eta(\lambda) + \frac{3}{2}k\log(q)$	$\lambda \leqslant \log(q) \leqslant \eta(\lambda)$
$\mathsf{CL\text{-}HSM}_{q^k}$	$k \log q$	$(3k + \frac{3}{2})\log(q)$	$\eta(\lambda) < \log(q)$
$\mathsf{CL_C\text{-}HSM}_{q^k}$	$k \log q$	$\frac{3}{2}(k+1)\log(q)$	$\eta(\lambda) < \log(q)$
$\mathsf{CL\text{-}HSM}_{2^k}$	k	$\frac{3}{2}\eta'(\lambda) + 3k + \frac{15}{2}$	-
$\mathsf{CL_C\text{-}HSM}_{2^k}$	k	$\frac{3}{2}\eta'(\lambda) + \frac{3}{2}k + 6$	-

Table 1. Ciphertext sizes of class groups based schemes

To optimize ciphertext expansion, for plaintexts of bitsize $\nu < \lambda$, only the 2^k setting is possible. With $\mathsf{CL_C\text{-}HSM}_{2^k}$ using $k = \nu$, the expansion is:

$$\frac{3}{2}\frac{\eta'(\lambda)}{\nu} + \frac{6}{\nu} + \frac{3}{2}.$$

For plaintexts of bitsize ν with $\lambda \leqslant \nu$, $\mathsf{CL_C\text{-}HSM}_{q^k}$ is the best choice. One can get a ciphertext expansion of

$$\frac{3}{2}\frac{\eta(\lambda)}{\nu} + \frac{3}{2}.$$

This tends to 3/2 when the bitsize of messages tends to infinity, for a fixed level of security. The choice of q and k to meet this expansion depends on the value of ν. For $\lambda \leqslant \nu \leqslant \eta(\lambda)$, one can use any combination of the prime q and power k such that $\nu = k \log q$ and $\log q > \lambda$. Using $k = 1$ and a prime q of bitsize ν is thus the simplest choice in this case. For plaintexts of bitsize $\nu > \eta(\lambda)$, one has to choose the prime q and the integer k such that $\log q = \nu/k \leqslant \eta(\lambda)$.

Note that as $\eta'(\lambda) > \eta(\lambda)$, the 2^k setting is less interesting for large messages, in terms of compactness.

4.2 Explicit isomorphisms for $\ker \varphi$

For $\mathsf{CL\text{-}HSM}_{\mathsf{q}^k}$, the cyclic group F of size $M = q^k$ in which the discrete logarithm should be easy is exactly the kernel of the surjective homomorphism φ_{q^k} from $\mathrm{Cl}(\Delta_{q^k})$ to $\mathrm{Cl}(\Delta_K)$. For $\mathsf{CL\text{-}HSM}_{2^k}$, the cyclic group F is of size $M = 2^k$ and is a subgroup of index 2 of the kernel of the surjective homomorphism $\varphi_{2^{k+1}}$. To be able to compute discrete logarithms in F, we will show how to represent its elements in a manner that allows very fast computations. The main tool used in previous works to analyse these kernels is the fact that there exists an effective isomorphism with some quotient of the ring of integers. This is summarize in the following theorem.

Theorem 4. *Let $\Delta_K < -4$ be a fundamental negative discriminant, ℓ be a positive integer and $\Delta_\ell = \ell^2 \Delta_K$. Then, the kernel of the surjective homomorphism $\varphi_\ell : \mathrm{Cl}(\Delta_\ell) \twoheadrightarrow \mathrm{Cl}(\Delta_K)$ is isomorphic to*

$$\mathbb{G}_\ell := \left(\mathcal{O}_{\Delta_K}/\ell\mathcal{O}_{\Delta_K}\right)^\times / \left(\mathbf{Z}/\ell\mathbf{Z}\right)^\times .$$

Moreover, let α represents an equivalence class in $\mathbb{G}_\ell$. Denote by (a, b, c) a form corresponding to the ideal $\alpha\mathcal{O}_{\Delta_K}$, where a is coprime to ℓ. Then the class of $\ker \varphi_\ell$ corresponding to α is the class of the (non-necessarily reduced) form $(a, b\ell, c\ell^2)$.

Proof. See [26, Prop. 7.20, Prop. 7.22, Th. 7.24] for the theoretical side. See [36, Prop. 2], [37, Algo. 1–3] and [8, Prop 2.9] for the computational side. □

We then apply this theorem to get a correspondence between the representation of elements of F in the q^k setting with quadratic integers.

Theorem 5. *Let k, q, Δ_K, Δ_{q^k}, f and F be generated as in Algorithm 11. Then, the set $\{1 + t\sqrt{\Delta_K} \mid t = 0 \text{ or } t \text{ odd in } -q^k < t < q^k\}$ has size q^k and is a complete set of representatives of $\mathbb{G}_{q^k}$. The explicit group isomorphism of Theorem 4 is given by*

$$\begin{aligned} \mathbb{G}_{q^k} &\to F \\ 1 + t\sqrt{\Delta_K} &\mapsto \left(q^{2j}, uq^j, \frac{u^2 - q^{2(k-j)}\Delta_K}{4}\right), \end{aligned}$$

where $j = 0$ and $u = 1$ for $t = 0$ and $j = k - \mathrm{val_q}\, t$ and $u = \left(\frac{t}{q^{k-j}}\right)^{-1} \bmod 2q^j$ with centered remainder otherwise.

The generator f of F corresponds to $1 + \sqrt{\Delta_K}$, its order is q^k. Moreover, all the forms are reduced if and only if $q^{2k} \leq \frac{1-\Delta_K}{4}$.

Proof. The proof is similar to [19, Proposition 1] that proves the case $k = 1$.

For $t = 0$, the ideal generated by 1 is the maximal order $\mathcal{O}_{\Delta_K}$. When lifted in $\mathcal{O}_{\Delta_{q^k}}$ using [37, Algo. 2], it corresponds to the ideal $\mathbf{Z} + \frac{q^k+\sqrt{\Delta_{q^k}}}{2}\mathbf{Z}$. This ideal corresponds to the form $(1, -q^k, c)$ with c such that its discriminant is Δ_{q^k}, which is equivalent to the reduced form $(1, 1, (1 - \Delta_{q^k})/4)$.

Let t be an odd integer in $-q^k < t < q^k$. Using [8, Prop 2.9], we obtain that the ideal generated by $\frac{1+t\sqrt{\Delta_K}}{2}$, an element equivalent to $1+t\sqrt{\Delta_K}$ in $\mathbb{G}_{q^k}$, is $a\mathbf{Z}+\frac{b+\sqrt{\Delta_{q^k}}}{2}\mathbf{Z}$ by setting $a = \frac{1-t^2\Delta_K}{4}$ and $b = t\Delta_K \bmod 2a$ with centered remainder. As a is coprime with q, we can use [37, Algo. 2] to lift it in $\mathcal{O}_{\Delta_{q^k}}$ and we obtain the ideal $a\mathbf{Z}+\frac{bq^k+\sqrt{\Delta_{q^k}}}{2}\mathbf{Z}$. It corresponds to the form $(\frac{1-t^2\Delta_K}{4}, -t\Delta_K q^k, -q^{2k}\Delta_K)$. Let u and v be the Bézout coefficients of the two coprime integers $\frac{t}{q^{k-j}}$ and $2q^j$: $\frac{t}{q^{k-j}}u + 2q^j v = 1$. We can always choose u such that $-q^j < u < q^j$. Finally, we apply the transformation matrix

$$\begin{pmatrix} 2q^j & u \\ -\frac{t}{q^{k-j}} & v \end{pmatrix}$$

of determinant 1 to the form and obtain $(q^{2j}, q^j u, \frac{u^2-q^{2(k-j)}\Delta_K}{4})$.

The fact that the form $f = (q^{2k}, q^k, \frac{1-\Delta_K}{4})$ corresponds to $1 + \sqrt{\Delta_K}$ comes from the case $t = 1$. We then prove that f is a generator of F. As the order of F is q^k, it is sufficient to prove that $\left(1 + \sqrt{\Delta_K}\right)^{q^{k-1}} \neq 1$.

We prove this by induction by showing that for $i \geqslant 2$,

$$\left(1 + \sqrt{\Delta_K}\right)^{q^{i-1}} \equiv 1 + q^{i-1}\sqrt{\Delta_K} \pmod{q^i}.$$

For $i = 2$, $\left(1 + \sqrt{\Delta_K}\right)^q = \sum_{j=0}^{q} \binom{q}{j} \sqrt{\Delta_K}^j$. As $q|\Delta_K$ and $q|\binom{q}{j}$ for $j = 1, \ldots, q-1$, we get $\left(1 + \sqrt{\Delta_K}\right)^q \equiv 1 + q\sqrt{\Delta_K} \pmod{q^2}$. Now suppose that there exists a quadratic integer α such that $\left(1 + \sqrt{\Delta_K}\right)^{q^{i-1}} = 1 + q^{i-1}\sqrt{\Delta_K} + \alpha q^i = 1 + q^{i-1}(\sqrt{\Delta_K} + \alpha q)$. We then rise this equality to the power q, and using again binomial expansion, one obtains $\left(1 + \sqrt{\Delta_K}\right)^{q^i} \equiv 1 + q^i\sqrt{\Delta_K} \pmod{q^{i+1}}$ as for $j \geqslant 2$, $q^{j(i-1)} \equiv 0 \pmod{q^{i+1}}$ as $i \geqslant 2$. □

Now we use Theorem 4 to get a correspondence between the representation of elements of F in the 2^k setting with quadratic integers.

Theorem 6. *Let k, Δ_K, $\Delta_{2^{k+1}}$, f and F be generated as in Algorithm 12. Then, the set $\{1 + \tau\sqrt{-2N} \mid 0 \leq \tau < 2^{k+1}\}$ has size 2^{k+1} and is a complete set of representatives of $\mathbb{G}_{2^{k+1}}$. We will consider the subset $\{1 + \tau\sqrt{-2N} \mid 0 \leq \tau < 2^{k+1}, \tau \text{ even}\} = \{1 + t\sqrt{\Delta_K} \mid 0 \leq t < 2^k\}$. The explicit group isomorphism of Theorem 4 restricted to this subset of representatives is given by*

$$\begin{aligned} \mathbb{G}_{2^{k+1}} &\to F \\ 1 + t\sqrt{\Delta_K} &\mapsto \left(2^{2j}, u2^{j+1}, u^2 - 2^{2(k-j)}\Delta_K\right), \end{aligned}$$

where $j = 0$ and $u = 0$ for $t = 0$ and $j = k - \mathrm{val}_2\, t$ and $u = \left(\frac{t}{2^{k-j}}\right)^{-1} \bmod 2^j$ with centered remainder for $0 < t < 2^k$.

The generator f of F corresponds to $1 + \sqrt{\Delta_K}$, its order is 2^k and it is a square. Note that all the forms are reduced if and only if $2^{2k} \leq 1 - \Delta_K$.

Proof. For $t = 0$, the ideal generated by 1 is the maximal order $\mathcal{O}_{\Delta_K}$. When lifted in $\mathcal{O}_{\Delta_{2^{k+1}}}$ using [37, Algo. 2], it corresponds to the ideal $\mathbf{Z} + \frac{2^{k+1}+\sqrt{\Delta_{2^{k+1}}}}{2}\mathbf{Z}$. This ideal corresponds to the form $(1, -2^{k+1}, c)$ with c such that its discriminant is $\Delta_{2^{k+1}}$, which is equivalent to the reduced form $(1, 0, -\frac{\Delta_{2^{k+1}}}{4})$.

Let t be an integer in $0 < t < 2^k$. Using [8, Prop 2.9], we obtain that the ideal generated by $1+t\sqrt{\Delta_K}$ is $a\mathbf{Z} + \frac{b+\sqrt{\Delta_K}}{2}$ with $a = 1-t^2\Delta_K$ and $b = t\Delta_K \bmod 2a$ with centered remainder. As a is odd, we can use [37, Algo. 2] to lift it in $\mathcal{O}_{\Delta_{2^{k+1}}}$ and we obtain the ideal $a\mathbf{Z} + \frac{b2^{k+1}+\sqrt{\Delta_{2^{k+1}}}}{2}$. It corresponds to the form $(a, -b2^{k+1}, c)$ with c such that the discriminant is $\Delta_{2^{k+1}}$. It is equivalent to $(1 - t^2\Delta_K, -t\Delta_K 2^{k+1}, -2^{2k}\Delta_K)$. Let u and v be the Bézout coefficients of the two coprime integers $\frac{t}{2^{k-j}}$ and 2^j: $\frac{t}{2^{k-j}}u + 2^j v = 1$. We can always choose u such that $-2^{j-1} < u < 2^{j-1}$. Finally, we apply the transformation matrix

$$\begin{pmatrix} 2^j & u \\ -\frac{t}{2^{k-j}} & v \end{pmatrix}$$

of determinant 1 to the form and obtain $(2^{2j}, 2^{j+1}u, u^2 - 2^{2(k-j)}\Delta_K)$.

The fact that the form $f = (2^{2k}, 2^{k+1}, 1 - \Delta_K)$ corresponds to $1 + \sqrt{\Delta_K}$ comes from the case $t = 1$. The fact that it is a square of order 2^k was proven in [21, Theorem 1]. □

4.3 Computations in F

Theorems 5 and 6 are used to speedup computation in the subgroup F. During encryption, one need to compute f^m, where f is the generator of the group F and m in a non-negative integer less than M. This computation can be done without using quadratic forms. Instead one computes $\left(1 + \sqrt{\Delta}\right)^m$ in $\mathbb{G}_{q^k}$ (resp. $\mathbb{G}_{2^{k+1}}$), then Theorem 5 (resp. Theorem 6) is used to get the form corresponding to this element. It will be the quadratic form corresponding to f^m. Implementation details are given in Subsection 5.3.

During decryption, one need to compute the discrete logarithm of a given element $f_t \in F$ in basis f. For more efficiency, we move the problem into $\mathbb{G}_{q^k}$ or $\mathbb{G}_{2^{k+1}}$. In these groups, the discrete logarithms can be easily computed with $O(k)$ operations (see Subsection 5.4 for implementation details). However, in order to use this method we need to find the representative of f_t as given in Theorems 5 and 6. This is direct if this representative is reduced (*i.e*, if q^k or 2^k is small enough compared to $|\Delta_K|$ as stated in the theorems). Otherwise, we use [38, Algo. 1] to invert the maps of Theorems 5 and 6. This last case corresponds to the *large message variant* of [19]. In this work, the form f_t was lifted in another non-maximal order, chosen to make some adaptation of the representative forms reduced. This involves an exponentiation to the power q^k or 2^k. Our new method in this case is much faster. As a result, the distinction between the two cases becomes irrelevant from the user's point of view: the cost of computing the representative is completely negligible in both cases.

5 Algorithmic Perspective

In this section, we present some algorithmic aspects of our implementation in `BICYCL` of the arithmetic of class groups of imaginary quadratic fields and of the different cryptosystems described in the previous sections.

We first give some details of our implementation choices for the composition of quadratic forms and for exponentiation, from which can benefit any cryptographic protocols based on class groups. This can be found in the classes `QFI` and `ClassGroup` of `BICYCL`.

We then focus on the exponentiation and discrete logarithm computations in the subgroup F defined in Section 4 in the context of the `CL` encryption schemes.

5.1 Composition of quadratic forms

For the composition and reduction of quadratic forms, we implemented Shanks' NUCOMP algorithm [50] (as well as NUDUPL for the computation of the square of a form). At the lowest level, our implementation makes extensive use of GMP's arithmetic functions [33], with the exception of the partial extended GCD. Partial extended GCD is a core component of NUCOMP. It performs the same steps as a classical extended GCD but terminates once the computed remainders reach a given bound. In the context of NUCOMP and NUDUPL, a common choice for this bound is $\Delta^{1/4}$. We implemented this function by modifying one of the extended GCD algorithm implemented in GMP: Lehmer's variant[4]. GMP implementation of Lehmer's variant uses the two most-significant words to approximate the successive quotients appearing in the extended GCD algorithm. We modify the algorithm to stop it once the remainders were below the targeted bound. The Flint library [35] also implements the Lehmer's variant for its function `xgcd_partial` but only uses the most-significant word to approximate the quotient. The PARI/GP library [48], in its function `parteucl` implementing the partial extended GCD, performs a full euclidean division at each step.

In Figure 3, we compare our implementation of NUDUPL to the implementation found in Flint (via the add-on Antic library) and PARI/GP. We also compare our implementation to libqform [49] which is the library used to implement several `CL` variants from [27]. Finally, we compare to the implementation that won the Chia VDF competition [23], whose goal was to perform successive NUDUPL as fast as possible. Our implementation improves all these libraries, for all cryptographic sizes of discriminant. Comparisons for NUCOMP gave very similar timings.

[4] For more details on GMP implementation of Lehmer's variant, see GMP documentation `https://gmplib.org/manual/Lehmer_0027s-Algorithm`

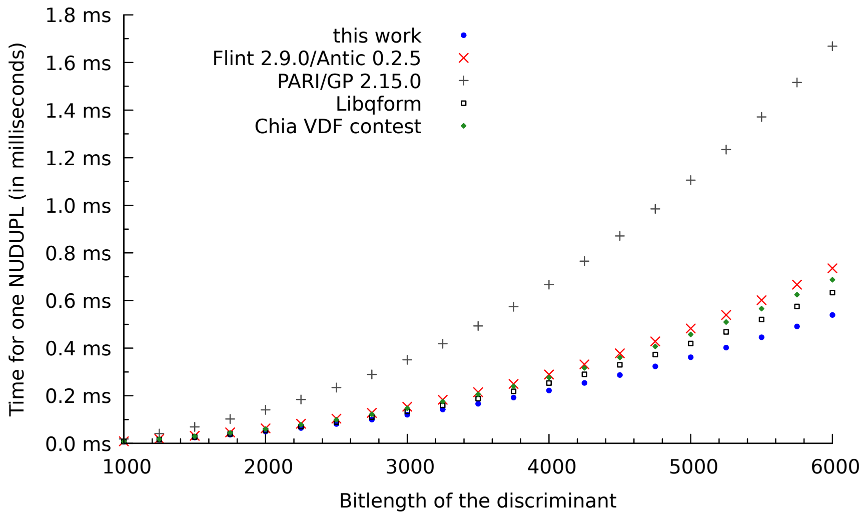

Fig. 3. Comparison for the NUDUPL operation for different libraries. All timings were computed using one thread of a Intel i7-8665U CPU at 1.90GHz. For each discriminant size, NUDUPL is executed 10000 times for 10 different random discriminants.

5.2 Exponentiations in the class group

The Encrypt and Decrypt Algorithms involve exponentiations of quadratic forms. For Encrypt, both h and pk are known in advance, which opens the door to various optimization strategies at the cost of some precomputations and extra storage. For Decrypt, one needs to compute and invert the form c_1^{sk}. Since c_1 is not known in advance, the above optimizations do not apply. In class groups, we note that inverting a form is virtually free[5]. Thus, one can take advantage of signed-digit representations and the numerous algorithms developed in the context of elliptic curves. We detail the algorithms implemented in our library in the next paragraphs.

Encrypt – fixed-basis exponentiations: Recall that the Encrypt algorithm, described in Algorithm 2, outputs a pair (c_1, c_2) with $c_1 = h^r$ and $c_2 = f^m \mathsf{pk}^r$. The special, easier case of the computation of f^m will be explain in Subsection 5.3.

Let us focus on the other two exponentiations. Our code evaluates h^r and pk^r in parallel using the same algorithm[6], which mixes a variant of the width-2 comb method [46] and Solinas' Joint Sparse Form (JSF) [51].

[5] If $f = (a, b, c)$, then the inverse of the class of f is represented by $(a, -b, c)$.
[6] There does not seem to be an efficient way to exploit the fact that one raises both basis to the same power.

Let us assume that r is a $2n$-bits integer. In the comb method, the scalar r is divided into w chunks of $2n/w$ bits each. The bits of r are then scanned w at a time. In our implementation, the $2n$-bit scalar r is split in half (i.e. $w = 2$) so that $r = r_0 + 2^n r_1$. Hence, we get $h^r = h^{r_0} \times h_n^{r_1}$ with $h_n = h^{2^n}$. Since h is fixed, and the size of r is bounded, one can precompute h_n offline. Hence, this reduces a $2n$-bits exponentiation to an n-bits double-exponentiation which is implemented using Straus-Shamir's trick [52]. Using minor extra-storage for h_n, one can thus divide by 2 the number of calls to NUDUPL.

As already observed, one can take advantage of algorithms based on signed-digit representations. In particular, Solinas' Joint Sparse Form [51] allows to jointly rewrite the pair (r_0, r_1) of n-bit exponents using digits in $\{-1, 0, 1\}$ so that the double-exponentiation requires exactly n NUDUPL and $n/2$ NUCOMP on average. For computing $h^{r_0} \times h_n^{r_1}$, each call to NUCOMP involve either h, h_n, $h \times h_n$, $h \times h_n^{-1}$ or their respective inverse. Note that there is no need to store both a form and its inverse.

Finally, since the cost of NUDUPL is almost identical to that of NUCOMP, it is advantageous to adapt to our context the variant of the comb method presented in [46, Algo 3.45], even for $w = 2$. This approach further divides the number of NUDUPL by 2, at the cost of $(3^4-1)/2 = 40$ extra precomputed forms which can be computed with 36 calls to NUCOMP. These precomputed forms are obtained as products of forms in $h^{\pm 1}$, $h_{n/2}^{\pm 1}$, $h_n^{\pm 1}$, $h_{3n/2}^{\pm 1}$, where $h_{n/2} = h^{2^{(n/2)}}$ and $h_{3n/2} = h^{2^{3n/2}}$.

Decrypt – variable basis: The Decrypt algorithm, described in Algorithm 3, involves one exponentiation $c_1^{-\mathsf{sk}}$ where the exponent is the secret key. Contrary to the encryption algorithm, the base c_1 is not known in advance. Hence, the above techniques are not advantageous. Our implementation uses a left-to-right wNAF exponentiation with $w = 7$ which does not require to scan the bits of the exponents first for the NAF recoding (cf [40]). Instead of a fixed size, one could opt for an adaptive strategy for the window width depending on the size of the discriminant. But we reckon that the gain should be very marginal.

5.3 Exponentiations in F

As described in Subsection 4.3, the computation of an exponentiation in F corresponds to computing an exponentiation of $1 + \sqrt{\Delta_K}$ in $\mathbb{G}_M$. Efficient exponentiations of quadratic integers can be done using Lucas sequence as

$$\left(\frac{P + \sqrt{\Delta_K}}{2}\right)^n = \left(\frac{V_n(P,Q) + U_n(P,Q)\sqrt{\Delta_K}}{2}\right)^n,$$

where $U_n(P,Q)$ and $V_n(P,Q)$ are the Lucas sequences of the first and second kind and Q is such that $\Delta_K = P^2 - 4Q$. Terms of Lucas sequences can be easily computed using the fact that

$$\begin{pmatrix} U_n(P,Q) \\ U_{n+1}(P,Q) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -Q & P \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix} \text{ and } \begin{pmatrix} V_n(P,Q) \\ V_{n+1}(P,Q) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -Q & P \end{pmatrix}^n \begin{pmatrix} 2 \\ P \end{pmatrix}.$$

Note that we only need to compute $\begin{pmatrix} 0 & 1 \\ -Q & P \end{pmatrix}^n$ modulo M.

For $\mathsf{CL\text{-}HSM}_{q^k}$, we use $P = 1$ and $Q = \frac{1-\Delta_K}{4}$. The values $U_n(P,Q)$ and $V_n(P,Q)$ are computed modulo q^k by computing the power of the above matrix. Then, we compute $t = U_n(P,Q)\left(V_n(P,Q)\right)^{-1} \bmod q^k$ and obtain the representative $1 + t\sqrt{\Delta_K}$ (note that $V_n(P,Q)$ is guaranteed to be invertible modulo q^k). Then we can use Theorem 5 to find the corresponding form.

For $\mathsf{CL\text{-}HSM}_{2^k}$, we use $P = 2$ and $Q = 1 - \Delta_K$. The values $U_n(P,Q)$ and $V_n(P,Q)$ are computed modulo 2^k by computing the power of the above matrix. Then, we compute $t = U_n(P,Q)\left(\frac{V_n(P,Q)}{2}\right)^{-1} \bmod 2^k$ and obtain the representative $1 + t\sqrt{\Delta_K}$ (note that $\frac{V_n(P,Q)}{2}$ is guaranteed to be odd). Then we can use Theorem 6 to find the corresponding form.

Note that in the case of $\mathsf{CL\text{-}HSM}_q$, it is even easier to compute t as we have $\left(1 + \sqrt{\Delta_K}\right)^n \equiv 1 + n\sqrt{\Delta_K}$ in $\mathbb{G}_q$ as q divides Δ_K, as originally remarked in [19].

5.4 Discrete logarithms in F

The first step to efficiently compute discrete logarithms in F is to move the problem into $\mathbb{G}_M$ as described in Section 4.3. It remains to compute the discrete logarithm of an element $1 + t\sqrt{\Delta_K}$ in basis $1 + \sqrt{\Delta_K}$ in that group.

We first describe the case $M = q^k$. Given $1 + t\sqrt{\Delta_K}$ we compute the digits m_i in basis q of the unknown exponent m starting by the least significant one. At first look, the process is similar to Pohlig–Hellman method. However, contrary to this method, which would calculate at step i and exponentiation to the power q^{k-i} for $i = 1, \ldots, k-1$, our algorithm only uses an exponentiation to the power q at each step.

To obtain this more efficient method, we use the fact, noted at the end of the previous subsection, that $m_0 \equiv t \pmod{q}$. Then, let $i > 0$ and assume that the values of m_j, for $0 \leq j < i$, were already computed. We will consider $1 + t_i\sqrt{\Delta_K}$, defined as a representative of the element

$$(1 + t\sqrt{\Delta_K}) \cdot \left(1 + \sqrt{\Delta_K}\right)^{-\sum_{j=0}^{i-1} m_j q^j}.$$

The discrete logarithm of $1 + t_i\sqrt{\Delta_K}$ in basis $1 + \sqrt{\Delta_K}$ is then divisible by q^i. Then, as

$$\left(1 + \sqrt{\Delta_K}\right)^{m_i q^i + m_{i+1} q^{i+1} + \cdots} \equiv 1 + m_i q^i \sqrt{\Delta_K} \pmod{q^{i+1}},$$

we can compute m_i as $t_i / q^i \bmod q$. This process is formalized in Algorithm 13.

The case $M = 2^k$ is similar to the case $M = q^k$: the bits of the unknown exponent m are computed one at a time. It is even simpler to implement as the exponentiation by m_i is trivial to compute as m_i can only be 0 or 1.

Algorithm 13: $\mathsf{Solve}_{\mathsf{DL}}$ for CL_{q^k}

```
   Input: pp, $f_t$
   Result: $m$ such that $f_t = f^m$
   —
1  Compute $1 + t_0\sqrt{\Delta_K}$, a representative of $f_t$ in $\mathbb{G}_{q^k}$
2  $\alpha_0 := 1 + \sqrt{\Delta_K}$
3  for $i = 0$ to $k - 1$ do
4      $m_i := t_i/q^i \bmod q$
5      Compute $1 + t_{i+1}\sqrt{\Delta_K}$, a representative of $(1 + t_i\sqrt{\Delta_K}) \cdot \alpha_i^{-m_i}$
6      $\alpha_{i+1} := \alpha_i^q$
7  end
8  return $\sum_{i=0}^{k-1} m_i q^i$
```

6 Benchmarks of LHE Schemes Implemented in BICYCL

We have implemented several linearly homomorphic encryption schemes in our library `BICYCL`. First, the different linearly homomorphic encryption schemes based on class groups, which is the main purpose of this library, namely the schemes $\mathsf{CL\text{-}HSM}_{\mathsf{q^k}}$, $\mathsf{CL\text{-}HSM}_{\mathsf{2^k}}$ and their compact variants (as `CL_HSMqk` and `CL_HSM2k` classes).

For comparisons, we have also implemented Paillier ([47]) which is the most popular linearly homomorphic encryption scheme (`Paillier` class). As shown in Remark 1, the underlying LHE of Camenisch-Shoup encryption ([13]), based on the same security assumption than Paillier, can be viewed as an instantiation of our generic construction of Section 3. Moreover, several optimisations detailed in Subsection 5.2 for computing exponentiations can be applied to Camenisch-Shoup, as encryption also uses fixed-basis exponentiations, whereas this is not possible for Paillier. For fair comparison, we have therefore also implemented this scheme in `BICYCL` (`CamenischShoup` class).

In this section, we provide timings for $\mathsf{CL\text{-}HSM}_{\mathsf{q^k}}$, $\mathsf{CL\text{-}HSM}_{\mathsf{2^k}}$ and compare to Paillier and Camenisch-Shoup. All timings are performed on a standard laptop (Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz). The results are averages over 100 operations.

Parameters sizes Table 2 gives the sizes of fundamental discriminants and RSA moduli for various security levels that we used in `BICYCL` (available in the `RSA_modulus_bitsize` and `discriminant_bitsize` methods of the `SecLevel` class). These sizes come from the estimations done in [6], commonly used for implementation of class group based cryptography. Note that in particular, the discriminant Δ_K for $\mathsf{CL\text{-}HSM}_{\mathsf{2^k}}$ is defined as $-8N$ with N following the first column. The discriminants for $\mathsf{CL\text{-}HSM}_{\mathsf{q^k}}$ are given by the second column. They are smaller than RSA moduli as the best algorithms to compute discrete logarithms in class groups are in $L_{1/2}$ instead of $L_{1/3}$ for factoring integers.

Security level, λ	Size of RSA modulus N, $\eta'(\lambda)$	Size of Δ_K for $\mathsf{CL\text{-}HSM}_{\mathsf{q}^{\mathsf{k}}}$, $\eta(\lambda)$
112	2048 bits	1348 bits
128	3072 bits	1827 bits
192	7680 bits	3598 bits
256	15360 bits	5971 bits

Table 2. Size in bits of N and Δ_K for each security level.

Comparisons of LHE schemes We then give in Table 3 a comparison between four cryptosystems. For $\mathsf{CL\text{-}HSM}_{\mathsf{q}^{\mathsf{k}}}$, we choose to give timings for the case $k = 1$, therefore denoted $\mathsf{CL\text{-}HSM}_{\mathsf{q}}$, and q of bitsize twice the security parameter. This choice is driven by applications where the linearly encryption scheme is used to encrypt integers modulo the order of an elliptic curve, like in [14]. For $\mathsf{CL\text{-}HSM}_{\mathsf{2}^{\mathsf{k}}}$, we use $k = 64$ independently of the security level, as the main purpose of this scheme is to provide a threshold LHE homomorphic modulo 2^k compatible with applications to multi-party computations that closely match data manipulated by a CPU.

This table shows substantial improvements for the schemes based on class groups compared to previous implementations. Thanks to our many improvements, at the level of group operations and at the level of exponentiations, our experiments show that the gain we have on the size of the parameters allows to *compensate the additional cost of quadratic forms arithmetic* compared to traditional arithmetic in $\mathbf{Z}/N\mathbf{Z}$.

For instance, when comparing $\mathsf{CL\text{-}HSM}_{\mathsf{q}}$'s encryption with Paillier's encryption, $\mathsf{CL\text{-}HSM}_{\mathsf{q}}$ is faster for *all* security levels. This is in contrast with previous implementations like [14] whose implementation was only faster than Paillier's encryption for large security parameters. In details, $\mathsf{CL\text{-}HSM}_{\mathsf{q}}$'s encryption is 1.5 times faster for 112-bits security, 2.7 times faster for 128-bits security, 7 times faster for 192-bits security and 13.8 times faster for 256 bits security. Note that two threads are used to perform the two exponentiations in parallel during encryption for $\mathsf{CL\text{-}HSM}_{\mathsf{q}^{\mathsf{k}}}$, $\mathsf{CL\text{-}HSM}_{\mathsf{2}^{\mathsf{k}}}$ and Camenisch-Shoup's encryption. This is useless for Paillier's encryption. Note that our implementation of Paillier is faster than the one from Relic [2], essentially because Relic uses a generic exponentiation to perform the exponentiation of $1 + N$ modulo N^2 whereas it is indeed virtually free.

Decryption of $\mathsf{CL\text{-}HSM}_{\mathsf{q}}$ is faster than Paillier's starting from 128 bits of security. Overall this shows that $\mathsf{CL\text{-}HSM}_{\mathsf{q}}$ is highly competitive compared to Paillier. The optimizations for the exponentiations in Camenisch-Shoup make also this scheme faster than Paillier. However this is at the cost of very large ciphertexts (Camenisch-Shoup needs two elements of $\mathbf{Z}/N^2\mathbf{Z}$ against one for Paillier).

Concerning setup and key generation, once the class group is computed, key generation in $\mathsf{CL\text{-}HSM}_{\mathsf{q}}$ is the fastest, since it consists of a single exponentiation in the class group, whereas Paillier's key generation needs to generate large

Sec. level		$\mathsf{CL\text{-}HSM_q}$	Paillier	Camenisch–Shoup	$\mathsf{CL\text{-}HSM_{2^k}}$
112	ciphertext	**2694** bits	4096 bits	8192 bits	3272 bits
	setup	0.300 s	-	0.051 s	0.571 s
	keygen	**0.011** s	0.039 s	0.012 s	0.019 s
	encrypt	4.39 ms	6.57 ms	**3.00** ms	7.78 ms
	decrypt	9.70 ms	**6.56** ms	7.17 ms	17.7 ms
128	ciphertext	**3509** bits	6144 bits	12288 bits	4808 bits
	setup	0.586 s	-	0.178 s	1.78 s
	keygen	**0.019** s	0.121 s	0.036s	0.044 s
	encrypt	**7.68** ms	20.9 ms	8.62 ms	17.4 ms
	decrypt	**17.8** ms	20.8 ms	21.8 ms	40.1 ms
192	ciphertext	**6549** bits	15360 bits	30720 bits	11720 bits
	setup	7.65 s	-	5.55 s	27.3 s
	keygen	**0.072** s	5.63 s	0.374 s	0.319 s
	encrypt	**28.2** ms	198 ms	108 ms	128 ms
	decrypt	**67.3** ms	199 ms	205 ms	300 ms
256	ciphertext	**10493** bits	30720 bits	61440 bits	23240 bits
	setup	72.0 s	-	53.4 s	263 s
	keygen	**0.216** s	79.6 s	2.25 s	1.77 s
	encrypt	**84.9** ms	1169 ms	624 ms	691 ms
	decrypt	**202** ms	1168 ms	1262 ms	1600 ms

Table 3. Comparison between $\mathsf{CL\text{-}HSM_q}$, Paillier and Camenisch-Shoup and $\mathsf{CL\text{-}HSM_{2^k}}$ where the bitsize of q is twice the security level and $k = 64$

primes, and Camenisch-Shoup's consists of an exponentiation in a group that is significantly larger.

The $\mathsf{CL\text{-}HSM}_{2^k}$ scheme gives encryption timings that are globally intermediate between Camenisch-Shoup and Paillier and a worse decryption time. This is due to the fact that security is based on factoring, so we do not benefit from a small discriminant. However, timings are in the same order of magnitude compared to the other factoring based schemes.

Concerning ciphertext sizes, for 112-bits security and a q of 224 bits, ciphertexts for $\mathsf{CL\text{-}HSM}_\mathsf{q}$ are 1.5 (resp. 3) times shorter than Paillier's (resp. Camenisch-Shoup's). For 192-bits of security, the difference is of course much higher: ciphertexts for $\mathsf{CL\text{-}HSM}_\mathsf{q}$ are 2.3 (resp. 4.7) times shorter than Paillier's (resp. Camenisch-Shoup's). Note that the elements that are transmitted are compressed using the technique from [30] (see Section 2) and the time of encryption (resp. decryption) includes compression (resp. decompression) which corresponds to less than 1% of the time.

Compact variants As shown in Table 1, with the compact variants we gain $3/2\log(q)$ bits for $\mathsf{CL\text{-}HSM}_\mathsf{q}$ and $3/2\,k$ bits for $\mathsf{CL\text{-}HSM}_{2^k}$ on the size of the ciphertexts. Concretely, for the choices made in Table 3, this means that $\mathsf{CL}_\mathsf{C}\text{-}\mathsf{HSM}_\mathsf{q}$ has ciphertexts of bitsizes 2358, 3125 or 5973 depending of the security level. For $\mathsf{CL}_\mathsf{C}\text{-}\mathsf{HSM}_{2^k}$, we get ciphertexts of 3174, 4710 or 11622 bits. This reduction on the size comes with an additional cost for computing the lift during encryption and decryption (essentially an exponentiation to the power M). Our benchmark shows that depending on the security level $\mathsf{CL}_\mathsf{C}\text{-}\mathsf{HSM}_\mathsf{q}$ adds 35% to 55% more time for encryption and 4% to 15% for decryption. So for applications where bandwidth is crucial but encryption time is less sensitive, this variant is adequate. For $\mathsf{CL}_\mathsf{C}\text{-}\mathsf{HSM}_{2^k}$ the overheads are smaller as $M = 2^k$ is also smaller: around 6% to 9% more time for encryption and 3% to 6% for decryption.

Comparisons with previous implementations for CL We run `BICYCL` on the same machine used for the benchmark of [19] (which describe the original DDH variant of CL). Our experiments shows that the $\mathsf{CL\text{-}HSM}_\mathsf{q}$ implementation of `BICYCL` improves encryption time by a factor 20, and decryption by a factor 5. Compared to the timings announced in [27] (still for the original DDH variant of CL) that were run on a machine with a slightly more powerful CPU, we get an improvement of factor 7 for encryption and 2 for decryption.

For $\mathsf{CL\text{-}HSM}_{2^k}$, running `BICYCL` on the same machine used for the benchmark of [21] shows that we improve the encryption time by a factor 9, and decryption by a factor 4.

Overall, these results show that we obtain improvements not only because of the improvement of the basic arithmetic has shown in Figure 3, but especially thanks to our tailored implementation of the exponentiations.

References

1. D. Abram, I. Damgård, C. Orlandi, and P. Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In *CRYPTO 2022*. Springer-Verlag, 2022.
2. D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.
3. T. Attema, I. Cascudo, R. Cramer, I. B. Damgård, and D. Escudero. Vector commitments over rings and compressed σ-protocols. Cryptology ePrint Archive, Report 2022/181, 2022. `https://eprint.iacr.org/2022/181`.
4. W. Beullens, T. Kleinjung, and F. Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 227–247. Springer, Heidelberg, Dec. 2019.
5. J.-F. Biasse. Improvements in the computation of ideal class groups of imaginary quadratic number fields. *Advances in Mathematics of Communications*, 4(2):141–154, 2010.
6. J.-F. Biasse, M. J. Jacobson, and A. K. Silvester. Security estimates for quadratic field based cryptosystems. In R. Steinfeld and P. Hawkes, editors, *ACISP 10*, volume 6168 of *LNCS*, pages 233–247. Springer, Heidelberg, July 2010.
7. D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, Aug. 2019.
8. J. Buchmann, C. Thiel, and H. Williams. Short representation of quadratic integers. In W. Bosma and A. van der Poorten, editors, *Computational Algebra and Number Theory*, pages 159–185, Dordrecht, 1995. Springer Netherlands.
9. J. Buchmann and U. Vollmer. *Binary Quadratic Forms: An Algorithmic Approach*. Algorithms and Computation in Mathematics. Springer Berlin Heidelberg, 2007.
10. J. Buchmann and H. C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1(2):107–118, June 1988.
11. J. Buchmann and H. C. Williams. A key exchange system based on real quadratic fields. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 335–343. Springer, Heidelberg, Aug. 1990.
12. B. Bünz, B. Fisch, and A. Szepieniec. Transparent SNARKs from DARK compilers. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020.
13. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Heidelberg, Aug. 2003.
14. G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 191–221. Springer, Heidelberg, Aug. 2019.
15. G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. Bandwidth-efficient threshold EC-DSA. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 266–296. Springer, Heidelberg, May 2020.

16. G. Castagnos, L. Imbert, and F. Laguillaumie. Encryption switching protocols revisited: Switching modulo p. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 255–287. Springer, Heidelberg, Aug. 2017.
17. G. Castagnos, A. Joux, F. Laguillaumie, and P. Q. Nguyen. Factoring pq^2 with quadratic forms: Nice cryptanalyses. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 469–486. Springer, Heidelberg, Dec. 2009.
18. G. Castagnos and F. Laguillaumie. On the security of cryptosystems with quadratic decryption: The nicest cryptanalysis. In A. Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 260–277. Springer, Heidelberg, Apr. 2009.
19. G. Castagnos and F. Laguillaumie. Linearly homomorphic encryption from DDH. In K. Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 487–505. Springer, Heidelberg, Apr. 2015.
20. G. Castagnos, F. Laguillaumie, and I. Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 733–764. Springer, Heidelberg, Dec. 2018.
21. G. Castagnos, F. Laguillaumie, and I. Tucker. Threshold linearly homomorphic encryption on $\mathbf{Z}/2^k\mathbf{Z}$. Cryptology ePrint Archive, Paper 2022/1143, 2022. `https://eprint.iacr.org/2022/1143`, to appear at *ASIACRYPT 2022*.
22. P. Chaidos and G. Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 193–221. Springer, Heidelberg, Apr. / May 2018.
23. CHIA. Chia verifiable delay function competition. `https://medium.com/@chia.net/chia-vdf-competition-guide-5382e1f4bd39`, 2018.
24. G. Couteau, D. Goudarzi, M. Klooß, and M. Reichle. Sharp: Short relaxed range proofs. Cryptology ePrint Archive, Paper 2022/1153, 2022. `https://eprint.iacr.org/2022/1153`.
25. G. Couteau, M. Klooß, H. Lin, and M. Reichle. Efficient range proofs with transparent setup from bounded integer commitments. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 247–277. Springer, Heidelberg, Oct. 2021.
26. D. Cox. *Primes of the Form $x^2 + ny^2$: Fermat, Class Field Theory, and Complex Multiplication.* Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts. Wiley, 2014.
27. P. Das, M. J. Jacobson Jr., and R. Scheidler. Improved efficiency of a linearly homomorphic cryptosystem. In *Codes, Cryptology and Information Security*, pages 349–368. Springer International Publishing, 2019.
28. Y. Deng, S. Ma, X. Zhang, H. Wang, X. Song, and X. Xie. Promise Σ-protocol: How to construct efficient threshold ECDSA from encryptions based on class groups. In M. Tibouchi and H. Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 557–586. Springer, Heidelberg, Dec. 2021.
29. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
30. S. Dobson, S. Galbraith, and B. Smith. Trustless unknown-order groups. *Mathematical Cryptology*, 1(1):1–15, 2021.
31. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, Aug. 1984.

32. N. Glaeser, M. Maffei, G. Malavolta, P. Moreno-Sanchez, E. Tairi, and S. A. Thyagarajan. Foundations of coin mixing services. Cryptology ePrint Archive, Paper 2022/942, 2022. `https://eprint.iacr.org/2022/942`.
33. GMP. *The GNU Multiple Precision Arithmetic Library.* `https://gmplib.org/`.
34. J. L. Hafner and K. S. McCurley. A rigorous subexponential algorithm for computation of class groups. *Journal of the American Mathematical Society*, 2(4):837–850, 1989.
35. W. Hart, F. Johansson, and S. Pancratz. FLINT: Fast Library for Number Theory, 2022. Version 2.9.0, `http://flintlib.org`.
36. D. Hühnlein. Efficient implementation of cryptosystems based on non-maximal imaginary quadratic orders. In H. M. Heys and C. M. Adams, editors, *SAC 1999*, volume 1758 of *LNCS*, pages 147–162. Springer, Heidelberg, Aug. 1999.
37. D. Hühnlein, M. J. Jacobson Jr., S. Paulus, and T. Takagi. A cryptosystem based on non-maximal imaginary quadratic orders with fast decryption. In K. Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 294–307. Springer, Heidelberg, May / June 1998.
38. D. Hühnlein, M. J. Jacobson Jr., and D. Weber. Towards practical non-interactive public key cryptosystems using non-maximal imaginary quadratic orders. In D. R. Stinson and S. E. Tavares, editors, *SAC 2000*, volume 2012 of *LNCS*, pages 275–287. Springer, Heidelberg, Aug. 2001.
39. M. J. Jacobson Jr. Computing discrete logarithms in quadratic orders. *Journal of Cryptology*, 13(4):473–492, Sept. 2000.
40. B. King. wNAF*, an efficient left-to-right signed digit recoding algorithm. In S. M. Bellovin, R. Gennaro, A. D. Keromytis, and M. Yung, editors, *ACNS 08*, volume 5037 of *LNCS*, pages 429–445. Springer, Heidelberg, June 2008.
41. T. Kleinjung. Quadratic sieving. *Mathematics of Computation*, 85(300):1861–1873, 2016.
42. J. Lagarias. Worst-case complexity bounds for algorithms in the theory of integral quadratic forms. *Journal of Algorithms*, 1(2):142 – 186, 1980.
43. R. W. F. Lai and G. Malavolta. Subvector commitments with application to succinct arguments. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 530–560. Springer, Heidelberg, Aug. 2019.
44. H. Lipmaa. Secure accumulators from euclidean rings without trusted setup. In F. Bao, P. Samarati, and J. Zhou, editors, *ACNS 12*, volume 7341 of *LNCS*, pages 224–240. Springer, Heidelberg, June 2012.
45. K. S. McCurley. Cryptographic key distribution and computation in class groups. In R. A. Molin, editor, *Proc. NATO Advanced Study Inst. on Number Theory and Applications, Banff, 1988*, Boston, 1989. Kluwer.
46. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, Florida, 1996.
47. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.
48. PARI Group, Univ. Bordeaux. *PARI/GP version* `2.15.0`, 2022. available from `http://pari.math.u-bordeaux.fr/`.
49. M. Sayles. libqform. `https://github.com/maxwellsayles/libqform`, 2014.
50. D. Shanks. On Gauss and composition I, II. In *Proc. NATO ASI on Number Theory and Applications*, pages 163–179. Kluwer Academic Press, 1989.
51. J. A. Solinas. Low-weight binary representations for pairs of integers. Research report CORR 2001-41, Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada, 2001.

52. E. G. Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 71(7):806–808, 1964.
53. S. A. K. Thyagarajan, G. Castagnos, F. Laguillaumie, and G. Malavolta. Efficient CCA timed commitments in class groups. In G. Vigna and E. Shi, editors, *ACM CCS 2021*, pages 2663–2684. ACM Press, Nov. 2021.
54. I. Tucker. *Functional encryption and distributed signatures based on projective hash functions, the benefit of class groups*. PhD thesis, Université de Lyon, 2020.
55. B. Wesolowski. Efficient verifiable delay functions. *Journal of Cryptology*, 33(4):2113–2147, Oct. 2020.
56. T. H. Yuen, H. Cui, and X. Xie. Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In J. Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 481–511. Springer, Heidelberg, May 2021.
57. Zengo. Class: Rust library for building iqc. `https://github.com/ZenGo-X/class`.