Efficient Extended GCD and Class Groups from Secure Integer Arithmetic

Efficient Extended GCD and Class Groups from Secure Integer Arithmetic

Berry Schoenmakers and Toon Segers(✉)

Department Math & CS, TU Eindhoven, Eindhoven, The Netherlands
{l.a.m.schoenmakers,a.j.m.segers}@tue.nl

Abstract. In this paper we first present an efficient protocol for the secure computation of the extended greatest common divisor, assuming basic secure integer arithmetic common to many MPC frameworks. The protocol is based on Bernstein and Yang's constant-time 2-adic algorithm, which we adapt to work purely over the integers. This yields a much better approach for the MPC setting, but raises a new concern about the growth of the Bézout coefficients. By a careful analysis we are able to prove that the Bézout coefficients in our protocol will never exceed $3\max(a, b)$ in absolute value for inputs a and b. Next, we present efficient protocols for implementing class groups of imaginary quadratic number fields in the MPC setting. We start from Shanks' original algorithms for the efficient composition of binary quadratic forms and combine these with our particular adaptation of a forms reduction algorithm due to Agarwal and Frandsen. We will formulate this result in terms of secure groups, which are introduced as oblivious data structures implementing finite groups in a privacy-preserving manner. Our results show how class group operations can be run efficiently between multiple parties operating jointly on secret-shared group elements. We have integrated secure class groups in MPyC along with other instances of secure groups such as Schnorr groups and elliptic curves.

1 Introduction

Recall that a finite group consists of a finite set $\mathbb{G}$ of group elements (including the identity element) together with an associative group operation $*$ and the corresponding inverse operation. In this paper we will formulate our protocols implementing class groups in an MPC setting in terms of secure groups. A secure group is introduced as a cryptographic scheme for a group $(\mathbb{G}, *)$ supporting a secret-shared representation of group elements from $\mathbb{G}$ and an oblivious implementation of the group operation $*$. We note that Bar-Ilan and Beaver [BIB89, Lemma 6] already considered a generic protocol for secure inversion of group elements, as a natural generalization of secure matrix multiplication. A secure group will cover further tasks such as protocols for group inversion, equality tests of group elements, and random sampling, all operating on secret-shared values.

Our main goal is to support secure class groups. We focus on ideal class groups of imaginary quadratic fields, which have been studied extensively for

S. Dolev et al. (Eds.): CSCML 2023, LNCS 13914, pp. 32–48, 2023.
https://doi.org/10.1007/978-3-031-34671-2_3

use in cryptography by Buchmann et al. (see [BV07] and references therein) and recently got renewed interest (see, e.g., [Wes19, BHR+21, DGS22]). For the implementation of the class group operation, we present efficient protocols for the composition and reduction of binary quadratic forms.

A key step in our protocol for the composition of forms is the secure computation of the extended greatest common divisor. We start from the constant-time algorithms proposed by Bernstein and Yang [BY19]. Their approach relies on the use of 2-adic arithmetic but we will work purely over the integers for an efficient solution in the MPC setting. The core of the protocol centers around Bernstein and Yang's divsteps2 algorithm which takes a fixed number of roughly 3ℓ iterations when run on numbers of bit length at most ℓ. The work for each iteration is dominated by secure comparisons with numbers of bit length at most $\log_2 \ell$, requiring $O(\log \ell)$ secure multiplications each. We develop a novel analysis showing that the Bézout coefficients are bounded above by $3 \max(a, b)$ in absolute value when computing the extended greatest common divisor of numbers a and b. The number of $O(\ell \log \ell)$ secure multiplications compares favorably to, e.g., a basic binary gcd algorithm [MvOV96, Algorithm 14.61] which requires $O(\ell^2)$ secure multiplications. Other binary gcd algorithms such as Bojanczyk and Brent [BB87] (and related algorithms as considered by [BY19]) as well as the algorithm attributed to Penk [Knu69, Exercise 4.5.2.39] also lead to $O(\ell^2)$ secure multiplications due to the use of either full-size ℓ-bit secure comparisons or inner loops requiring $O(\ell)$ secure multiplications in the worst case.

To obtain an efficient protocol for the reduction of forms we start from the algorithm proposed by Agarwal and Frandsen [AF06]. A key property of their approach is that the use of full integer divisions is avoided in the main loop of the form reduction algorithm. We show how to adapt their algorithm to the MPC setting such that the work for each iteration of the main loop is limited to a small constant number of secure comparisons and operations of similar complexity requiring $O(\ell)$ secure multiplications each. For discriminant $\Delta < 0$ of bit length ℓ and forms with coefficients all of bit length at most ℓ, we show that $\ell/2$ iterations suffice for the reduction, which leads to $O(\ell^2)$ secure multiplications overall.

We conclude the paper with a brief discussion of the support for secure groups in MPyC and applications in threshold cryptography and verifiable MPC. Apart from the implementation of class groups, MPyC also supports various other secure groups such as elliptic curves and quadratic residues in a multiparty setting. We use these secure groups to implement noninteractive zero-knowledge proofs where the role of the prover is distributed between multiple parties.

Roadmap. The paper is organized as follows. Section 2 introduces preliminaries for secure multiparty computation. Section 3 presents the protocol for the extended greatest common divisor. Section 4 introduces a cryptographic scheme for secure groups. Section 5 presents generic protocols related to secure groups for the following tasks: conditional (if-else), random sampling, inversion and exponentiation. Section 6 presents specific protocols for secure class groups. Section 7 concludes with remarks about implementation and applications.

2 MPC Setting

We consider an MPC setting with m parties tolerating a dishonest minority of up to t passively corrupt parties, $0 \leq t < m/2$. The basic protocols for secure addition and multiplication over a finite field rely on Shamir secret sharing [BGW88,GRR98]. We write $[\![a]\!]$ (or sometimes $[\![a]\!]_p$) for a Shamir secret sharing of a finite field element $a \in \mathbb{F}_p$. For the scope of this paper, it suffices to consider finite fields of prime order p, where we assume $p > m$. For secure integer arithmetic, we use the common representation for signed integers a in a bounded range, e.g., with $\mathrm{len}(a) \leq \ell \ll p$, where $\mathrm{len}(a) = \lceil \log_2(|a| + 1) \rceil$. The amount $\mathrm{len}(p) - \ell$ of excess bits is often referred to as "headroom".

To operate on secret-shared integer values we use common protocols for secure integer arithmetic as building blocks. E.g., we let $[\![r]\!] \leftarrow \mathsf{random}(\mathbb{F}_p)$ denote the generation of a uniformly random $r \in_R \mathbb{F}_p$ and $[\![a_0]\!], \ldots, [\![a_{\ell-1}]\!] \leftarrow \mathsf{bits}([\![a]\!])$ denote the bit decomposition of a. Also $[\![d]\!] \leftarrow \mathsf{if\_else}([\![c]\!], [\![a]\!], [\![b]\!])$ denotes the secure conditional, where $d = c(a-b)+b$. A protocol that generates a vector of j secret-shared bits in $\mathbb{F}_p \cap \{0,1\}$ is denoted by $\mathsf{random\ bits}(\mathbb{F}_p, j)$. See [Hoo12,Tof07,DFK+06] for these and other common MPC protocols, which we will use as black boxes.

We also use secure integer division $([\![q]\!], [\![r]\!]) \leftarrow \mathsf{divmod}([\![a]\!], [\![b]\!])$ with $a = bq + r$ and $0 \leq r < b$ as a primitive. Our implementation is based on the Newton-Raphson method [ACS02,CS10]. Finally, we assume an efficient protocol for securely determining the bit length $[\![\mathrm{len}(a)]\!]$. See Sect. 7.

3 Secure Extended GCD

This section presents a new protocol for computing the extended greatest common divisor (xgcd) of secret-shared integers. In the next section we use this protocol for the group operation for class groups of imaginary quadratic fields in MPC.

We first present Algorithm 1, which can be viewed as an alternative to the divsteps2 algorithm by Bernstein and Yang [BY19], on which it is based, but without the use of any 2-adic arithmetic. Algorithm 1 works purely over the integers. To compute, for instance, a modular inverse we do not need any (potentially costly) pre- or postprocessing (in contrast to Bernstein and Yang's algorithm recip2), which would complicate the conversion to an MPC protocol. Hence, our xgcd algorithm may also be of independent interest for other settings in which the use of 2-adic arithmetic is not straightforward. (Using precomputation for modular inverses can be beneficial, particularly in applications that reuse the modulus. In settings where inputs are not reusable, the precomputation cannot be reused. This is the case in our application of the xgcd to compute class group operations.)

As starting point we take the constant-time extended gcd algorithm divsteps2 by Bernstein and Yang [BY19, Figure 10.1]. Our protocol xgcd, see Protocol 1, retains the algorithmic flow of their divsteps2 algorithm, which is controlled by the following step function [BY19, Section 8]:

Algorithm 1. $\mathsf{xgcd}(n, a, b)$ $\quad n, a, b \in \mathbb{N}$, a odd

$\delta, f, g, u, v, q, r \leftarrow 1, a, b, 1, 0, 0, 1$ ▷ $f = ua + vb$, $g = qa + rb$
for $i = 1$ **to** n **do**
  $g_0 \leftarrow g \bmod 2$
  if $\delta > 0$ and $g_0 = 1$ **then**
    $\delta, f, g, u, v, q, r \leftarrow -\delta, g, -f, q, r, -u, -v$
  if $g_0 = 1$ **then**
    $g, q, r \leftarrow g + f, q + u, r + v$
  if $r \bmod 2 = 1$ **then**
    $q, r \leftarrow q - b, r + a$
  $\delta, g, q, r \leftarrow \delta + 1, g/2, q/2, r/2$
if $f < 0$ **then**
  $f, u, v \leftarrow -f, -u, -v$
return f, u, v

$$\mathsf{divstep}(\delta, f, g) = \begin{cases} (1 - \delta, g, (g - f)/2), & \text{if } \delta > 0 \text{ and } g \text{ is odd}, \\ (1 + \delta, f, (g + (g \bmod 2)f)/2), & \text{otherwise.} \end{cases} \tag{1}$$

Throughout, variable f is ensured to be an odd integer, and therefore the divisions by 2 are without remainder in both cases of the step function. Bernstein and Yang argue that this step function compares favorably with alternatives from the literature, e.g., the Brent–Kung step function [BK85] and the Stehlé–Zimmermann step function [SZ04]. The computational overhead of function divstep is small, and the required number of iterations n as a function of the bit lengths of the inputs a and b compares favorably with the alternatives. Concretely, with $(\delta_n, f_n, g_n) = \mathsf{divstep}^n(1, a, b)$ for odd a, Bernstein and Yang prove that $f_n = \gcd(a, b)$ and $g_n = 0$ holds for $n = \mathsf{iterations}(\ell)$, where $\ell = \max(\mathrm{len}(a), \mathrm{len}(b))$ and

$$\mathsf{iterations}(d) = \begin{cases} \lfloor (49d + 80)/17 \rfloor, & \text{if } d < 46, \\ \lfloor (49d + 57)/17 \rfloor, & \text{otherwise.} \end{cases} \tag{2}$$

Hence, $\mathsf{iterations}(\ell) \approx 3\ell$.

The first major change compared to Bernstein and Yang's divsteps2 algorithm is that we entirely drop the use of truncation for f and g. In our MPC setting, f and g are secret-shared values (over a prime field of large order) and therefore limiting the sizes of f and g is not useful. The second major change is that we will avoid the use of 2-adic arithmetic entirely, by ensuring that the Bézout coefficients will remain integral throughout all iterations. Concretely, this means that we will make sure that coefficients q and r are even before the division by 2 at the end of each iteration: if q and r are odd, we use $q - b$ and $r + a$ instead, which will then be even, because a is odd by assumption. We thus obtain Algorithm 1 as an alternative to Bernstein-Yang's constant-time algorithm.

Protocol 1. $\mathsf{xgcd}(n, [\![a]\!], [\![b]\!])$ $\qquad n, a, b \in \mathbb{N}$, a odd

$[\![\delta]\!], [\![f]\!], [\![g]\!], [\![v]\!], [\![r]\!] \leftarrow 1, [\![a]\!], [\![b]\!], 0, 1$
for $i = 1$ **to** n **do**
  $[\![g_0]\!] \leftarrow [\![g \bmod 2]\!]$
  if $[\![\delta]\!] > 0$ and $[\![g_0]\!] = 1$ **then**
    $[\![\delta]\!], [\![f]\!], [\![g]\!], [\![v]\!], [\![r]\!] \leftarrow -[\![\delta]\!], [\![g]\!], -[\![f]\!], [\![r]\!], -[\![v]\!]$
  if $[\![g_0]\!] = 1$ **then**
    $[\![g]\!], [\![r]\!] \leftarrow [\![g]\!] + [\![f]\!], [\![r]\!] + [\![v]\!]$
  if $[\![r \bmod 2]\!] = 1$ **then**
    $[\![r]\!] \leftarrow [\![r]\!] + [\![a]\!]$
  $[\![\delta]\!], [\![g]\!], [\![r]\!] \leftarrow [\![\delta]\!] + 1, [\![g]\!]/2, [\![r]\!]/2$
if $[\![f]\!] < 0$ **then**
  $[\![f]\!], [\![v]\!] \leftarrow -[\![f]\!], -[\![v]\!]$
$[\![u]\!] \leftarrow ([\![f]\!] - [\![v]\!] * [\![b]\!])/[\![a]\!]$
return $[\![f]\!], [\![u]\!], [\![v]\!]$ $\qquad \triangleright\ f = ua + vb = \gcd(a, b)$

We turn Algorithm 1 into a secure protocol operating on secret-shared values as follows. We drop variables q and u from the main loop, and instead set $u = (f - vb)/a$ at the end. Overall, this saves $3n$ secure multiplications for q and n secure multiplications for u, that are otherwise needed to implement the if-then statements obliviously. See Protocol 1 for the result.

All operations on the remaining variables δ, g, f, v, r are done securely. The secure computations of $[\![g \bmod 2]\!]$ and $[\![r \bmod 2]\!]$ are done by using a secure protocol for computing the least significant bit. The secure computation of $[\![\delta > 0]\!]$ is the most expensive part of each iteration. However, by taking into account that δ is bounded above by $i + 1$, we can further reduce the cost of this secure comparison.

The result is a secure protocol with a single loop of $n \approx 3\ell$ iterations, where ℓ denotes the maximum bit length for a and b. Per iteration, the work is proportional to $O(\log \ell)$ secure multiplications, as the comparison for $\log \ell$ bit numbers determines the complexity of the adapted divstep. The resulting $O(\ell \log \ell)$ compares favorably to, e.g., the binary extended gcd from [MvOV96, Algorithm 14.61], which would require $O(\ell^2)$ secure multiplications.

For the general case, where a is not known to be odd, we use an auxiliary protocol to securely count the number of trailing zeros of a given secret-shared integer. We have designed and implemented a novel approach requiring $O(\ell)$ secure multiplications in $O(1)$ rounds for this task. Moreover, we have designed and implemented protocols for the secure modular inverse of a modulo b (provided $\gcd(a, b) = 1$), and for computing $\gcd(a, b)$ and $\mathrm{lcm}(a, b)$ securely. See Sect. 7.

Bounding the Bezout Coefficients

We conclude this section with a result on the bounds for the Bézout coefficients computed by our xgcd algorithm (and protocol). The novelty of this result is

due to the fact that the bounds are explicit (avoiding big-O notation), while the xgcd algorithm avoids full-sized comparisons to control the size of intermediate variables. To this end, we analyse Algorithm 1 and show in Theorem 1 below that the Bézout coefficients are bounded by $3a$.

For the analysis, we partition an execution of Algorithm 1 into k consecutive runs as follows. A new run starts with each assignment to variable v: the first run starts with the initialization $v \leftarrow 0$ in the first line and each next run starts with an update $v \leftarrow r$ in line 5.

By v_j we denote the value assigned to v at the start of the jth run, $1 \le j \le k$. Hence, $v_1 = 0$ and v_k will be the final value of v. We define $v_0 = -1$.

Let e_j denote the length of the jth run, which is defined as the number of times δ is incremented (in line 10) during the run. Hence $\sum_{j=1}^{k} e_j = n$. Note that $e_1 = 0$ is possible, but $e_j \ge 1$ for $2 \le j \le k$.

Let δ_j be the value of δ at the end of the jth run. We define $\delta_0 = -1$. Then $\delta_j \ge 1$ for $1 \le j \le k-1$. Note that $\delta_k \le 0$ is possible, but this will be irrelevant for the analysis. Also, $e_j = \delta_j + \delta_{j-1}$ for $1 \le j \le k$.

We want to show that all v_j are bounded by induction on j.

At the start of the jth run, r is set to $-v_{j-1}$. At the end of each loop iteration the value of r, which is ensured to be even at that point, is halved. This will limit the growth of $|r|$. On the other hand, $|r|$ may grow because of the additions of v and/or a. Since $\delta > 0$ precisely during the last $\delta_j - 1$ iterations of the jth run, however, it follows that g must then be even, and therefore $|r|$ can only grow because of the addition of a.

To analyze the extreme values for r at the end of each run, we introduce the following three auxiliary functions:

$$\begin{aligned}
\phi(r, v, N) &= (r + v(2^N - 1))/2^N, \\
\psi(r, v, \delta, \delta') &= \phi(\phi(r, \min(v, 0), \delta + 1), 0, \delta' - 1), \\
\Psi(r, v, \delta, \delta') &= \phi(\phi(r, \max(v, 0) + a, \delta + 1), a, \delta' - 1).
\end{aligned}$$

The relevant monotonicity properties of these functions are stated as follows.

Lemma 1. *Functions $\phi(r, v, N)$, $\psi(r, v, \delta, \delta')$, and $\Psi(r, v, \delta, \delta')$ are increasing in r and nondecreasing in v.*

The next two lemmas establish the basic bounds for v_j, which follow almost directly from the way we have defined ψ and Ψ.

Lemma 2. *We have $v_0 = -1$, $v_1 = 0$, and for $2 \le j \le k$:*

$$\psi(-v_{j-2}, v_{j-1}, \delta_{j-1}, \delta_j) \le v_j \le \Psi(-v_{j-2}, v_{j-1}, \delta_{j-1}, \delta_j).$$

Lemma 3. $v_j \le \Psi(-v_{j-2}, \Psi(-v_{j-3}, v_{j-2}, \delta_{j-2}, \delta_{j-1}), \delta_{j-1}, \delta_j)$ *for $j \ge 3$.*

Proof. From Lemma 2, we know that $v_j \le \Psi(-v_{j-2}, v_{j-1}, \delta_{j-1}, \delta_j)$. Since function $\Psi(r, v, \delta, \delta')$ is nondecreasing in v, we get the claimed bound for v_j because we also have $v_{j-1} \le \Psi(-v_{j-3}, v_{j-2}, \delta_{j-2}, \delta_{j-1})$ (using that $j - 1 \ge 2$). □

Finally, we prove our main result, establishing a nontrivial bound for the Bézout coefficients v_j.

Theorem 1. *$|v_j| \le 3a$ for $0 \le j \le k$.*

Proof. The proof is by induction on j. Since $v_0 = -1$ and $v_1 = 0$, the bound clearly holds for $j \le 1$ as $a \ge 1$. For v_2, we have the following bounds from Lemma 2:

$$\psi(1, 0, \delta_1, \delta_2) \le v_2 \le \Psi(1, 0, \delta_1, \delta_2).$$

Since $\psi(1, 0, \delta_1, \delta_2) \ge 0$ and $\Psi(1, 0, \delta_1, \delta_2) \le a$, the bound also holds for $j = 2$.

For $j \ge 3$, we first prove the lower bound $v_j \ge -3a$. From Lemma 2, we have

$$v_j \ge \psi(-v_{j-2}, v_{j-1}, \delta_{j-1}, \delta_j).$$

Since $\psi(r, v, \delta, \delta')$ is increasing in r and nondecreasing in v, we then get

$$v_j \ge \psi(-3a, -3a, \delta_{j-1}, \delta_j),$$

as $-v_{j-2} \ge -3a$ and $v_{j-1} \ge -3a$ follow from the induction hypothesis.

Hence, the lower bound for v_j holds, as

$$\psi(-3a, -3a, \delta_{j-1}, \delta_j) = \phi(-3a, 0, \delta_j - 1) \ge -3a.$$

Next we prove the upper bound $v_j \le 3a$. From Lemma 3, we have

$$v_j \le \Psi(-v_{j-2}, \Psi(-v_{j-3}, v_{j-2}, \delta_{j-2}, \delta_{j-1}), \delta_{j-1}, \delta_j).$$

Since $\Psi(r, v, \delta, \delta')$ is increasing in r and nondecreasing in v, we get

$$v_j \le \Psi(-v_{j-2}, \Psi(3a, v_{j-2}, \delta_{j-2}, \delta_{j-1}), \delta_{j-1}, \delta_j),$$

as $-v_{j-3} \le 3a$ on account of the induction hypothesis.

To complete the proof we distinguish the cases $v_{j-2} \le 0$ and $v_{j-2} > 0$.

If $v_{j-2} \le 0$, we see that

$$\Psi(\alpha a, v_{j-2}, \delta_{j-2}, \delta_{j-1}) = \phi(\phi(3a, a, \delta_{j-2} + 1), a, \delta_{j-1} - 1),$$

which is independent of v_{j-2} and is bounded above by $\phi(3a, a, 2) = 3a/2$.

Since $\Psi(r, v, \delta, \delta')$ is nondecreasing in v, we therefore have

$$v_j \le \Psi(-v_{j-2}, 3a/2, \delta_{j-1}, \delta_j).$$

And as $\Psi(r, v, \delta, \delta')$ is increasing in r, we conclude

$$v_j \le \Psi(3a, 3a/2, \delta_{j-1}, \delta_j),$$

as $-v_{j-2} \le 3a$ on account of the induction hypothesis. This leads to

$$v_j \le \phi(3a, 5a/2, 2) = (3a + 15a/2)/4 = 21a/8 < 3a.$$

If $v_{j-2} > 0$, we see that

$$\Psi(3a, v_{j-2}, \delta_{j-2}, \delta_{j-1}) = \phi(\phi(3a, v_{j-2} + a, \delta_{j-2} + 1), a, \delta_{j-1} - 1)$$

still depends on v_{j-2}. To prove the upper bound for v_j in this case, we therefore introduce:

$$f(v, \delta, \delta', \delta'') = \Psi(-v, \Psi(3a, v, \delta, \delta'), \delta', \delta''),$$

for $0 < v \leq 3a$, hence $\Psi(r, v, \delta, \delta') = \phi(\phi(r, v + a, \delta + 1), a, \delta' - 1)$.

For the derivative of f with respect to v we get, for $\delta, \delta', \delta'' \geq 1$:

$$\frac{\partial f}{\partial v} = \left(3 \cdot 2^{\delta+\delta'} - 2^{\delta'+1} - 2^{\delta+1} + 1\right) 2^{-\delta-2\delta'-\delta''} > 0,$$

hence we set $v = 3a$ at its maximal value.

Finally, we have that $g(\delta, \delta', \delta'') = f(3a, \delta, \delta', \delta'')$ is increasing in δ and decreasing both in δ' and in δ'', for $\delta, \delta', \delta'' \geq 1$:

$$\begin{aligned}
\frac{\partial g}{\partial \delta} &= a\left(2^{\delta'+1} - 1\right) 2^{-\delta-2\delta'-\delta''} \log 2 > 0 \\
\frac{\partial g}{\partial \delta'} &= -a\left(7 \cdot 2^{\delta+\delta'} - 3 \cdot 2^{\delta+2} - 2^{\delta'+1} + 2\right) 2^{-\delta-2\delta'-\delta''} \log 2 < 0 \\
\frac{\partial g}{\partial \delta''} &= -a\left(7 \cdot 2^{\delta+\delta'} + 2^{\delta+2\delta'+1} - 3 \cdot 2^{\delta+1} - 2^{\delta'+1} + 1\right) 2^{-\delta-2\delta'-\delta''} \log 2 < 0.
\end{aligned}$$

Therefore, the maximum value of f is

$$\lim_{\delta \to \infty} f(3a, \delta, 1, 1) = 3a,$$

which is the desired upper bound for v_j. □

4 Definition of Secure Groups

Let $(\mathbb{G}, *)$ denote an arbitrary finite group, written multiplicatively. To define secure group schemes we will use $[\![a]\!]_{\mathbb{G}}$ to denote a secure representation of a group element $a \in \mathbb{G}$. In general, such a secure representation $[\![a]\!]_{\mathbb{G}}$ will be constructed from one or more secret-shared finite field elements. Also, we may compose secure representations, e.g., we may put $[\![a]\!]_{\mathbb{G}} = ([\![a']\!]_{\mathbb{G}'}, [\![a'']\!]_{\mathbb{G}''})$ as secure representation of $a = (a', a'') \in \mathbb{G}$ for a direct product group $\mathbb{G} = \mathbb{G}' \times \mathbb{G}''$.

A secure group scheme should allow us to apply the group operation $*$ to given $[\![a]\!]_{\mathbb{G}}$ and $[\![b]\!]_{\mathbb{G}}$, and obtain $[\![a * b]\!]_{\mathbb{G}}$ as a result. We will refer to this as a secure application of the group operation, or as a secure group operation, for short. Similarly, a secure group scheme may allow us to perform a secure inversion, which lets us obtain $[\![a^{-1}]\!]_{\mathbb{G}}$ for a given $[\![a]\!]_{\mathbb{G}}$. Another common task is to generate a random sample $[\![a]\!]_{\mathbb{G}}$ with $a \in_R \mathbb{G}$, hence to obtain a secure representation of a group element a drawn from the uniform distribution on $\mathbb{G}$.

To cover a representative set of tasks, we define secure groups as follows.

Definition 1. *A* **secure group scheme** *for* $(\mathbb{G}, *)$ *comprises protocols for the following tasks, where* $a, b \in \mathbb{G}$*.*

Group operation. *Given* $[\![a]\!]_{\mathbb{G}}$ *and* $[\![b]\!]_{\mathbb{G}}$*, compute* $[\![a * b]\!]_{\mathbb{G}}$*.*
Inversion. *Given* $[\![a]\!]_{\mathbb{G}}$*, compute* $[\![a^{-1}]\!]_{\mathbb{G}}$*.*
Equality test. *Given* $[\![a]\!]_{\mathbb{G}}$ *and* $[\![b]\!]_{\mathbb{G}}$*, compute* $[\![a = b]\!]$*.*
Conditional. *Given* $[\![a]\!]_{\mathbb{G}}, [\![b]\!]_{\mathbb{G}}$ *and* $[\![x]\!]$ *with* $x \in \{0, 1\}$*, compute* $[\![a^x b^{1-x}]\!]_{\mathbb{G}}$*.*
Exponentiation. *Given* $[\![a]\!]_{\mathbb{G}}$ *and* $[\![x]\!]$ *with* $x \in \mathbb{Z}$*, compute* $[\![a^x]\!]_{\mathbb{G}}$*.*
Random sampling. *Compute* $[\![a]\!]_{\mathbb{G}}$ *with* $a \in_R \mathbb{G}$ *(or, close to uniform).*
En/decoding. *For a set* S *and an injective map* $\sigma : S \to \mathbb{G}$*:*
- Encoding. *Given* $[\![s]\!]$*, compute* $[\![\sigma(s)]\!]_{\mathbb{G}}$*.*
- Decoding. *Given* $[\![a]\!]_{\mathbb{G}}$ *with* $a \in \sigma(S)$*, compute* $[\![\sigma^{-1}(a)]\!]$*.*

By default, all inputs and outputs to these protocols are secret-shared. In addition, the scheme also comprises variants of default protocols where some of the inputs and outputs are public and/or private.

By definition, a secure group scheme thus includes an ordinary group scheme where all protocols operate on public values. Also note that there may be multiple encoding/decodings for a group $\mathbb{G}$, each defined on a specific set S.

5 Generic Protocols for Secure Groups

The implementation of basic tasks such as the group operation and en/decoding strongly depend on the representation of the secure group. For other tasks, however, we may look for generic implementations. This section presents generic protocols for the following four tasks from Definition 1: conditional (if-else), random sampling, inverse, and exponentiation.

Secure Conditional. The secure conditional can be evaluated in terms of secure exponentiation in either of these two generic ways, if applicable: as $[\![a]\!]_{\mathbb{G}}^{[\![x]\!]} * [\![b]\!]_{\mathbb{G}}^{1-[\![x]\!]}$, or as $([\![a]\!]_{\mathbb{G}}/[\![b]\!]_{\mathbb{G}})^{[\![x]\!]} * [\![b]\!]_{\mathbb{G}}$, where $x \in \{0, 1\}$. This way it suffices to implement the basic operation $[\![a]\!]_{\mathbb{G}}^{[\![x]\!]}$ with $x \in \{0, 1\}$. And if base a is publicly known, it even suffices to implement $a^{[\![x]\!]}$ (with Protocol 7 as a good option to implement this operation).

In pseudocode, the secure conditional is denoted as $\mathsf{if\text{-}else}([\![x]\!], [\![a]\!]_{\mathbb{G}}, [\![b]\!]_{\mathbb{G}})$, with the obvious variations if a and/or b are publicly known. The condition $[\![x]\!]$ should always be secret.

Secure Random Sampling. The task of secure random sampling from a group $\mathbb{G}$ corresponds to generating $[\![a]\!]_{\mathbb{G}}$ with $a \in_R \mathbb{G}$ (or, close to uniform), see Definition 1. In this section we present several methods for secure random sampling.

A generic protocol is obtained by letting party $\mathcal{P}_i$ generate a random group element $a_i \in_R \mathbb{G}$ privately, and then use t (threshold) secure group operations to form $[\![a]\!]_{\mathbb{G}} = \prod_i [\![a_i]\!]_{\mathbb{G}}$ for a subset of $t + 1$ parties. A potential drawback of this generic method is the cost of t secure group operations, which may be avoided

if we use more direct methods, e.g., through efficient encodings for the group or direct use of the underlying representation of the group. For instance, to sample a point (x, y) on an elliptic curve, we may first generate $[\![x]\!]$ at random, and then solve for $\pm[\![y]\!]$, rejecting x if no solutions exist. Another potential drawback is the lack of public verifiability if parties generate random group elements privately. A common way to achieve verifiability is to start from publicly verifiable random bits, and use these bits to deterministically generate random samples.

Sampling in the Black Box Group Model. Given the structure of $\mathbb{G}$, we may reduce generating $[\![a]\!]_{\mathbb{G}}$ with $a \in_R \mathbb{G}$ to generating a random $[\![x]\!]$ with $x \in_R \mathbb{Z}_n$, in case $\mathbb{G} = \langle g \rangle$ is a cyclic group of order n, say. This extends to arbitrary abelian groups if we are given a generating set.

For arbitrary groups for which we do not know the structure of the group, however, we adopt a different approach referred to as sampling in the black box group model. An important property of algorithms such as Dixon's is that the problem of secure random sampling is reduced to the secure generation of random bits. If these bits are generated in a publicly verifiable random way, it follows that the entire protocol for random sampling in a group can be made verifiable.

For nonabelian groups, the algorithm of Dixon [Dix08, Theorem 1] is the state-of-the-art for provable complexity bounds. Its applicability suffers from an unknown constant. However, [Dix08, Theorem 3, Lemma 13(b)] offers a procedure that addresses this. In the second half of this section we argue why this technique compares favorably to other common heuristics.

Dixon's Algorithm. For our purpose, we apply Dixon's algorithm in the clear to construct a public array of group elements. Given $0 \leq \varepsilon < 1$, the algorithm is then used to securely generate ε-uniformly distributed random elements, meaning that each group element has probability $(1 \pm \varepsilon)/|\mathbb{G}|$ to appear as output. The number of group operations to construct a random element generator is proportional to $\log^2 |\mathbb{G}|$, see [Dix08, Remark 2].

For $a_1, \ldots, a_j \in \mathbb{G}$, let a *random cube of length* j, $Cube(a_1, \ldots, a_j)$, be defined by the probability distribution $\{a_1^{\epsilon_1} \cdots a_j^{\epsilon_j} : (\epsilon_1, \ldots, \epsilon_j) \in_R \{0,1\}^j\}$. Given a generating set $\mathcal{G}$ of $\mathbb{G}$, the following theorem shows that we can construct a random cube of length proportional to $\log |\mathbb{G}|$ that is 1/4-uniform with a given probability.

Theorem 2. **([Dix08, Theorem 1]).** *Let $\mathcal{G} = \{g_1, \ldots, g_d\}$ be a generating set of $\mathbb{G}$. Let $W_j = Cube(g_j^{-1}, \ldots, g_1^{-1}, g_1, \ldots, g_j)$ be a sequence of cubes, where for $j > d$, g_j is chosen at random from W_{j-1}. Then for each $\delta > 0$, there is a constant K_δ, independent of d or $\mathbb{G}$, such that with probability at least $1 - \delta$, distribution W_j is 1/4-uniform when $j \geq d + K_\delta \log |\mathbb{G}|$.*

Practical Procedure. Constant K_δ in Theorem 2 may still make the implementation impractical. The following theorem states that we can avoid the constant K_δ and reduce the cube length if we start from a distribution W that is close to

Protocol 2. $\mathsf{random}(\mathbb{G}, x)$ — random cube $Z_j(\boldsymbol{g})$ for $\mathbb{G}$

1: $[\![r_0]\!], \ldots, [\![r_{j-1}]\!] \leftarrow \mathsf{random\text{-}bits}(j)$
2: $[\![g_i^{xr_i}]\!]_{\mathbb{G}} \leftarrow \mathsf{if\text{-}else}([\![r_i]\!], g_i^x, 1_{\mathbb{G}})$, for $i = 0, \ldots, j-1$
3: $[\![a^x]\!]_{\mathbb{G}} \leftarrow \prod_i [\![g_i^{xr_i}]\!]_{\mathbb{G}}$
4: **return** $[\![a^x]\!]_{\mathbb{G}}$ ▷ random $\mathbb{G}$-element to the power x, $x \in \mathbb{Z}$

Protocol 3. $\mathsf{inverse}([\![a]\!]_{\mathbb{G}})$

1: $[\![r]\!]_{\mathbb{G}} \leftarrow \mathsf{random}(\mathbb{G})$
2: $a * r \leftarrow [\![a]\!]_{\mathbb{G}} * [\![r]\!]_{\mathbb{G}}$
3: $[\![a^{-1}]\!]_{\mathbb{G}} \leftarrow [\![r]\!]_{\mathbb{G}} * (a * r)^{-1}$
4: **return** $[\![a^{-1}]\!]_{\mathbb{G}}$

the uniform distribution U on $\mathbb{G}$ w.r.t. to the statistical (or, variational) distance $\Delta[W;U] = \frac{1}{2}\sum_{a\in\mathbb{G}} |W(a) - U(a)| = \max_{A\subset\mathbb{G}} |W(A) - U(A)|$. [Dix08, Lemma 13(b), page 11] describes the procedure in detail.

Theorem 3. **([Dix08, Theorem 3(c)]).** *Let U be the uniform distribution on $\mathbb{G}$ and suppose W is a distribution with $\Delta[W;U] \leq \varepsilon < 1$. Let $a_0, \ldots, a_{j-1} \in \mathbb{G}$ be chosen independently according to distribution W. If $Z_j = Cube(a_0, \ldots, a_{j-1})$, then with probability at least $1 - 2^{-h}$, Z_j is 2^{-k}-uniform when*

$$j \geq \beta(2\log_2 |\mathbb{G}| + h + 2k), \text{ where } \beta := \log_2(2/(1+\varepsilon)). \tag{3}$$

Given a random cube Z_j of length j that is sufficiently close to uniform random, Protocol 2 is a general protocol for secure random sampling from a group $\mathbb{G}$. The protocol requires j secure random bits and $j-1$ secure group operations. The result is raised to the power x, where $x = 1$ is intended as the default case, and this can be extended to several powers.

Secure Inverse. The sampling protocols from Sect. 5 allow us to invert secure group elements for groups with known or unknown order. Protocol 3 implements this functionality following the classical approach by [BIB89].

Secure Exponentiation. We start this section with a generally applicable protocol for secure exponentiation $[\![a^x]\!]_{\mathbb{G}}$ based on the binary representation of $[\![x]\!]$, see Protocol 4. The computational complexity is dominated by about 2ℓ group operations and ℓ calls to lsb. For the round complexity we see that the ℓ iterations of the loop are done sequentially. The protocol can be optimized in lots of ways. For instance, it is more efficient to compute the bits of $[\![x]\!]$ all at once, including the sign bit, using a standard solution.

For abelian groups the linear dependency on ℓ for the round complexity can be removed, as we show in Protocol 5. The improvement is that the ℓ iterations of the loop can now be done in parallel. Using standard techniques for constant rounds protocols (see, e.g., [BIB89, DFK+06]) the overall round complexity can be made independent of ℓ.

Protocol 4. exponentiation($[\![a]\!]_{\mathbb{G}}, [\![x]\!]$)

$[\![b]\!]_{\mathbb{G}}, [\![y]\!] \leftarrow$ if-else($[\![x < 0]\!], (\mathsf{inverse}([\![a]\!]_{\mathbb{G}}), -[\![x]\!]), ([\![a]\!]_{\mathbb{G}}, [\![x]\!])$) ▷ skip if $x \geq 0$ ensured
$[\![c]\!]_{\mathbb{G}} \leftarrow [\![1]\!]_{\mathbb{G}}$
for $i = 0$ **to** $\ell - 1$ **do** ▷ $\mathrm{len}(x) \leq \ell$ assumed
  $[\![e_i]\!] \leftarrow \mathsf{lsb}([\![y]\!])$
  $[\![c]\!]_{\mathbb{G}} \leftarrow$ if-else($[\![e_i]\!], [\![b]\!]_{\mathbb{G}} * [\![c]\!]_{\mathbb{G}}, [\![c]\!]_{\mathbb{G}}$)
  $[\![y]\!] \leftarrow ([\![y]\!] - [\![e_i]\!])/2$
  $[\![b]\!]_{\mathbb{G}} \leftarrow [\![b]\!]^2_{\mathbb{G}}$
return $[\![c]\!]_{\mathbb{G}}$

Protocol 5. exponentiation($[\![a]\!]_{\mathbb{G}}, [\![x]\!]$) $\mathbb{G}$ abelian

$[\![x_0]\!], \ldots, [\![x_{\ell-1}]\!] \leftarrow \mathsf{bits}([\![x]\!])$ ▷ $\mathrm{len}(x) + 1 \leq \ell$ assumed for two's complement
$[\![r]\!]_{\mathbb{G}}, [\![r^{-1}]\!]_{\mathbb{G}}, \ldots, [\![r^{-2^{\ell-1}}]\!]_{\mathbb{G}} \leftarrow \mathsf{random}(\mathbb{G}, 1, -1, \ldots, -2^{\ell-1})$
$b \leftarrow [\![a]\!]_{\mathbb{G}} * [\![r]\!]_{\mathbb{G}}$ ▷ $b = a * r$
for $i = 0$ **to** $\ell - 1$ **do**
  $[\![c_i]\!]_{\mathbb{G}} \leftarrow$ if-else($[\![x_i]\!], b^{2^i} * [\![r^{-2^i}]\!]_{\mathbb{G}}, 1_{\mathbb{G}}$) ▷ in parallel
return $[\![c_0]\!]_{\mathbb{G}} * \cdots * [\![c_{\ell-2}]\!]_{\mathbb{G}} * [\![c_{\ell-1}]\!]^{-1}_{\mathbb{G}}$

These protocols can be optimized if either a or x is public. For instance, if a is public, the list of powers $a, a^2, a^4, \ldots$ can be computed locally, and if x is public, one can use techniques such as addition chains.

However, if one of the two inputs a or x is public, we can obtain efficient protocols by securely randomizing the other input. Protocol 6 solves the case that x is public, assuming that the group is abelian. The protocol uses secure random sampling from $\mathbb{G}$ to obtain both a random group element and its $(-x)$th power.

Protocol 7 solves the case that a is public. The protocol random-pair(a) outputs a random exponent $[\![r]\!]$ together with $[\![a^{-r}]\!]_{\mathbb{G}}$ for public input $a \in \mathbb{G}$. For public output a^x, we can also use public a^{-r} in the first step of the protocol. Finally, in the special case that a is an element of large order, parties may directly use their Shamir secret shares, and perform Lagrange interpolation in the exponent to raise a to the power $[\![x]\!]$.

6 Secure Class Groups

We focus on ideal class groups of imaginary quadratic fields, using $\mathrm{Cl}(\Delta)$ to denote the class group with discriminant $\Delta < 0$. We also use $\mathrm{Cl}(\Delta)$ to denote the isomorphic form class group of integral binary quadratic forms $f(x, y) = ax^2 + bxy + cy^2$ with discriminant $\Delta = b^2 - 4ac < 0$. These forms are written as $f = (a, b, c)$ with $a, b, c \in \mathbb{Z}$, where it is implied that f is primitive, that is, $\gcd(a, b, c) = 1$, and f is positive definite, that is $a > 0$.

For the composition of two forms $f_1, f_2 \in \mathrm{Cl}(\Delta)$ we will use the algorithm due to Shanks [Sha71], as presented by Cohen [Coh93, Algorithm 5.4.7]. We slightly adapt the algorithm, skipping some case distinctions that were introduced for

Protocol 6. exponentiation$([\![a]\!]_{\mathbb{G}}, x)$ public exponent x, $\mathbb{G}$ abelian

1: $[\![r]\!]_{\mathbb{G}}, [\![r^{-x}]\!]_{\mathbb{G}} \leftarrow$ random$(\mathbb{G}, 1, -x)$
2: $a * r \leftarrow [\![a]\!]_{\mathbb{G}} * [\![r]\!]_{\mathbb{G}}$
3: $[\![a^x]\!]_{\mathbb{G}} \leftarrow (a * r)^x * [\![r^{-x}]\!]_{\mathbb{G}}$
4: **return** $[\![a^x]\!]_{\mathbb{G}}$

Protocol 7. exponentiation$(a, [\![x]\!])$ public base a

1: $[\![r]\!], [\![a^{-r}]\!]_{\mathbb{G}} \leftarrow$ random-pair(a)
2: $x + r \leftarrow [\![x]\!] + [\![r]\!]$ ▷ $x + r$ should be sufficiently random
3: $[\![a^x]\!]_{\mathbb{G}} \leftarrow a^{x+r}[\![a^{-r}]\!]_{\mathbb{G}}$
4: **return** $[\![a^x]\!]_{\mathbb{G}}$

efficiency, see Algorithm 2. Apart from the computationally nontrivial reduction performed in the final step of the algorithm, the resulting algorithm only requires two xgcd's and one integer division. This compares favorably with alternatives such as the classical composition algorithm by Dirichlet [LD63] (see also [Cox11, Lemma 3.2] and [Lon19, Algorithm 6.1.1]).

As secure representation of a form $f = (a, b, c) \in \mathbb{G}$ we define $[\![f]\!]_{\mathbb{G}} = ([\![a]\!]_p, [\![b]\!]_p, [\![c]\!]_p)$, for a sufficiently large prime p. The prime p should be sufficiently large such that the intermediate forms computed by Algorithm 2 do not cause any overflow modulo p (also accounting for the "headroom" needed for secure comparison and secure integer division). Using Protocol 1 for secure xgcd and a protocol for secure integer division, Algorithm 2 is then easily transformed into a protocol for the secure composition of forms.

To reduce a form $f = (a, b, c)$, the classical reduction algorithm by Lagrange (see [BV07, Algorithm 5.3]) runs in at most $2 + \lceil \log_2(a/\sqrt{\Delta}) \rceil$ steps [BV07, Theorem 5.5.4], each step requiring an integer division. Our goal is to avoid the expensive (secure) integer divisions where possible.

To this end, we take the binary reduction algorithm by [AF06, Algorithm 3] as our starting point. (Recently, [DEH22] arrived at an equivalent algorithm to design a quantum circuit to reduce binary quadratic forms.) We minimize the number of comparisons by exploiting the invariant $a > 0$ and $c > 0$. The algorithm reduces forms by the following transformations in $SL_2(\mathbb{Z})$:

$$S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad T_m = \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix}.$$

The total number of iterations of the main loop required to achieve $|b| \leq 2a$ is at most $\text{len}(b)$, if $f = (a, b, c)$ is the given form. This follows from the fact that if $|b| \leq 2a$ does not hold yet, an iteration of the main loop will reduce $\text{len}(b)$ by at least 1. We will ensure that $\text{len}(b) \leq \text{len}(\Delta)$, such that it suffices to run the main loop for $\text{len}(\Delta)$ iterations, independent of the input. Note that we need to test $a > c$ in each iteration as well.

As an important optimization, we limit the number of iterations to $\text{len}(\Delta)/2$ by noting that it suffices to reduce b until $\text{len}(b) < \text{len}(\Delta)/2$. This ensures that

Algorithm 2. compose(f_1, f_2) f_1, f_2 primitive, positive definite, reduced

Input: $f_1 = (a_1, b_1, c_1)$ and $f_2 = (a_2, b_2, c_2)$
$s \leftarrow (b_1 + b_2)/2$
$d', x_1, y_1 \leftarrow \mathsf{xgcd}(a_1, a_2)$ ▷ $x_1 a_1 + y_1 a_2 = d' = \mathsf{gcd}(a_1, a_2)$
$d, x_2, y_2 \leftarrow \mathsf{xgcd}(s, d')$ ▷ $x_2 s + y_2 d' = \mathsf{gcd}(s, d')$
$v_1, v_2 \leftarrow a_1/d, a_2/d$
$r \leftarrow (y_1 y_2 (s - b_2) - x_2 c_2) \bmod v_1$ ▷ integer division
$a_3 \leftarrow v_1 v_2$
$b_3 \leftarrow b_2 + 2 v_2 r$
$c_3 \leftarrow (b_3^2 - \Delta)/(4a_3)$
$f_3 = (a_3, b_3, c_3)$
return reduce(f_3)

Algorithm 3. reduce(f) $f \in \mathrm{Cl}(\Delta)$ primitive, positive definite

Input: $f = (a, b, c)$
for $i = 1$ **to** $\lceil \mathrm{len}(\Delta)/2 \rceil$ **do** ▷ invariant $a > 0$ and $c > 0$
  if $b > 0$ **then** $s_b \leftarrow 1$ **else** $s_b \leftarrow -1$ ▷ using $\mathrm{len}(b) < \mathrm{len}(\Delta) - i$
  $j \leftarrow \mathrm{len}(b) - \mathrm{len}(a) - 1$
  $m \leftarrow -s_b 2^j$ ▷ compute 2^j together with $\mathrm{len}(a)$ and $\mathrm{len}(b)$ at no extra cost
  if $s_b b > 2a$ **then** ▷ $|b| > 2a$
    $f \leftarrow fT_m$ ▷ $fT_m = (a, b + 2ma, m^2 a + mb + c)$
  if $a > c$ **then**
    $f \leftarrow fS$ ▷ $fS = (c, -b, a)$
$m \leftarrow \lfloor \frac{a-b}{2a} \rfloor$ ▷ integer division
$f \leftarrow fT_m$
if $a > c$ **then**
  $f \leftarrow fS$
if $(a + b)(a - c) = 0$ and $b < 0$ **then** ▷ ensure $b \geq 0$ if $a = -b$ or $a = c$
  $b \leftarrow -b$ ▷ $f \leftarrow fS = fT_1$
return f

$|b| \leq \sqrt{|\Delta|}$ after the main loop. If $|b| \leq 2a$ does not hold yet, then it follows that $a < |b|/2 \leq \sqrt{|\Delta|/4}$, and we only need one "normalization" step to ensure $|b| \leq a$.

The result is presented as Algorithm 3. This algorithm avoids the integer division and multiple comparisons in the main loop of Lagrange's reduction algorithm, at the cost of three secure comparisons and two secure bit length computations in the main loop. To compute the bit length securely we use a novel protocol avoiding the use of a full bit decomposition.

For secure encoding to class groups, a given integer s will be mapped to a form (a, b, c) by computing a as a simple function of s and setting b as the square root of Δ modulo $4a$, see Algorithm 4. We note that this encoding improves upon the encoding proposed by [Sch03] (compare to [Sch99] as well), which relies on using prime numbers for a. Our algorithm avoids the need for primality tests, which are dominating the computational cost for the encoding algorithm.

Algorithm 4. $\mathsf{encode}(s, \mathrm{Cl}(\Delta))$ — s sufficiently small w.r.t. $|\Delta|$

$n = s \cdot gap_\ell$
$a \leftarrow n - 1 - (n \bmod 4)$
repeat
  $a \leftarrow a + 4$ ▷ $a \equiv 3 \pmod 4$
  $b \leftarrow \Delta^{\frac{a+1}{4}} \bmod a$
until $b^2 \equiv \Delta \pmod a$ and $\gcd(a, b) = 1$
if $\Delta \not\equiv b \pmod 2$ **then**
  $b \leftarrow a - b$
$f \leftarrow (a, b, \frac{b^2 - \Delta}{4a})$ ▷ $\Delta \equiv b^2 \pmod 4$
$d \leftarrow n - a$
return f, d

7 Concluding Remarks

We have presented protocols for the extended gcd and the class group operations in an MPC setting. Python implementations of these protocols have been integrated in the MPyC package [Sch18]. This package covers protocols for secure integer arithmetic involving basic operations like $+, *, <$. More advanced operations such as secure integer division and secure computation of the bit length are also included in the MPyC package (details for these protocols will be covered in other work).

The implementation in MPyC also extends to other secure groups, including Schnorr groups and elliptic curves. Many results pertaining to threshold cryptosystems and threshold signatures from the literature can thus be stated in terms of secure groups. For example, secure groups make it straightforward to conduct the prover side of a Σ-protocol, a compressed Σ-protocol [AC20] or succinct noninteractive argument of knowledge (SNARK) in MPC. This allows for convenient constructions of *verifiable MPC* protocols [SVdV16], i.e., proof systems that enable MPC parties to create a publicly verifiable proof of correctness of the MPC computation, even in the extreme case that all MPC parties are corrupt. A demonstration of verifiable MPC is available in a separate repository [SS22].

Acknowledgements. We thank Alessandro Danelon, Mark Abspoel, Niek Bouman, Thomas Attema, and the anonymous reviewers for their valuable comments. This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 780477 (PRIViLEDGE).

References

[AC20] Attema, T., Cramer, R.: Compressed *Sigma*-protocol theory and practical application to Plug & play secure algorithmics. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 513–543. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_18

[ACS02] Algesheimer, J., Camenisch, J., Shoup, V.: Efficient computation modulo a shared secret with application to the generation of shared safe-prime products, 417–432 (2002)

[AF06] Agarwal, S., Frandsen, G.S.: A new GCD algorithm for quadratic number rings with unique factorization. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 30–42. Springer, Heidelberg (2006). https://doi.org/10.1007/11682462_8

[BB87] Bojanczyk, A.W., Brent, R.P.: A systolic algorithm for extended GCD computation. Comput. Math. Appl. **14**(4), 233–238 (1987)

[BGW88] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Proceedings of Symposium on Theory of Computing (STOC '88), pp. 1–10. ACM (1988)

[BHR+21] Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Time- and space-efficient arguments from groups of unknown order. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 123–152. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_5

[BIB89] Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction, 201–209 (1989)

[BK85] Brent, R.P., Kung, H.T.: A systolic algorithm for integer GCD computation. In 1985 IEEE 7th Symposium on Computer Arithmetic (ARITH), pages 118–125 (1985)

[BV07] Buchmann, J.A., Vollmer, U.: Binary quadratic forms - an algorithmic approach, volume 20 of Algorithms and computation in mathematics. Springer (2007)

[BY19] Bernstein, D.J., Yang, B.-Y.: Fast constant-time GCD computation and modular inversion. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(3), 340–398 (2019)

[Coh93] Cohen, H.: A course in computational algebraic number theory, volume 138 of Graduate texts in mathematics. Springer (1993)

[Cox11] Cox, D.A.: Primes of the form $x^2 + ny^2$: Fermat, class field theory, and complex multiplication, volume 34. John Wiley & Sons (2011)

[CS10] Catrina, O., Saxena, A.: Secure computation with fixed-point numbers, 35–50 (2010)

[DEH22] David, N., Espitau, T., Hosoyamada, A.: Quantum binary quadratic form reduction. IACR Cryptol. ePrint Arch. p. 466 (2022)

[DFK+06] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation, pp. 285–304 (2006)

[DGS22] Dobson, S., Galbraith, S., Smith, B.: Trustless unknown-order groups. Math. Cryptol. **1**(2), 25–39 (2022)

[Dix08] Dixon, J.D.: Generating random elements in finite groups. Electron. J. Comb., 15(1) (2008)

[GRR98] Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: Proceedings of Principles of Distributed Computing, PODC '98, pp. 101–111. ACM (1998)

[Hoo12] Hoogh, de, S.J.A.: Design of large scale applications of secure multiparty computation : secure linear programming. PhD thesis, Technische Universiteit Eindhoven, Department of Mathematics and Computer Science (2012)

[Knu69] Knuth, D.E.: The art of computer programming, volume 2: Seminumerical algorithms (1969)
[LD63] Dirichlet, P.L.: Vorlesungen über Zahlentheorie (1863)
[Lon19] Long, L.: Binary quadratic forms. GitHub https://github.com/Chia-Network/vdf-competition/blob/master/classgroups.pdf (2019). Accessed 23 Jan 2020
[MvOV96] Menezes, A., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996)
[Sch99] Schaub, J.: Implementiering von Public-Key-Kryptosystemen über imaginär-quadratischen Ordnungen (Master's thesis). Technische Universität Darmstadt, Fachbereich Informatik (1999)
[Sch03] Schielzeth, D.: Realisierung der elgamal-verschlüsselung in quadratischen zählkorpern (Master's thesis). Technische Universität Berlin. http://www.math.tu-berlin.de/kant/publications.html (2003)
[Sch18] Schoenmakers, B.: MPyC secure multiparty computation in Python. GitHub https://github.com/lschoe/mpyc (2018)
[Sha71] Shanks, D.: Class number, a theory of factorization, and genera. In: Proceedings of the Symp. Math. Soc., 1971, volume 20, pages 41–440, 1971
[SS22] Schoenmakers, B., Segers, T.: Verifiable MPC. GitHub. https://github.com/toonsegers/verifiable_mpc (2022)
[SVdV16] Schoenmakers, B., Veeningen, M., de Vreede, N.: Trinocchio: privacy-preserving outsourcing by distributed verifiable computation. In: Manulis, M., Sadeghi, A.-R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 346–366. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39555-5_19
[SZ04] Stehlé, D., Zimmermann, P.: A binary recursive GCD algorithm. In: Buell, D., (edt.), Algorithmic Number Theory, pages 411–425, Berlin, Heidelberg. Springer, Berlin Heidelberg (2004)
[Tof07] Toft, T.: Primitives and applications for multi-party computation. PhD thesis, Aarhus University (2007)
[Wes19] Wesolowski, B.: Efficient verifiable delay functions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 379–407. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_13