cryptography

MDPI

Article

Secure Groups for Threshold Cryptography and Number-Theoretic Multiparty Computation †

Berry Schoenmakers and Toon Segers *

Department of Mathematics & Computer Science, TU Eindhoven, 5600 MB Eindhoven, The Netherlands; l.a.m.schoenmakers@tue.nl

* Correspondence: a.j.m.segers@tue.nl

† This paper is an extended version of our paper published in CSCML 2023.

Abstract: In this paper, we introduce secure groups as a cryptographic scheme representing finite groups together with a range of operations, including the group operation, inversion, random sampling, and encoding/decoding maps. We construct secure groups from oblivious group representations combined with cryptographic protocols, implementing the operations securely. We present both generic and specific constructions, in the latter case specifically for number-theoretic groups commonly used in cryptography. These include Schnorr groups (with quadratic residues as a special case), Weierstrass and Edwards elliptic curve groups, and class groups of imaginary quadratic number fields. For concreteness, we develop our protocols in the setting of secure multiparty computation based on Shamir secret sharing over a finite field, abstracted away by formulating our solutions in terms of an arithmetic black box for secure finite field arithmetic or for secure integer arithmetic. Secure finite field arithmetic suffices for many groups, including Schnorr groups and elliptic curve groups. For class groups, we need secure integer arithmetic to implement Shanks' classical algorithms for the composition of binary quadratic forms, which we will combine with our adaptation of a particular form reduction algorithm due to Agarwal and Frandsen. As a main result of independent interest, we also present an efficient protocol for the secure computation of the extended greatest common divisor. 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 multiparty computation 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. We have integrated secure groups in the Python package MPyC and have implemented threshold ElGamal and threshold DSA in terms of secure groups. We also mention how our results support verifiable multiparty computation, allowing parties to jointly create a publicly verifiable proof of correctness for the results accompanying the results of a secure computation.

Keywords: multiparty computation; extended gcd; elliptic curve cryptography; class groups; threshold cryptography

Citation: Schoenmakers, B.; Segers, T. Secure Groups for Threshold Cryptography and Number-Theoretic Multiparty Computation. *Cryptography* **2023**, *7*, 56. https://doi.org/10.3390/cryptography7040056

Academic Editors: Carlo Blundo and Josef Pieprzyk

Received: 6 September 2023
Revised: 21 October 2023
Accepted: 31 October 2023
Published: 9 November 2023

1. Introduction

A finite group is defined as a finite set of group elements together with an associative group operation and the corresponding inverse operation. We introduce secure groups as an oblivious data structure implementing finite groups in a privacy-preserving manner. Both the representation of group elements and the implementation of the group operations should be oblivious, meaning that no information can be inferred about the values of the group elements involved. Secure groups aim to significantly simplify implementations of threshold cryptography, providing efficient primitives for typical cryptographic operations such as secure exponentiation, random sampling, and encoding/decoding. This paper is an extended version of our paper published in CSCML 2023 [1].

We develop the underlying protocols in the setting of secure multiparty computation (MPC) based on Shamir secret sharing over a finite field. We will distinguish between generic constructions and specific constructions for finite groups used in cryptography, particularly the group of quadratic residues, elliptic curve groups, and class groups.

Generic constructions include the application of secure table lookup to the multiplication table of the group. Also, if a faithful linear representation over a finite field is available from representation theory, it directly permits an oblivious representation of the group elements (as square matrices over a finite field) and an oblivious implementation of the group operation (as matrix multiplication). We note that Bar-Ilan and Beaver already considered a generic protocol for secure inversion of group elements as a natural generalization of secure matrix multiplication ([2] Lemma 6). In general, the implementation of basic tasks such as the group operation and en/decoding depends on the representation of the secure group. Where practical, we may look for generic implementations first. Therefore, we present generic protocols for the following tasks: conditional (if–else), random sampling, inversion, and exponentiation.

We also define specific oblivious representations and operations for a set of frequently used finite groups. The multiplicative group $\mathbb{F}_q^*$ and any of its subgroups (quadratic residues and, more generally, Schnorr groups) directly permit an oblivious representation and group operation using secret shares over $\mathbb{F}_q$. Encoding and decoding for these groups is in general hard, so we present several techniques and trade-offs focusing on quadratic residues. Groups defined on elliptic curves over finite fields directly permit an oblivious representation. To facilitate the implementation of such groups, we employ complete formulas known for Weierstrass and Edwards curves. We employ a result from [3] for parallel architectures, which yields a secure group operation of particularly low multiplicative depth. Ideal class groups of imaginary quadratic fields have been studied extensively for use in cryptography by Buchmann et al. (see [4] and references therein) and recently received renewed interest (see, e.g., [5–7]). 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 binary quadratic forms is the secure computation of the extended greatest common divisor. We start from the constant-time algorithms proposed by Bernstein and Yang [8]. Their approach relies on the use of 2-adic arithmetic, but we will work purely over the integers for an efficient solution in an 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 total number of $O(\ell \log \ell)$ secure multiplications compares favorably to, e.g., a basic binary gcd algorithm ([9] Algorithm 14.61), which requires $O(\ell^2)$ secure multiplications. Other binary gcd algorithms such as Bojanczyk and Brent [10] (and related algorithms as considered by [8]) as well as the algorithm attributed to Penk ([11] Vol. 2, 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 [12]. 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 an 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.

Finally, as related work in the area of threshold cryptography, we note that several papers on threshold signatures [13,14], threshold Σ-proofs [15], and verifiable (or auditable)

MPC [16,17] implicitly implement some form of secure groups for $\mathbb{F}_q^*$ or elliptic curves, but these papers do not treat secure groups in general terms.

The paper is organized as follows. Above, we have elaborated on our contributions and related work regarding secure groups. In Section 2, we present preliminaries on secure multiparty computation. Section 3 presents our efficient protocol for the extended greatest common divisor, including a careful analysis establishing a nontrivial bound on the growth of the Bézout coefficients. Section 4 introduces our cryptographic scheme for secure groups. In Section 5, we present generic constructions for secure groups, and, in Section 6, we present generic protocols for the following tasks: conditional (if–else), random sampling, inversion, and exponentiation. Section 7 presents specific constructions and protocols for groups used in cryptography, particularly quadratic residues, elliptic curve groups, and class groups. As an example application in threshold cryptography, Section 8 presents a simple threshold cryptosystem built from secure groups. We conclude in Section 9 with remarks about implementations 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 [18,19], easily extended to cover secure and efficient dot products as well. We write $[\![a]\!]$ (also $[\![a]\!]_p$) for a Shamir secret sharing of a finite field element $a \in \mathbb{F}_{p^d}$, where we assume $p^d > m$. For secure integer arithmetic, we have $d = 1$ (prime fields) and 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. For example, 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, $[\![e]\!] \leftarrow \mathsf{if\_else}([\![c]\!], [\![a]\!], [\![b]\!])$ denotes the secure conditional, where $e = c(a - b) + b$. A protocol that generates a vector of j secret-shared bits in $\mathbb{F}_p \cap \{0, 1\}$ is denoted by $\mathsf{random\text{-}bits}(\mathbb{F}_p, j)$. See [20–22] 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 [23–25]. Finally, we assume an efficient protocol for securely determining the bit length $[\![\mathrm{len}(a)]\!]$. See also Section 9.

3. Secure Extended GCD

This section presents a new protocol for the secure computation of the extended greatest common divisor (xgcd) of secret-shared integers. In Section 7.3, we use this protocol for the composition of binary quadratic forms in secure class groups.

We first present Algorithm 1, which can be viewed as an alternative to the divsteps2 algorithm by Bernstein and Yang [8], 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 post-processing (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).

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

As a starting point, we take the constant-time extended gcd algorithm divsteps2 by Bernstein and Yang ([8] 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 ([8] Section 8):

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

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 [26] and the Stehlé–Zimmermann step function [27]. 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 xgcd$(n, [\![a]\!], [\![b]\!])$ $n, a, b \in \mathbb{N}$, a odd

1: $[\![\delta]\!], [\![f]\!], [\![g]\!], [\![v]\!], [\![r]\!] \leftarrow 1, [\![a]\!], [\![b]\!], 0, 1$
2: **for** $i = 1$ **to** n **do**
3: $\quad [\![g_0]\!] \leftarrow [\![g \bmod 2]\!]$
4: $\quad$ **if** $[\![\delta]\!] > 0$ and $[\![g_0]\!] = 1$ **then**
5: $\quad\quad [\![\delta]\!], [\![f]\!], [\![g]\!], [\![v]\!], [\![r]\!] \leftarrow -[\![\delta]\!], [\![g]\!], -[\![f]\!], [\![r]\!], -[\![v]\!]$
6: $\quad$ **if** $[\![g_0]\!] = 1$ **then**
7: $\quad\quad [\![g]\!], [\![r]\!] \leftarrow [\![g]\!] + [\![f]\!], [\![r]\!] + [\![v]\!]$
8: $\quad$ **if** $[\![r \bmod 2]\!] = 1$ **then**
9: $\quad\quad [\![r]\!] \leftarrow [\![r]\!] + [\![a]\!]$
10: $\quad [\![\delta]\!], [\![g]\!], [\![r]\!] \leftarrow [\![\delta]\!] + 1, [\![g]\!]/2, [\![r]\!]/2$
11: **if** $[\![f]\!] < 0$ **then**
12: $\quad [\![f]\!], [\![v]\!] \leftarrow -[\![f]\!], -[\![v]\!]$
13: $[\![u]\!] \leftarrow ([\![f]\!] - [\![v]\!] * [\![b]\!])/[\![a]\!]$
14: **return** $[\![f]\!], [\![u]\!], [\![v]\!]$ ▷ $f = ua + vb = \gcd(a, b)$

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, which 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 completed securely. The secure computations of $[\![g \bmod 2]\!]$ and $[\![r \bmod 2]\!]$ are completed 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 ([9] 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 also Section 9.

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 analyze 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 by induction on j that all v_j are bounded.

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 obtain 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 obtain

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

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 $v_j \leq 3a$. From Lemma 3, we have

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

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

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

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

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

If $v_{j-2} \leq 0$, 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

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

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

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

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

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

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 obtain, for $\delta, \delta', \delta'' \geq 1$

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

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 $\delta, \delta', \delta'' \geq 1$:

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

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. Secure Groups

Let $(\mathbb{G}, *)$ denote an arbitrary finite group, written multiplicatively. To define secure group schemes, we 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 $a \in_R \mathbb{G}$ and 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 encodings/decodings for a group $\mathbb{G}$, each defined on a specific set S.

To illustrate what we mean by variants of default protocols, consider the following cases for secure exponentiation (which are covered in Section 6.4): (i) given public a and secret $[\![x]\!]$, compute secret $[\![a^x]\!]_{\mathbb{G}}$; (ii) given secret $[\![a]\!]_{\mathbb{G}}$ and public x, compute secret $[\![a^x]\!]_{\mathbb{G}}$. Similarly, variants of the trivial secure encoding/decoding protocols with $S = \mathbb{G}$ and σ the identity map on $\mathbb{G}$ will allow us to support private input and output of group elements: (i) given private input a, compute secret $[\![a]\!]_{\mathbb{G}}$, and (ii) given secret $[\![a]\!]_{\mathbb{G}}$, compute private output a. The private inputs and outputs may even belong to external parties not taking part in the MPC protocol.

We have included a representative range of tasks in our definition of secure groups. Even though some of the tasks are redundant, they are included nevertheless because these tasks cannot necessarily be implemented without loss of efficiency in terms of the other tasks. For instance, secure random sampling can be implemented by letting each party $\mathcal{P}_i$ generate a random group element $a_i \in_R \mathbb{G}$ as private input, for $i = 1, \ldots, t+1$, and then computing the product $[\![a_1]\!]_{\mathbb{G}} \cdots [\![a_{t+1}]\!]_{\mathbb{G}}$ securely. Depending on the implementation, however, this may be completed more efficiently for particular groups.

A class of tasks that we have not (yet) incorporated in the definition of secure groups includes higher-order versions of basic tasks. An important example is multi-exponentiation

$[\![\prod_i a_i^{x_i}]\!]_{\mathbb{G}}$. Similarly, the n-ary group operation $[\![\prod_{i=1}^{n} a_i]\!]_{\mathbb{G}}$ may be included as a basic task, for which there may be more efficient solutions compared to $n-1$ applications of the secure group operation.

5. Generic Constructions of Secure Groups

In this section, we present two generic constructions for secure groups. The first construction relies on secure (oblivious) table lookup and performs best for groups of limited size and without much structure. The second construction relies on linear representations, which may result in compact and efficient implementations for particular groups (like for Rubik's Cube group of order 43,252,003,274,489,856,000). In Section 7, we will complement these results with specific constructions for number-theoretic groups used in cryptography.

5.1. Secure Groups from Table Lookup

For our first generic construction of secure group schemes for arbitrary groups $\mathbb{G}$, we apply secure table lookup to the multiplication table of $\mathbb{G}$. This construction works best for relatively small groups because the performance is polynomial in $n = |\mathbb{G}|$.

Let $\sigma_0 : [0, n) \to \mathbb{G}$ be an arbitrary encoding for $\mathbb{G}$ such that $\sigma_0(0) = 1$ is the identity element of $\mathbb{G}$. The multiplication table for $\mathbb{G}$ is then represented by a public matrix $X \in [0, n)^{n \times n}$ with $X_{i,j} = \sigma_0^{-1}(\sigma_0(i) * \sigma_0(j))$ for all $i, j \in [0, n)$.

For the secure representation of any $a \in \mathbb{G}$, we set $[\![a]\!]_{\mathbb{G}} = [\![\sigma_0^{-1}(a)]\!]_p$ for a fixed prime $p \geq n$. Thus, each group element is represented by a secret-shared integer in the range $[0, n)$, and, in particular, we have $[\![1]\!]_{\mathbb{G}} = [\![0]\!]_p$. To implement the group operation securely, we take $[\![a * b]\!]_{\mathbb{G}} = [\![u]\!]_p^T X [\![v]\!]_p$, where $[\![u]\!]_p$ and $[\![v]\!]_p$ are secret-shared unit vectors of length n corresponding to $[\![a]\!]_{\mathbb{G}}$ and $[\![b]\!]_{\mathbb{G}}$, respectively. A unit vector has one entry equal to 1 and all its other entries equal to 0. There are efficient protocols for converting $[\![i]\!]_p$, $0 \leq i < n$ to the corresponding unit vector $[\![e_i]\!]_p$ and back.

The secure equality test between $[\![a]\!]_{\mathbb{G}} = [\![i]\!]_p$ and $[\![b]\!]_{\mathbb{G}} = [\![j]\!]_p$ reduces to a secure equality test $[\![i = j]\!]_p$.

The secure conditional can also be implemented efficiently given $[\![x]\!]_p$ with $x \in \{0, 1\}$. Given $[\![a]\!]_{\mathbb{G}} = [\![i]\!]_p$ and $[\![b]\!]_{\mathbb{G}} = [\![j]\!]_p$, we have $[\![a^x b^{1-x}]\!]_{\mathbb{G}} = [\![x]\!]_p([\![i]\!]_p - [\![j]\!]_p) + [\![j]\!]_p$; hence, the result is obtained with a single secure multiplication modulo p.

To generate a random group element, we generate a uniform random integer $[\![r]\!]_p$ with $r \in_R [0, n)$. This requires about $\log_2 n$ secure random bits (modulo p).

For the remaining tasks, we can use the generic protocols presented later in this paper. If desired, these protocols can also be optimized.

5.2. Secure Groups from Faithful Linear Representations

Our second generic construction of secure group schemes builds on the existence of faithful linear representations for arbitrary groups $\mathbb{G}$. A faithful linear representation of $\mathbb{G}$ is an injective homomorphism $\rho : \mathbb{G} \to GL_d(\mathbb{F}_q)$ for some finite field $\mathbb{F}_q$. The general existence of such linear representations can be inferred from Cayley's theorem, which states that every group $\mathbb{G}$ is isomorphic to a subgroup of the symmetric group acting on $\mathbb{G}$; clearly, every permutation can be represented by a unique permutation matrix over $\mathbb{F}_2$.

The basic idea is to use encoding $\sigma_\rho(A) = \rho^{-1}(A)$, defined for any $A \in \rho(\mathbb{G}) \subseteq GL_d(\mathbb{F}_q)$. Hence, we set $[\![a]\!]_{\mathbb{G}} = [\![\rho(a)]\!]_q$ such that group element a is represented by a secret-shared $d \times d$ matrix $A = \rho(a)$ with entries in $\mathbb{F}_q$. The secure group operation for $[\![a]\!]_{\mathbb{G}}$ and $[\![b]\!]_{\mathbb{G}}$ is implemented as a secure matrix product $[\![\rho(a)]\!]_q [\![\rho(b)]\!]_q$, which can be computed efficiently by performing d^2 secure dot products in parallel using one round of communication. The secure inverse of $[\![a]\!]_{\mathbb{G}}$ can be computed efficiently by generating a matrix $[\![R]\!]_q$ with $R \in_R GL_d(\mathbb{F}_q)$ and opening $[\![\rho(a)]\!]_q [\![R]\!]_q$, inverting this matrix in the clear to obtain $\rho(a^{-1})R^{-1}$, and multiplying this with $[\![R]\!]_q$ to obtain the result. Here, R can be any matrix in $GL_d(\mathbb{F}_q)$; hence, R does not have to lie in $\rho(\mathbb{G})$.

The secure equality test reduces to securely testing if $[\![\rho(a) - \rho(b)]\!]_q$ is the all-zero matrix. The secure conditional is implemented efficiently, provided $[\![x]\!]_q$ with $x \in \{0, 1\}$, by

setting $[\![a^x b^{1-x}]\!]_{\mathbb{G}} = [\![x]\!]_q([\![\rho(a)]\!]_q - [\![\rho(b)]\!]_q) + [\![\rho(b)]\!]_q$; hence, the result is obtained with a single secure multiplication of a scalar with a matrix over F_q.

The representation theory of finite groups [28] helps us to find not just any linear representation but rather (faithful) linear representations ρ of low degree d, preferably over a small finite field $\mathbb{F}_q$. As a simple example, we consider the representation of an arbitrary cyclic group of prime order p. These groups are all isomorphic to $(\mathbb{Z}_p, +)$, the group of integers modulo p with addition. To obtain a linear representation of degree 2 for this group, one takes

$$\rho : \mathbb{Z}_p \to GL_2(\mathbb{F}_p), \qquad i \mapsto \begin{pmatrix} 1 & i \\ 0 & 1 \end{pmatrix}.$$

The essential property is that $\rho(i)\rho(j) = \rho(i+j)$ for all $i, j \in \mathbb{Z}_p$. However, a linear representation of degree 1 is also possible: in fact, for a cyclic group of any order n, $n \geq 1$, let g be an element of order n in $\mathbb{F}_q^*$ for some prime power q. Then, we simply take $\rho(i) = g^i$ for $i \in \mathbb{Z}_n$.

As a more advanced example, we illustrate the use of a linear representation of *minimum* degree for the well-known Rubik's Cube group. The Rubik's Cube group $\mathbb{G}$ is a subgroup of S_{48} generated by the six permutations corresponding to a clockwise turn of each side (keeping the center pieces at rest). The smallest faithful linear representation of the Rubik's Cube group turns out to be of degree 20 [29].

Concretely, linear representation $\rho : \mathbb{G} \to GL_{20}(\mathbb{F}_7)$ can be used, where $\rho(a)$ for $a \in \mathbb{G}$ corresponds with a generalized permutation matrix that encodes all 20 movable cubies as positions (12 edge and 8 corner cubies). Each position encodes an edge flip or a corner twist as elements in $\mathbb{F}_7$ of multiplicative order 2 and 3, respectively. We can define $\rho(a)$ as a block diagonal matrix over $\mathbb{F}_7$ with two blocks: a 12×12 generalized permutation matrix with its nonzero entries in $\{-1, 1\}$ and an 8×8 generalized permutation matrix with its nonzero entries in $\{2, 4, 1\}$. Moreover, the following conditions should be satisfied: for each block, the product of its nonzero entries is equal to 1 (modulo 7), and the permutations corresponding to the two blocks are of equal parity (i.e., both permutations are even, or both are odd). This results in $12!\,2^{12}\,8!\,3^8/2/3/2 = 2^{10}3^7 8!12!$ possible representations, matching the order of the Rubik's Cube group. As a "toy example" of an application of secure groups, one can replace the trusted shuffler in a Rubik's Cube competition by using secure random sampling in the Rubik's Cube group.

Note that cryptographic applications often require hardness of the discrete logarithm problem for the particular group, making linear representations unsuitable for these applications. However, in non-cryptographic applications of secure groups, linear representations can help the construction of an efficient secure group representation, as illustrated above for the Rubik's Cube group.

6. Generic Protocols for Secure Groups

In Section 5, we have shown several representations for secure groups and discussed protocols for implementing the tasks listed in Definition 1. In general, the implementation of basic tasks such as the group operation itself and en/decoding strongly depends 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, inversion, and exponentiation.

6.1. Secure Conditional

It is usually best to implement the secure conditional directly in terms of the underlying representation of a secure group, as we have shown in Sections 5.1 and 5.2. Alternatively, 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 $x \in \{0, 1\}$.

Moreover, 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 will be denoted as if–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.

6.2. 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 using 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.

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.

When we do not know the structure of the group, we adopt a different approach referred to as sampling in the black box group model. Dixon's algorithm (Theorem 2 below) is the state of the art for provable complexity bounds, particularly for nonabelian groups.

We apply Theorem 2 in the clear to construct a public array of group elements, referred to as a random cube. The number of group operations to construct such a random element generator is proportional to $\log^2 |\mathbb{G}|$; see ([30] Remark 2). Then, given $0 \leq \varepsilon < 1$, Protocol 2 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. This technique reduces secure random sampling to 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.

Define probability distribution $Cube(a_1, \ldots, a_j) = \{a_1^{\epsilon_1} \cdots a_j^{\epsilon_j} : (\epsilon_1, \ldots, \epsilon_j) \in_R \{0,1\}^j\}$, referred to as a random cube of length j, for $a_1, \ldots, a_j \in \mathbb{G}$. Given a generating set $\mathcal{G}$ of $\mathbb{G}$, Dixon's theorem shows how to construct a random cube of length proportional to $\log |\mathbb{G}|$ that is $1/4$-uniform with a given probability.

Theorem 2 ([30] 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 $j \geq d + K_\delta \log |\mathbb{G}|$.*

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 the uniform distribution U on $\mathbb{G}$ w.r.t. to the statistical (or variational) distance $\Delta[W; U] = \frac{1}{2} \sum_{a \in \mathbb{G}} |W(a) - U(a)| = \max_{A \subset \mathbb{G}} |W(A) - U(A)|$. Ref. [30] (Lemma 13(b), page 11) describes the procedure in detail.

Theorem 3 ([30] 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*

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

To illustrate this approach, we apply this result to the Rubik's Cube group $\mathbb{G}$ of order $\approx 2^{65}$. Assume that we have generated a 1/4-uniform random cube W using Theorem 2. In practice, this means that we construct a long random cube W and apply a statistical test to see if W is sufficiently large. Once this pre-processing step is completed, we can apply Theorem 3 with $\varepsilon = 1/8$, to generate, with probability at least $1 - 2^{-10}$, a new 2^{-k}-uniform cube Z_j of length $j \approx 0.83(10 + 2k + 2 \cdot 65.23)$ group elements. Choosing $k = 20$ requires a random cube of length $j = 150$ group elements.

Taking advantage of the particular structure of the Rubik's Cube group as reflected by the linear representation of Section 5.2, a more efficient for uniformly random sampling runs as follows. A random element is constructed by first generating a random permutation of the 12 edge cubies, along with 12 binary edge orientation indicators, with values of +1 or -1 such that the product of all 12 indicators is +1. Similarly, a permutation for the 8 corner cubies is randomly generated, with their orientations selected from 1, 2, 4 modulo 7 subject to the condition that the product of all 8 orientations is 1 modulo 7. Both permutations must also have the same parity.

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.

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

1: $[\![r_0]\!], \ldots, [\![r_{j-1}]\!] \leftarrow$ random-bits(j)
2: $[\![g_i^{xr_i}]\!]_{\mathbb{G}} \leftarrow$ if–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, $x \in \mathbb{Z}$

To conclude this section, we compare Dixon's technique with two heuristics used in practice: the product replacement algorithm [31] and Prospector [32]. In the product replacement algorithm, the state is an array a of length n elements that generate the group. The product replacement algorithm repeatedly replaces the ith coefficient a_i by $a_i * {a_j}^{\pm 1}$ or ${a_j}^{\pm 1} * a_i$ for random $1 \leq i, j \leq n$ and $j \neq i$. After a given number of steps, a random element of a is then returned. Similar to Protocol 2 and given a pre-processed public array a, returning a random coefficient in MPC requires securely sampling a random unit vector of length n, r, and performing $\prod_i$ if–else$([\![r_i]\!], a_i, 1_{\mathbb{G}})$. The required length of a is polynomial in $\log|\mathbb{G}|$ [33], but precise bounds on the length of the array and corresponding probability of uniformly random elements is an open problem.

The Prospector algorithm is an extension of product replacement and produces elements that have a close to uniform distribution when testing for a predetermined property of the group element. Heuristically, Prospector is shown to output public arrays of very short length. Outputs are represented by so-called straight line programs (SLPs), a sequence of elements of $\mathbb{G}$, a, such that, given a generating set $\mathcal{G}$, every new element either belongs to $\mathcal{G}$, which is the inverse of a preceding element, or the product of two preceding elements. The aim is to produce small SLPs (trading off algorithm efficiency).

Prospector tests newly produced elements for randomness. This is completed by producing batches of test data using a map $t : \mathbb{G} \to \mathbb{N}$ and applying the χ^2 test. The Prospector paper ([32] Tables 1–6) demonstrates that it can produce SLPs of length ≤ 100 for large groups such as $Sym(1000)$ and $SL(150, 11^3)$.

Using Prospector to create SLPs in MPC requires implementing secure protocols for the map t to produce test data. Their efficiency depends on the group property used for the statistical test. We leave the potential of this approach as further research.

6.3. Secure Inversion

The sampling protocols from Section 6.2 allow us to invert secure group elements for groups with known or unknown order. This requires (close to) uniform sampling of random elements in the secure group. Protocol 3 implements this functionality following the classical approach from [2].

Protocol 3 inversion($[\![a]\!]_{\mathbb{G}}$)

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** $[\![a^{-1}]\!]_{\mathbb{G}}$

6.4. 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 completed sequentially. The protocol can be optimized in many ways. For instance, it is more efficient to compute the bits of $[\![x]\!]$ all at once, including the sign bit, using a standard solution.

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

1: $[\![b]\!]_{\mathbb{G}}, [\![y]\!] \leftarrow \mathsf{if\text{–}else}([\![x<0]\!], (\mathsf{inversion}([\![a]\!]_{\mathbb{G}}), -[\![x]\!]), ([\![a]\!]_{\mathbb{G}}, [\![x]\!]))$ ▷ skip if $x \geq 0$ ensured
2: $[\![c]\!]_{\mathbb{G}} \leftarrow [\![1]\!]_{\mathbb{G}}$
3: **for** $i = 0$ **to** $\ell - 1$ **do** ▷ $\mathrm{len}(x) \leq \ell$ assumed
4: $\quad [\![e_i]\!] \leftarrow \mathsf{lsb}([\![y]\!])$
5: $\quad [\![c]\!]_{\mathbb{G}} \leftarrow \mathsf{if\text{–}else}([\![e_i]\!], [\![b]\!]_{\mathbb{G}} * [\![c]\!]_{\mathbb{G}}, [\![c]\!]_{\mathbb{G}})$
6: $\quad [\![y]\!] \leftarrow ([\![y]\!] - [\![e_i]\!])/2$
7: $\quad [\![b]\!]_{\mathbb{G}} \leftarrow [\![b]\!]^2_{\mathbb{G}}$
8: **return** $[\![c]\!]_{\mathbb{G}}$

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 completed in parallel. Using standard techniques for constant rounds protocols (see, e.g., [2,22]), the overall round complexity can be established independent of ℓ.

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

1: $[\![x_0]\!], \ldots, [\![x_{\ell-1}]\!] \leftarrow \mathsf{bits}([\![x]\!])$ ▷ $\mathrm{len}(x) + 1 \leq \ell$ assumed for two's complement
2: $[\![r]\!]_{\mathbb{G}}, [\![r^{-1}]\!]_{\mathbb{G}}, \ldots, [\![r^{-2^{\ell-1}}]\!]_{\mathbb{G}} \leftarrow \mathsf{random}(\mathbb{G}, 1, -1, \ldots, -2^{\ell-1})$
3: $b \leftarrow [\![a]\!]_{\mathbb{G}} * [\![r]\!]_{\mathbb{G}}$ ▷ $b = a * r$
4: **for** $i = 0$ **to** $\ell - 1$ **do**
5: $\quad [\![c_i]\!]_{\mathbb{G}} \leftarrow \mathsf{if\text{–}else}([\![x_i]\!], b^{2^i} * [\![r^{-2^i}]\!]_{\mathbb{G}}, 1_{\mathbb{G}})$ ▷ in parallel
6: **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. We can also obtain more 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 6 exponentiation($[\![a]\!]_{\mathbb{G}}, x$) public exponent x, $\mathbb{G}$ abelian

1: $[\![r]\!]_{\mathbb{G}}, [\![r^{-x}]\!]_{\mathbb{G}} \leftarrow \mathsf{random}(\mathbb{G}, 1, -x)$
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 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 $[\![x]\!]$.

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

1: $[\![r]\!], [\![a^{-r}]\!]_{\mathbb{G}} \leftarrow \mathsf{random\text{-}pair}(a)$
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** $[\![a^x]\!]_{\mathbb{G}}$

7. Specific Constructions of Secure Groups

In this section, we present specific constructions for three types of number-theoretic groups commonly used in cryptography. The first type are quadratic residue groups (more generally, Schnorr groups), allowing for a direct secure representation with secret shares in the surrounding finite field. The second type are elliptic curve groups, where we will apply secret-sharing coordinatewise over the curve's field of definition. The third type are class groups, where we will apply secret-sharing to the components of binary quadratic forms in combination with secure integer arithmetic.

7.1. Secure Quadratic Residue Groups

The first specific type of group that we consider is $QR_p = (\mathbb{Z}_p^*)^2$, the group of quadratic residues modulo an odd prime p. Quadratic residue groups, and, more generally, subgroups of $\mathbb{F}_q^*$ (Schnorr groups), are used widely in cryptography, where $n = |QR_p| = (p-1)/2$ is chosen sufficiently large to ensure that the discrete log problem is hard. Often, n is assumed to be prime as well.

Secure group schemes for $\mathbb{F}_q^*$ and all its subgroups are easily obtained using $[\![a]\!]_{\mathbb{G}} = [\![a]\!]_q$ as secure representation for $a \in \mathbb{F}_q^*$. Protocols for all tasks listed in Definition 1 can be obtained with standard techniques, except for the task of secure en/decoding. To enable integer-valued inputs and outputs for applications of secure groups, we often need encoding functions $\sigma\colon S \to \mathbb{G}$ with $S \subset \mathbb{Z}$. Efficient en/decoding is hard for arbitrary subgroups of $\mathbb{F}_q^*$, but, for QR_p, efficient solutions are possible.

Many encodings (or embeddings) for $\mathbb{G} = QR_p$ have been proposed in the cryptographic literature. For our purposes, we consider the following four encodings for $\mathbb{G}$:

$$\begin{aligned}
&\sigma_1\colon \{1,\dots,n\} \to \mathbb{G}, && s \mapsto s^2 \bmod p\\
&\sigma_2\colon \{1,\dots,n\} \to \mathbb{G}, && s \mapsto \begin{cases} s, & \text{if } s \in \mathbb{G}\\ p-s, & \text{if } s \notin \mathbb{G}\end{cases}\\
&\sigma_3\colon \{0,\dots,\lfloor p/k\rfloor - 1\} \to \mathbb{G}, && s \mapsto \min(\mathbb{G} \cap \{ks + i\colon 0 \le i < k\})\\
&\sigma_4\colon \{1,\dots,n\} \to \mathbb{G}, && s \mapsto H(s, \arg\min_i\{i \ge 1\colon H(s,i) \in \mathbb{G}\}),
\end{aligned}$$

where $1 < k < p$ and H is a cryptographic hash function with codomain $\mathbb{Z}_p^*$.

Encoding σ_1 is the natural encoding for $\mathbb{G}$. Since squaring is a 2-to-1 mapping on $\mathbb{Z}_p^*$ as $s^2 = (-s)^2$, it follows that σ_1 is a bijective encoding on $\{1,\dots,n\}$. Decoding of $a \in \mathbb{G}$ amounts to taking the unique modular square root of $a \le n$.

For $p \equiv 3 \pmod 4$, σ_2 is another bijective encoding for $\mathbb{G}$, generalizing the encoding defined for safe primes p in ([34] Section 4.2, Example 2). We see that $\sigma_2(s) = p - s$ for $s \notin \mathbb{G}$ is indeed a quadratic residue modulo p because $p - s \equiv (-1)s \pmod p$ is the product of two quadratic nonresidues. Decoding of $a \in \mathbb{G}$ amounts to $\sigma_2^{-1}(a) = a$ if $a < p/2$ and $\sigma_2^{-1}(a) = p - a$ otherwise.

Encoding σ_3 resembles an encoding introduced by Koblitz in the context of elliptic curve cryptosystems ([35] Section 3.2). The value of $\sigma_3(s)$ is well-defined as long as each interval $\{ks + i \colon 0 \leq i < k\}$ contains a quadratic residue. This can be ensured by picking k sufficiently large as a function of p. The classical result by Burgess [36] implies that $k = \lceil \sqrt{p} \rceil$ ensures successful encoding for sufficiently large p (see also [37]). Under the extended Riemann Hypothesis, Ankeny [38] proved that the least quadratic nonresidue for prime p equals $O(\log^2 p)$.

In practice, we may set parameter $k = \lceil \log_2 p \rceil$ or a small multiple thereof. Probabilistic heuristics for small primes (<20 bits) suggest that the greatest number of consecutive quadratic nonresidues in the average case is fit by $\log_2(p)$. Buell and Hudson [39] computed the lengths of the longest sequences of consecutive residues and nonresidues. By numerical analysis, the authors find that the longest sequence of residues and nonresidues for a given prime p is fitted very closely by $\log_2(p) + \delta$ with δ slightly larger than 1, which suggests a choice of k for an average-case selection of modulus p. A small dataset from [37] using smaller primes (http://www.math.caltech.edu/people/hummel.html, accessed on 20 September 2023) shows sequences of length $< 2(\log(p) + \delta)$ in the worst case. Heuristics for larger primes (say $\geq$ 256-bit) are computationally heavy and beyond the scope of this paper.

In general, encoding σ_3 is not bijective. Decoding of any a in the range of σ_3 is simple as $\sigma_3^{-1}(a) = \lfloor a/k \rfloor$. Note that encoding σ_3 is not constant time and may leak sensitive information in certain applications.

Finally, encoding σ_4 corresponds to a hash function used in certain elliptic curve signature schemes ([40] Section 3.2). Since $\Pr[H(s,i) \in \mathbb{G}] = \frac{1}{2}$ in the random oracle model, computing $\sigma_4(s)$ requires two hashes and two Legendre symbols on average. The collision resistance of H implies that the encoding will be "computationally injective" in the sense that it is infeasible to find $s \neq s'$ for which $\sigma_4(s) = \sigma_4(s')$. Due the one-wayness of H, however, decoding of any a in the range σ_4 amounts to an exhaustive search.

For our purposes, we are interested in secure computation of these encodings and decodings. We briefly compare the performance for the four encodings. A secure encoding $[\![\sigma_1(s)]\!]$ amounts to a secure squaring of $[\![s]\!]$, which is very efficient. Secure decoding $[\![\sigma_1^{-1}(a)]\!]$ requires taking the modular square root of a quadratic residue $[\![a]\!]$. This can be completed efficiently by multiplying $[\![a]\!]$ with a uniformly random square $[\![r]\!]^2$, opening the result ar^2, taking a square root $a^{1/2}r$ (in the clear) and dividing this by $[\![r]\!]$. Finally, we need one secure comparison to make sure that the result is in $\{1, \ldots, n\}$.

Similarly, a secure encoding $[\![\sigma_2(s)]\!]$ amounts to securely evaluating the Legendre symbol $[\![(s \mid p)]\!]$. Since s is known to be nonzero, we simply multiply $[\![s]\!]$ with a uniformly random square $[\![r]\!]^2$ and also with $[\![1 - 2b]\!] = [\![(-1)^b]\!]$ for a uniformly random bit b, opening the result $sr^2(-1)^b$, for which we compute the Legendre symbol $z = (sr^2(-1)^b \mid p)$ in the clear. Then, $[\![(s \mid p)]\!] = z[\![(-1)^b]\!]$. Secure decoding boils down to a secure comparison with public $(p-1)/2$, which is quite efficient.

A secure encoding $[\![\sigma_3(s)]\!]$ amounts to the secure evaluation of k Legendre symbols $[\![(ks + i \mid p)]\!]$ and finding the first $+1$ among these. With k of the order $\log_2 p$, this represents a considerable amount of work. However, secure decoding $[\![\sigma_3^{-1}(a)]\!] = [\![\lfloor a/k \rfloor]\!]$ is much more efficient, especially if k is a power of two.

Finally, σ_4 is not practical to compute obliviously. To obliviously select the smallest i such that $[\![(H(s,i) \mid p)]\!] = +1$ requires computing $[\![H(s,i)]\!]$ for $0 \leq i < |\mathbb{G}|$, which is not practical. Also, σ_4 does not allow efficient decoding due to the one-wayness of H. However, this encoding is still useful to map private inputs $s \in \{1, \ldots, n\}$ to (unique) secret-shared group elements $[\![\sigma_4(s)]\!]_{\mathbb{G}}$.

7.2. Secure Elliptic Curve Groups

Let $E(\mathbb{F}_q)$ denote the finite group of points on an elliptic curve group E over $\mathbb{F}_q$. For cryptographic purposes, we often use a subgroup $\mathbb{G} \subseteq E(\mathbb{F}_q)$ of large prime order, such that the discrete log problem and the (decisional) Diffie–Hellman problem are hard in $\mathbb{G}$.

As secure representation of an affine point $P = (x, y) \in \mathbb{G}$, we take $[\![P]\!]_{\mathbb{G}} = ([\![x]\!], [\![y]\!])$, where it is left understood that x and y are secret-shared over $\mathbb{F}_q$. For efficient implementation of the secure group operation, it is advantageous to use complete formulas, that is, group law formulas without any exceptions. Complete formulas are known for groups of prime-order Weierstrass curves [41] and Edwards curves [3,42].

As an example, the complete formula for the group operation $P_1 + P_2$ on a twisted Edwards curve is provided by

$$\left(\frac{x_1 y_2 + y_1 x_2}{1 + d x_1 y_1 x_2 y_2}, \frac{y_1 y_2 - a x_1 x_2}{1 - d x_1 y_1 x_2 y_2} \right),$$

where $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are arbitrary points on the curve. In particular, one can see that $(0, 1)$ acts as the identity element. This way, it is straightforward to compute $[\![P_1 + P_2]\!]_{\mathbb{G}}$ given $[\![P_1]\!]_{\mathbb{G}}$ and $[\![P_2]\!]_{\mathbb{G}}$. The number of secure operations over $\mathbb{F}_q$ is low, like at most 7 secure multiplications (and 2 secure inversions). The multiplicative depth is, however, at least 3.

Using twisted Edwards curves with extended coordinates (and $a = -1$), the result from Hisil et al. ([3] Section 4.2) yields a multiplicative depth of 2 for secure curve point addition, performing four multiplications in parallel in each round. Protocol 8 is based on their complete formula. For secure point doubling, this protocol can be optimized, essentially replacing the four multiplications in the first round by four squarings.

Protocol 8 add($[\![P_1]\!]_{\mathbb{G}}, [\![P_2]\!]_{\mathbb{G}}$)

1: $([\![x_1]\!], [\![y_1]\!], [\![t_1]\!], [\![z_1]\!]) \leftarrow [\![P_1]\!]_{\mathbb{G}}$
2: $([\![x_2]\!], [\![y_2]\!], [\![t_2]\!], [\![z_2]\!]) \leftarrow [\![P_2]\!]_{\mathbb{G}}$
3: $[\![r_1]\!], [\![r_2]\!], [\![r_3]\!], [\![r_4]\!] \leftarrow [\![y_1]\!] - [\![x_1]\!], [\![y_2]\!] - [\![x_2]\!], [\![y_1]\!] + [\![x_1]\!], [\![y_2]\!] + [\![x_2]\!]$
4: $[\![r_1]\!], [\![r_2]\!], [\![r_3]\!], [\![r_4]\!] \leftarrow [\![r_1]\!][\![r_2]\!], [\![r_3]\!][\![r_4]\!], 2d[\![t_1]\!][\![t_2]\!], 2[\![z_1]\!][\![z_2]\!]$ ▷ 4M
5: $[\![r_1]\!], [\![r_2]\!], [\![r_3]\!], [\![r_4]\!] \leftarrow [\![r_2]\!] - [\![r_1]\!], [\![r_4]\!] - [\![r_3]\!], [\![r_4]\!] + [\![r_3]\!], [\![r_2]\!] + [\![r_1]\!]$
6: $[\![x_3]\!], [\![y_3]\!], [\![t_3]\!], [\![z_3]\!] \leftarrow [\![r_1]\!][\![r_2]\!], [\![r_3]\!][\![r_4]\!], [\![r_2]\!][\![r_3]\!], [\![r_1]\!][\![r_4]\!]$ ▷ 4M
7: $[\![P_3]\!]_{\mathbb{G}} \leftarrow ([\![x_3]\!], [\![y_3]\!], [\![t_3]\!], [\![z_3]\!])$
8: **return** $[\![P_3]\!]_{\mathbb{G}}$

We illustrate en/decoding for secure elliptic curve groups with an example similar to encoding σ_3 of Section 7.1. (The equivalent of σ_4 of Section 7.1 for elliptic curves is often referred to as hash to curve; see the IETF draft for Hashing to Elliptic Curves [43] for detailed specifications.) We conclude this section with an alternative en/decoding based on [44], which mitigates some of the concerns of σ_3 and σ_4 for elliptic curves at the cost of performance loss in MPC.

Consider a twisted Edwards curve $E(\mathbb{F}_p)$ with curve equation $E : ax^2 + y^2 = 1 + dx^2y^2$ for given $a, d \in \mathbb{F}_p$. For encoding input s given parameter k, we repeatedly set $x = sk + i$ for varying i and test if x corresponds to a valid point $(x, y) \in E(\mathbb{F}_p)$, which amounts to testing if $u = (1 - ax^2)/(1 - dx^2)$ is a quadratic residue modulo p. Then, we define $\sigma(s) = (x, y)$, where y is a square root of u modulo p.

Secure decoding amounts to recovering $s = \lfloor x/k \rfloor$. In general, secure decoding can be optimized if k is a power of 2. Moreover, if increment i is also available as an auxiliary secret-shared input $[\![i]\!]$, secure decoding may be implemented basically for free as $[\![s]\!] = ([\![x]\!] - [\![i]\!])/k$.

Testing $O(k)$ times for quadratic residuosity may become a bottleneck. Note that, if the application requires constant-time encoding, one should test the full range of size k. Also note that many modern elliptic curve implementations with simple formulas do not

provide a prime-order group but a group with a small cofactor h, usually $h = 4$ or 8. This introduces the risk that this encoding leaks information, e.g., when an attacker creates a point whose order divides h. A defense is to multiply points by h and abort the protocol if the result is the identity.

For odd-characteristic elliptic curves with a point of order 2, Elligator [44] presents a bijective map that is constant-time and avoids $O(k)$ quadratic residuosity tests. The cost in MPC of the forward map $\sigma : \mathbb{F}_p \to E(\mathbb{F}_p)$ is dominated by the Legendre symbol and the modular square root. The reverse map requires a comparison and two modular square roots. Choosing between naive hash to curve or Elligator is a security and efficiency trade-off that depends on the application.

7.3. Secure Class Groups

We now 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 [45], as presented by Cohen ([46] Algorithm 5.4.7). We slightly adapt the algorithm, skipping some case distinctions that were introduced for efficiency; see Algorithm 2. Apart from the computationally nontrivial reduction performed in the final step of the algorithm, the resulting algorithm only requires two xgcds and one integer division. This compares favorably with alternatives such as the classical composition algorithm by Dirichlet [47] (see also ([48] Lemma 3.2) and ([49] Algorithm 6.1.1)).

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 $f_2 = (a_2, b_2, c_2)$
1: $s \leftarrow (b_1 + b_2)/2$
2: $d', x_1, y_1 \leftarrow \mathsf{xgcd}(a_1, a_2)$ ▷ $x_1 a_1 + y_1 a_2 = d' = \gcd(a_1, a_2)$
3: $d, x_2, y_2 \leftarrow \mathsf{xgcd}(s, d')$ ▷ $x_2 s + y_2 d' = \gcd(s, d')$
4: $v_1, v_2 \leftarrow a_1/d, a_2/d$
5: $r \leftarrow (y_1 y_2 (s - b_2) - x_2 c_2) \bmod v_1$ ▷ integer division
6: $a_3 \leftarrow v_1 v_2$
7: $b_3 \leftarrow b_2 + 2v_2 r$
8: $c_3 \leftarrow (b_3^2 - \Delta)/(4a_3)$
9: $f_3 = (a_3, b_3, c_3)$
10: **return** reduce(f_3)

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 ([4] Algorithm 5.3)) runs in at most $2 + \lceil \log_2(a/\sqrt{\Delta}) \rceil$ steps ([4] 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 ([12] Algorithm 3) as our starting point. (Recently, Ref. [50] 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 $|b| \leq \sqrt{|\Delta|}$ 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.

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

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

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 [51] (compare to [52] 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.

Let k be our parameter as above. Given a worst-case prime gap for an ℓ-bit discriminant, gap_ℓ, we avoid the need to return the distance by setting parameter $k = gap_\ell$ and mapping input a to a prime $a' = a \cdot k + i$ for some $0 \leq i < k$. Algorithm 4 is similar to searching for candidates by incrementing i in encoding σ_3 of Section 7.1. After a successful encoding of a' per Algorithm 4 to a so-called prime form $f_{a'} = (a', b, c)$, we can discard the distance knowing that $a = \lfloor a'/gap_\ell \rfloor$.

Algorithm 4 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

We improve upon the encoding suggested by [51]. We do not search for a prime a, which requires a costly primality test. Instead, we take $a \equiv 3 \pmod 4$ and simply test whether $b^2 \equiv \Delta \pmod a$ for $b = \Delta^{\frac{a+1}{4}} \bmod a$. This condition will certainly hold if a is prime and Δ is a quadratic residue modulo a because then we have $b^2 \equiv \Delta^{(a+1)/2} \equiv \Delta^{(a-1)/2}\Delta \equiv \Delta \pmod a$. Hence, the success rate will be no worse than for [51].

For $|\Delta|$, an odd prime, the frequency of prime forms (a', b, c) approximately corresponds to the quadratic residuosity of a' modulo $|\Delta|$. We refer to ([4] Proposition 3.4.5) for the exact frequency of prime forms.

Finally, we briefly discuss random sampling. Even though class groups are abelian, random sampling using a generating set is inefficient due to the complexity of computing the generating set for $Cl(\Delta)$. Given a (sufficiently complete) generating set $\mathcal{G}$ of the relevant (sub)group $\mathbb{G}$, without knowledge of the order $Cl(\Delta)$, the general idea is for $g_i \in \mathcal{G}$ and random $e_i \in \{1, \ldots, |\Delta|\}$, for $i \in \{0, \ldots, d-1\}$, to compute a random element $\prod_{i=0}^{d-1} g_i^{e_i}$, e.g., using techniques from Lenstra and Pomerance [53].

8. MPC-Based Threshold Cryptosystems

As a concrete example of MPC-based threshold cryptography, we consider the well-known threshold ElGamal cryptosystem [54]. Given a secure group for an MPC setting with m parties and a corruption threshold of t, $0 \leq t < m/2$ (cf. Section 2), we obtain the following scheme for the semi-honest case.

Example 1. *A (t, m)-*threshold ElGamal cryptosystem *is constructed as follows, given a secure group scheme for a cyclic group $\mathbb{G} = \langle g \rangle$ of large prime order p.*

Distributed key generation. *The parties generate $[\![x]\!]$ with $x \in_R \mathbb{Z}_p$ and use secure exponentiation to compute $h = g^x$. The parties keep private key $[\![x]\!]$ in shares and output public key h.*

Encryption. *Given message $M \in \mathbb{G}$, pick $u \in_R \mathbb{Z}_p$. The ciphertext for public key h is the pair $(g^u, h^u M)$.*

Threshold decryption. *Given ciphertext (A, B), the parties use $[\![x]\!]$ to compute A^x by secure exponentiation. The parties output message $M = B/A^x$.*

The complexity of the protocols for distributed key generation and threshold decryption is entirely hidden in the underlying MPC framework supporting secure groups.

We may extend this basic threshold ElGamal cryptosystem as follows, noting that, instead of using message $M \in \mathbb{G}$ in the clear, we can also use $[\![M]\!]_{\mathbb{G}}$ in shares, just as we keep the private key $[\![x]\!]$ in shares.

Example 2. *The threshold ElGamal cryptosytem of Example 1 is extended as follows to handle secret-shared messages $[\![M]\!]_{\mathbb{G}}$.*

Encryption of a shared message. *Given message $[\![M]\!]_{\mathbb{G}}$, the parties generate $[\![u]\!]$ with $u \in_R \mathbb{Z}_p$ and output the pair $(g^{[\![u]\!]}, h^{[\![u]\!]}[\![M]\!]_{\mathbb{G}})$ as ciphertext for public key h. Here, secure expo-*

nentiation is used twice, either with public output (A, B) *or with secret-shared output* $([\![A]\!]_{\mathbb{G}}, [\![B]\!]_{\mathbb{G}})$.

Threshold decryption to a shared message. *Given ciphertext* (A, B)*, the parties compute* $[\![A^x]\!]_{\mathbb{G}} = A^{[\![x]\!]}$ *using secure exponentiation. The parties compute and keep message* $[\![M]\!]_{\mathbb{G}} = B / [\![A^x]\!]_{\mathbb{G}}$ *in shares. Similarly, the parties may decrypt a ciphertext* $([\![A]\!]_{\mathbb{G}}, [\![B]\!]_{\mathbb{G}})$.

Combining these two protocols, we can conduct things like reencryption of a given ciphertext for another public key: use a threshold decryption to a shared message followed by an encryption of the shared message under the other public key. As a final example, we show a direct way to perform reencryption.

Example 3. *The threshold ElGamal cryptosytem of Example 1 is extended as follows to support efficient reencryption.*

Reencryption. *Assume the parties hold shares for private keys* $[\![x_1]\!]$ *and* $[\![x_2]\!]$*, and then they may compute* $[\![x_1 / x_2]\!]$ *and use this with secure exponentiation to convert ciphertext* (A, B) *for public key* $h_1 = g^{x_1}$ *into ciphertext* $(A^{[\![x_1/x_2]\!]}, B)$ *for public key* $h_2 = g^{x_2}$.

Reencryption key. *Assume the parties hold shares for private keys* $[\![x_1]\!]$ *and* $[\![x_2]\!]$*, and then they may compute and open* $[\![x_1 / x_2]\!]$ *as a reencryption key.*

9. Conclusions

We have integrated secure groups as defined in this paper in the Python package MPyC [55], including support for symmetric groups, quadratic residues, Schnorr groups, various elliptic curve groups (Weierstrass, Edwards), and class groups. The secure groups implementation builds on the secure integer arithmetic provided by MPyC, which covers basic operations like $+, *, <$ as well as more advanced operations such as secure integer division with remainder and secure computation of the bit length. In particular, the implementation includes our protocol for the extended gcd, which is of independent interest, and many of the specific protocols of Section 7. To demonstrate applications of secure groups in threshold cryptography, a demo for threshold ElGamal (cf. Examples 1 and 2) is provided ([55] `elgamal.py`). Also, a demo for threshold DSA (Digital Signature Algorithm) is provided ([55] `dsa.py`).

In a separate repository [56], we provide implementations of verifiable MPC built using secure groups. In verifiable MPC, the parties generate publicly verifiable proofs of correctness for the results of a multiparty computations. Even in the extreme case when all parties are corrupt, these proofs remain unforgeable. Technically, we extend a multiparty computation with the generation of these noninteractive proofs in terms of secure groups, covering, e.g., compressed Σ-protocols [57]. We also cover succinct noninteractive arguments of knowledge (SNARKs) as used in Trinocchio [58].

Author Contributions: B.S. and T.S. contributed equally to the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 780477 (PRIViLEDGE).

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: We thank Alessandro Danelon, Mark Abspoel, Niek Bouman, Thomas Attema, and the CSCML 2023 reviewers for their valuable comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Schoenmakers, B.; Segers, A.J.M. Efficient Extended GCD and Class Groups from Secure Integer Arithmetic. In *Proceedings of the Cyber Security, Cryptology, and Machine Learning—7th International Symposium, CSCML 2023, Be'er Sheva, Israel, 29–30 June 2023*; Lecure Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2023; Volume 13914, pp. 32–48. [CrossRef]
2. Bar-Ilan, J.; Beaver, D. Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction. In Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, AB, Canada, 14–16 August 1989; pp. 201–209. [CrossRef]
3. Hisil, H.; Wong, K.K.; Carter, G.; Dawson, E. Twisted Edwards Curves Revisited. In *Proceedings of the Advances in Cryptology—ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, 7–11 December 2008*; Pieprzyk, J., Ed.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5350, pp. 326–343. [CrossRef]
4. Buchmann, J.A.; Vollmer, U. *Binary Quadratic Forms—An Algorithmic Approach*; Algorithms and Computation in Mathematics; Springer: Berlin/Heidelberg, Germany, 2007; Volume 20.
5. Wesolowski, B. Efficient Verifiable Delay Functions. In *Proceedings of the Advances in Cryptology—EUROCRYPT 2019—38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, 19–23 May 2019*; Ishai, Y., Rijmen, V., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11478, pp. 379–407. [CrossRef]
6. Block, A.R.; Holmgren, J.; Rosen, A.; Rothblum, R.D.; Soni, P. Time- and Space-Efficient Arguments from Groups of Unknown Order. In *Proceedings of the Advances in Cryptology—CRYPTO 2021—41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, 16–20 August 2021*; Malkin, T., Peikert, C., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2021; Volume 12828, pp. 123–152. [CrossRef]
7. Dobson, S.; Galbraith, S.; Smith, B. Trustless unknown-order groups. *Math. Cryptol.* **2022**, *1*, 25–39.
8. Bernstein, D.J.; Yang, B. Fast constant-time gcd computation and modular inversion. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2019**, *2019*, 340–398. [CrossRef]
9. Menezes, A.; van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*; CRC Press: Boca Raton, FL, USA, 1996. [CrossRef]
10. Bojanczyk, A.; Brent, R. A systolic algorithm for extended GCD computation. *Comput. Math. Appl.* **1987**, *14*, 233–238. [CrossRef]
11. Knuth, D.E. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*; Pearson Education: Reading, MA, USA, 1969; p. 646.
12. Agarwal, S.; Frandsen, G.S. A New GCD Algorithm for Quadratic Number Rings with Unique Factorization. In *Proceedings of the LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, 20–24 March 2006*; Correa, J.R., Hevia, A., Kiwi, M.A., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2006; Volume 3887, pp. 30–42. [CrossRef]
13. Smart, N.P.; Alaoui, Y.T. Distributing Any Elliptic Curve Based Protocol. In *Proceedings of the Cryptography and Coding—17th IMA International Conference, IMACC 2019, Oxford, UK, 16–18 December 2019*; Albrecht, M., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11929, pp. 342–366. [CrossRef]
14. Lindell, Y. Fast Secure Two-Party ECDSA Signing. *J. Cryptol.* **2021**, *34*, 44. [CrossRef]
15. Keller, M.; Mikkelsen, G.L.; Rupp, A. Efficient Threshold Zero-Knowledge with Applications to User-Centric Protocols. In *Proceedings of the Information Theoretic Security—6th International Conference, ICITS 2012, Montreal, QC, Canada, 15–17 August 2012*; Smith, A.D., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7412, pp. 147–166. [CrossRef]
16. Baum, C.; Damgård, I.; Orlandi, C. Publicly Auditable Secure Multi-Party Computation. In *Proceedings of the Security and Cryptography for Networks—9th International Conference, SCN 2014, Amalfi, Italy, 3–5 September 2014*; Abdalla, M., Prisco, R.D., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8642, pp. 175–196. [CrossRef]
17. Veeningen, M. Pinocchio-Based Adaptive zk-SNARKs and Secure/Correct Adaptive Function Evaluation. In *Proceedings of the Progress in Cryptology—AFRICACRYPT 2017—9th International Conference on Cryptology in Africa, Dakar, Senegal, 24–26 May 2017*; Joye, M., Nitaj, A., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10239, pp. 21–39. [CrossRef]
18. Ben-Or, M.; Goldwasser, S.; Wigderson, A. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, 2–4 May 1988*; Simon, J., Ed.; ACM: New York, NY, USA, 1988; pp. 1–10. [CrossRef]
19. Gennaro, R.; Rabin, M.O.; Rabin, T. Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC'98, Puerto Vallarta, Mexico, 28 June–2 July 1998*; Coan, B.A., Afek, Y., Eds.; ACM: New York, NY, USA, 1998; pp. 101–111. [CrossRef]
20. de Hoogh, S. Design of Large Scale Applications of Secure Multiparty Computation: Secure Linear Programming. Ph.D. Thesis, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2012. [CrossRef]
21. Toft, T. Primitives and Applications for Multi-Party Computation. Ph.D. Thesis, Aarhus University, Aarhus, Denmark, 2007.
22. Damgård, I.; Fitzi, M.; Kiltz, E.; Nielsen, J.B.; Toft, T. *Unconditionally Secure Constant-Rounds Multi-Party Computation for Equality, Comparison, Bits and Exponentiation*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 285–304. [CrossRef]

23. Algesheimer, J.; Camenisch, J.; Shoup, V. *Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 417–432. [CrossRef]
24. Catrina, O.; Saxena, A. Secure Computation with Fixed-Point Numbers. In Proceedings of the Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, 25–28 January 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 35–50.
25. Korzilius, S.; Schoenmakers, B. Divisions and Square Roots with Tight Error Analysis from Newton–Raphson Iteration in Secure Fixed-Point Arithmetic. *Cryptography* **2023**, *7*, 43. [CrossRef]
26. Brent, R.P.; Kung, H.T. A systolic algorithm for integer GCD computation. In Proceedings of the 1985 IEEE 7th Symposium on Computer Arithmetic (ARITH), Urbana, IL, USA, 4–6 June 1985; pp. 118–125. [CrossRef]
27. Stehlé, D.; Zimmermann, P. A Binary Recursive Gcd Algorithm. In *Proceedings of the Algorithmic Number Theory, Burlington, VT, USA, 13–18 June 2004*; Buell, D., Ed.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 411–425.
28. Serre, J. Linear representations of finite groups. In *Graduate Texts in Mathematics*; Springer: Berlin/Heidelberg, Germany, 1977; Volume 42.
29. Holt, D.; Makholm, H. What Are Some Group Representation of the Rubik's Cube Group? StackExchange Mathematics. 2016. Available online: https://math.stackexchange.com/questions/1587307/what-are-some-group-representation-of-the-rubiks-cube-group (accessed on 23 January 2020).
30. Dixon, J.D. Generating Random Elements in Finite Groups. *Electron. J. Comb.* **2008**, *15*, R94. [CrossRef] [PubMed]
31. Leedham-Green, C.; Murray, S.H. Variants of product replacement. *Contemp. Math.* **2002**, *298*, 97–104.
32. Bäärnhielm, H.; Leedham-Green, C.R. The Product Replacement Prospector. *J. Symb. Comput.* **2012**, *47*, 64–75. [CrossRef]
33. Pak, I. The product replacement algorithm is polynomial. In Proceedings of the 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, Redondo Beach, CA, USA, 12–14 November 2000; IEEE: New York, NY, USA, 2000; pp. 476–485. [CrossRef]
34. Cramer, R.; Shoup, V. Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. *SIAM J. Comput.* **2003**, *33*, 167–226. [CrossRef]
35. Koblitz, N. (Ed.) Elliptic Curve Cryptosystems. *Math. Comput*. **1987**, *48*, 203–209. [CrossRef]
36. Burgess, D. A note on the distribution of residues and non-residues. *J. Lond. Math. Soc.* **1963**, *1*, 253–256. [CrossRef]
37. Hummel, P. On consecutive quadratic non-residues: A conjecture of Issai Schur. *J. Number Theory* **2003**, *103*, 257–266. [CrossRef]
38. Ankeny, N.C. The Least Quadratic Non Residue. *Ann. Math.* **1952**, *55*, 65–72. [CrossRef]
39. Buell, D.A.; Hudson, R.H. On runs of consecutive quadratic residues and quadratic nonresidues. *BIT Numer. Math.* **1984**, *24*, 243–247. [CrossRef]
40. Boneh, D.; Lynn, B.; Shacham, H. Short Signatures from the Weil Pairing. *J. Cryptol.* **2004**, *17*, 297–319. [CrossRef]
41. Renes, J.; Costello, C.; Batina, L. Complete Addition Formulas for Prime Order Elliptic Curves. In Proceedings of the Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, 8–12 May 2016; pp. 403–428. [CrossRef]
42. Bernstein, D.J.; Lange, T. Faster Addition and Doubling on Elliptic Curves. In Proceedings of the Advances in Cryptology—ASIACRYPT 2007: 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, 2–6 December 2007; pp. 29–50. [CrossRef]
43. Faz-Hernandez, A.; Scott, S.; Sullivan, N.; Wahby, R.; Wood, C. IETF Internet Research Taskforce, CFRG Workgroup: Hashing to Elliptic Curves. 2017. Available online: https://www.ietf.org/archive/id/draft-irtf-cfrg-hash-to-curve-10.html (accessed on 23 May 2022).
44. Bernstein, D.J.; Hamburg, M.; Krasnova, A.; Lange, T. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, 4–8 November 2013*; Sadeghi, A., Gligor, V.D., Yung, M., Eds.; ACM: New York, NY, USA, 2013; pp. 967–980. [CrossRef]
45. Shanks, D. Class number, a theory of factorization, and genera. *Proc. Symp. Math. Soc.* **1971**, *20*, 41–440.
46. Cohen, H. A Course in Computational Algebraic Number Theory. In *Graduate Texts in Mathematics*; Springer: Berlin/Heidelberg, Germany, 1993; Volume 138.
47. Lejeune Dirichlet, P. Vorlesungen über Zahlentheorie, Original Published in 1863. 2nd Edition Text Published in 1871. Translation by John Stillwell: Lectures on Number Theory, History of Mathematics Source Series Volume: 16; American Mathematical Society: New York, NY, USA, 1999; ISBN 0-8218-2017-6. Available online: http://www-gdz.sub.uni-goettingen.de/cgi-bin/digbib.cgi?PPN30976923X (accessed on 20 October 2023).
48. Cox, D.A. *Primes of the Form $x^2 + ny^2$: Fermat, Class Field Theory, and Complex Multiplication*; John Wiley & Sons: Hoboken, NJ, USA, 2011; Volume 34.
49. Long, L. Binary Quadratic Forms. GitHub. 2019. Available online: https://github.com/Chia-Network/vdf-competition/blob/master/classgroups.pdf (accessed on 23 January 2020).
50. David, N.; Espitau, T.; Hosoyamada, A. Quantum binary quadratic form reduction. *Cryptol. ePrint Arch.* **2022**.
51. Schielzeth, D. Realisierung der ElGamal-Verschlüsselung in Quadratischen Zählkorpern. Master's Thesis, Technische Universität Berlin, Berlin, Germany, 2003. Available online: https://page.math.tu-berlin.de/~kant/publications/diplom/schielzeth.pdf (accessed on 20 October 2023).

52. Schaub, J. Implementiering von Public-Key-Kryptosystemen Über Imaginär-Quadratischen Ordnungen. Master's Thesis, Technische Universität Darmstadt, Fachbereich Informatik, Darmstadt, Germany, 1999.
53. Lenstra, H.W.; Pomerance, C. A rigorous time bound for factoring integers. *J. Am. Math. Soc.* **1992**, *5*, 483. [CrossRef]
54. Pedersen, T.P. A Threshold Cryptosystem without a Trusted Party. In Proceedings of the Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques, Brighton, UK, 8–11 April 1991; pp. 522–526.
55. Schoenmakers, B. MPyC Secure Multiparty Computation in Python. GitHub. 2018. Available online: https://github.com/lschoe/mpyc (accessed on 20 October 2023).
56. Schoenmakers, B.; Segers, A.J.M. Verifiable MPC. GitHub. 2022. Available online: https://github.com/toonsegers/verifiable_mpc (accessed on 20 October 2023).
57. Attema, T.; Cramer, R. Compressed Σ-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics. In *Proceedings of the Advances in Cryptology—CRYPTO 2020—40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, 17–21 August 2020*; Micciancio, D., Ristenpart, T., Eds.; Lecture Notes in Computer Science; Proceedings, Part III; Springer: Berlin/Heidelberg, Germany, 2020; Volume 12172, pp. 513–543. [CrossRef]
58. Schoenmakers, B.; Veeningen, M.; de Vreede, N. Trinocchio: Privacy-Preserving Outsourcing by Distributed Verifiable Computation. In *Proceedings of the Applied Cryptography and Network Security—14th International Conference, ACNS 2016, Guildford, UK, 19–22 June 2016*; Manulis, M., Sadeghi, A., Schneider, S., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2016; Volume 9696, pp. 346–366. [CrossRef]