An Alternative Group for Applications of ElGamal in Cryptographic Protocols

Rolf Haenni[1] Ilona Starý Kořánová[2]

Abstract: The subgroup of quadratic residues modulo a large safe prime is the most common choice in practice for the ElGamal cryptosystem. Computations in this group are simple and sufficiently efficient for at least 128 bits of security, and the DDH problem seems to be hard. In its practical application, however, this particular group has also several disadvantages, for example the relatively high cost for testing group membership or the uneven message space. In this paper, we discuss an alternative group for ElGamal, called multiplicative group of absolute values modulo a safe prime, which is isomorphic to the subgroup of quadratic residues, but with a slightly different group operation and much better properties for practical applications such as e-voting.

Keywords: ElGamal Encryption Scheme, DDH Assumption, Group Theory, Factoring Group, E-Voting Protocols, Practical Implementations

1 Introduction

In cryptographic protocol design, the ElGamal cryptosystem is a common choice for achieving confidentiality in different contexts, for example for protecting the secrecy of the submitted votes in an e-voting application. ElGamal is simple, efficient, and well understood, and its homomorphic property offers a flexible toolbox of cryptographic operations such as re-encryption or threshold decryption. In e-voting applications, homomorphic tallying and mixnets are the two most prominent approaches for achieving vote secrecy and E2E-verifiability simultaneously. Both techniques rely on the homomorphic property.

ElGamal is IND-CPA secure in groups in which the decisional Diffie-Hellman (DDH) problem cannot be solved efficiently. Since the DDH problem is simpler than the related CDH (computational Diffie-Hellman) or DL (discrete logarithm) problems, selecting an appropriate group is more delicate. In the multiplicative group $\mathbb{Z}_p^*$ of integers modulo a prime p (also denoted as the *quotient group* $(\mathbb{Z}/p\mathbb{Z})^\times$ *of units*), for example, the DDH problem can be solved efficiently using the Legendre symbol, even if solving CDH or DL is generally believed to be a hard problem. Therefore, $\mathbb{Z}_p^*$ is not a suitable group for ElGamal [2].

[1] Bern University of Applied Sciences, 2501 Biel, Switzerland, rolf.haenni@bfh.ch
[2] Univerzita Karlova, 116 36 Prague 1, Czech Republic, ilona.koranova@ff.cuni.cz

1.1 The Subgroup of Quadratic Residues

A better choice is the subgroup $\mathbb{G}_q \subset \mathbb{Z}_p^*$ of quadratic residues modulo a large *safe* prime $p = 2q+1$, which contains only half of the elements of $\mathbb{Z}_p^*$ (the quadratic residues $x^2 \bmod p$). To this day, no efficient non-quantum algorithm is known for solving DDH in $\mathbb{G}_q$ efficiently, and this is why $\mathbb{G}_q$ is often selected for ElGamal. Other options such as elliptic curves are slightly more complicated to use and less flexible in concrete applications (their main advantage comes from the shorter keys). In practical applications, however, $\mathbb{G}_q$ also has several disadvantages:

- In applications of ElGamal, where elements of $\mathbb{G}_q$ are exchanged between the participants of a cryptographic protocol, it is important to confirm the membership of each received group element, because otherwise the IND-CPA property of ElGamal is no longer guaranteed.[3] In a worst-case scenario, not checking group memberships in a single case may undermine the security of the whole application. In $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$, all integers between 1 and $p-1$ are group members, but in $\mathbb{G}_q$, the group members depend strongly on p. While 5 for example is a member of $\mathbb{G}_5 = \{1, 3, 4, 5, 9\}$, it is not a member of $\mathbb{G}_{11} = \{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\}$. Testing membership in $\mathbb{G}_q$ requires the computation of either a modular exponentiation or a Legendre symbol. In both cases, compared to testing membership in $\mathbb{Z}_p^*$, this is relatively expensive and may have an impact on the overall performance of the application (see Sect. 4 for a discussion on the cost of membership testing in different programming languages).
- A related problem of using $\mathbb{G}_q$ for ElGamal is the fact that $\mathbb{G}_q$ itself is the message space of the encryption scheme. If a general-purpose message $m \in \{0,1\}^n$ is given as a sequence of bits of length $n < \|q\|$, then m needs to be encoded into $\mathbb{G}_q$ as a preliminary step before the encryption and decoded from $\mathbb{G}_q$ as a additional step after the decryption. Several options for such an encoding exist, but since they all have their advantages and disadvantages, expert knowledge is needed for selecting the most appropriate encoding depending on the application.[4]
- One particular approach for encoding messages into $\mathbb{G}_q$ is to define an explicit mapping from the message space into corresponding elements of $\mathbb{G}_q$. Obviously, this approach only works if the size of the message space is reasonably small to allow the enumeration of all elements. In e-voting applications, for example, where the number of voting options is usually very limited, single voting options are often encoded as

[3] Implementing systematic group membership tests is a general best practice in cryptographic protocol design. Here is a current blog about this topic: `https://blog.trailofbits.com/2022/11/29/specialized-zero-knowledge-proof-failures`. We also refer to [3,4] for a related discussions on Helios and TLS backdoors.

[4] Examples of common message encodings $\Gamma : \mathbb{Z}_q \rightarrow \mathbb{G}_q$ are the following: (1) $\Gamma(m) = \left(\frac{m+1}{p}\right)(m+1) \bmod p$, (2) $\Gamma(x) = g^m \bmod p$, and (3) $\Gamma(m) = (m+1)^2 \bmod p$. In [5], these encodings are called T2, T3, and T4, respectively.

prime numbers in $\mathbb{G}_q$ and multiple voting options as corresponding products of prime numbers (examples of systems using this representation can be found in [7,8,11,12]). After decryption, individual votes are obtained from factorizing this product (under the condition that the product is smaller than p). The problem in such a system, in which p is not a fixed system parameter (for example to support different security levels), this mapping needs to be refined depending on the selected safe prime p. This is not a difficult problem, because for n voting options one could simply select the n smallest primes in $\mathbb{G}_q$, and they can be computed and stored efficiently, but it is still a bothersome complication.

- Typically, applications relying on the hardness of the discrete logarithm require the selection of one or multiple group generators. Since every element of $\mathbb{G}_q$ (except the identity element 1) generates the whole group, generators can be found easily. However, if again p is not a fixed system parameter, then the generator selection must be repeated whenever a new safe prime p is selected. In a mixnet-based e-voting application, in which N denotes the number of submitted votes (for example $N = 100'000$), the same amount N of verifiably random (independent) generators needs to be chosen as a preparatory step for proving the correctness of the shuffle. This is again not a very difficult problem [1, Appendix A.2.3], but it also complicates the implementation of ElGamal for such purposes.

To illustrate the practical difficulties of using $\mathbb{G}_q$ for ElGamal, consider the current system specification of the Swiss Post e-voting system [12, Section 3.4.2]. To cope with the fact that the system selects a different safe prime in every election, the encoding of the voting options into prime numbers of $\mathbb{G}_q$ is implemented by an object called pTable. This object needs to be known by every protocol participant, and they all need to have exactly the same object for executing the protocol. The latest protocol version properly takes care of this problem using digital signatures, but it clearly adds to the overall protocol complexity. Similar complications exist in other implementations [5].

1.2 Contribution and Paper Overview

In this paper, we propose an alternative group for ElGamal, denoted by $\mathbb{Z}_p^+ = \{1, \dots, q\}$, which eliminates all the practical disadvantages of $\mathbb{G}_q$ listed in the previous subsection. We call it *multiplicative group of absolute values modulo p*, where $p = 2q + 1$ is a safe prime as for $\mathbb{G}_q$. Its group operation is only slightly more expensive than modular multiplication, and in modular exponentiations, the small overhead is required only once. Nevertheless, we can show that $\mathbb{Z}_p^+$ and $\mathbb{G}_q$ are isomorphic, and that the isomorphism can be computed efficiently in both directions. This implies that the hardness of the DDH problem in $\mathbb{Z}_p^+$ must be the same as in $\mathbb{G}_q$.

The rest of the paper is organized as follows: in Sect. 2, we introduce the alternative group $\mathbb{Z}_p^+$ for ElGamal and prove the existence of an efficient isomorphism between $\mathbb{Z}_p^+$ and $\mathbb{G}_q$, in

Sect. 3, we review the practical problems mentioned in the previous subsection from the perspective of $\mathbb{Z}_p^+$, in Sect. 4, we discuss the cost of membership testing based on some experimental results, and in Sect. 5, we summarize our findings and formulate a general recommendation and final conclusion.

2 An Alternative DDH Secure Group for ElGamal

The motivation for finding an alternative group for ElGamal comes from a proposal in [8, Subsection 12.1] to optimize the performance of group membership testing in $\mathbb{G}_q$. The idea of the proposed optimization is to represent quadratic residues $x \in \mathbb{G}_q$ by one of their square roots

$$\sqrt{x} = \pm x^{\frac{q+1}{2}} \bmod p,$$

and to use this square root as a witness to test group membership by checking the equality $x = (\sqrt{x})^2 \bmod p$ (using a single modular multiplication). Using this technique, it turns out that group operations can be performed directly on the square roots. This observation is the starting point for the alternative group discussed in this paper.

2.1 Background on Group Theory

A group is an algebraic structure $(G, \circ, \mathrm{inv}, e)$, where G is a finite set of *group elements*, $\circ : G \times G \rightarrow G$ a binary operation called *group operation*, $\mathrm{inv} : G \rightarrow G$ a unary operation called *inverse*, and $e \in G$ a specific group element called *identity*, such that the following properties are satisfied:

- Associativity: $x \circ (y \circ z) = (x \circ y) \circ z, \forall x, y, z \in G$,
- Inverse: $x \circ \mathrm{inv}(x) = e, \forall x \in G$,
- Identity: $x \circ e = e \circ x = x, \forall x \in G$.

It is common to simply use G for referring to the whole group $(G, \circ, \mathrm{inv}, e)$. If a group G is finite, then $q = |G|$ is called *group order*. If a subset $H \subseteq G$ is closed under the group operation, i. e., if $x \circ y \in H$ for all $x, y \in H$, then $(H, \circ, \mathrm{inv}, e)$ is called a *subgroup* of G. Lagrange's theorem states that the subgroup order $|H|$ always divides the group order $|G|$.

Groups are sometimes written additively as $(G, +, -, 0)$ or multiplicatively as $(G, \cdot, ^{-1}, 1)$, depending on the nature of the group operation. In multiplicative groups, exponentiation $x = g^u$ is defined as the result of applying the group operator $u - 1$ times to a given element $g \in G$, and $g^0 = 1$ is defined to be the base case for $u = 0$. If g and x are given, then $u = \log_g x$ is the *discrete logarithm* to the base g of x. If $u = \log_g x$ exists for every $x \in G$

in a finite group of order q, then g is called a *generator*, because it generates the whole set $G = \{g^u : 0 \leq u < q\}$ of group elements. If at least one generator exists, the group is called *cyclic*. In a prime-order group, due to Lagrange's theorem, every element of $G \setminus \{1\}$ is a generator.

If a generator g of a finite cyclic group of order q is given, then computing u from a single given value $x = g^u$ is called *discrete logarithm problem* (DL), computing $z = g^{uv}$ from two given values $x = g^u$ and $y = g^v$ is called *computational Diffie-Hellman problem* (CDH), and deciding whether $w = uv \bmod q$ holds for three given values $x = g^u$, $y = g^v$, and $z = g^w$ is called *decisional Diffie-Hellman problem* (DDH). Clearly, solving DL also solves CDH and DDH, and solving CDH also solves DDH, but the converse is not true. DDH is therefore the simplest of the three problems.

In certain groups, these problems seem to be computationally hard, which means that no known algorithm solves the problem efficiently in polynomial time. In the multiplicative group $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$ of integers modulo a prime p, both DL and CDH are commonly believed to be hard, but DDH can be solved efficiently using the Legendre or Jacobi symbol. In large subgroups of $\mathbb{Z}_p^*$, however, DDH also seems to be hard. One particular case of such a subgroup is the prime-order group $\mathbb{G}_q = \{x^2 \bmod p : x \in \mathbb{Z}_p^*\}$ of quadratic residues for a safe prime $p = 2q + 1$. In this particular case, we have $|\mathbb{Z}_p^*| = 2q$ and $|\mathbb{G}_q| = q$, i. e., $\mathbb{G}_q$ contains exactly half of the elements of $\mathbb{Z}_p^*$. Group membership $x \in \mathbb{G}_q$ can be tested either by modular exponentiation $x^q \bmod p = 1$ or by computing the Legendre symbol $\left(\frac{x}{p}\right) = 1$.

2.2 The Multiplicative Group of Absolute Values Modulo p

Consider the additive group $(\mathbb{Z}_p, +, -, 0)$, where $\mathbb{Z}_p = \{0, \ldots, p-1\}$ denotes the set of non-negative integers smaller than the prime modulus p. Additions in $\mathbb{Z}_p$ are computed modulo p, which implies that the additive inverse $-x \bmod p$ (called *negation*) is equal to $p - x$ for $x \neq 0$ (trivially, -0 is equal to 0). For an odd prime $p > 2$, which implies $\mathbb{Z}_p = \{0, \ldots, 2q\}$ for some integer $q = (p-1)/2$ (not necessarily prime), we can decompose $\mathbb{Z}_p$ naturally into two disjoint sets

$$\mathbb{Z}_p^+ = \{1, \ldots, q\} \text{ and } \mathbb{Z}_p^- = \{q+1, \ldots, 2q\}$$

of *positive* and *negative* elements modulo p, respectively, with the property that $x \in \mathbb{Z}_p^+$ implies $-x \in \mathbb{Z}_p^-$ and $x \in \mathbb{Z}_p^-$ implies $-x \in \mathbb{Z}_p^+$.[5] Furthermore, we can define the *absolute value* $|x| \in \mathbb{Z}_p^+$ naturally as $|x| = x$ for $x \in \mathbb{Z}_p^+$ and $|x| = -x \bmod p$ for $x \in \mathbb{Z}_p^-$ (with the trivial case of $|0| = 0$), which is equivalent to computing $|x| = \min(x, p-x) \in \mathbb{Z}_p^+$ using one negation and one comparison.

[5] Our intuition for calling the element of the larger half of $\mathbb{Z}_p$ negative is the fact that they can be written equivalently as $\mathbb{Z}_p^- = \{p-q, \ldots, p-1\} = \{-q, \ldots, -1\}$. The full set of integers modulo p can then be written as $\mathbb{Z}_p = \{-q, \ldots, -1, 0, 1, \ldots, q\} = \{-q, \ldots, q\}$ with a natural symmetry between $\mathbb{Z}_p^+$ and $\mathbb{Z}_p^-$ around 0 (a similar idea has been used in the definition of the *absolute Rabin-function* [6, Section 6], but the context there is slightly different).

Given these ingredients, we are now ready propose an alternative group for ELGamal. For this, consider the algebraic structure $(\mathbb{Z}_p^+, \otimes, \text{inv}, 1)$ with the group operation and inverse defined as follows:

$$x \otimes y \overset{\text{def}}{=} |xy \bmod p|,$$

$$\text{inv}(x) \overset{\text{def}}{=} |x^{-1} \bmod p|.$$

Thus, compared to the operations in $\mathbb{Z}_p^*$, the overhead of the operations in $\mathbb{Z}_p^+$ is limited to the computation of the absolute value, which requires one comparison and at most one negation. Note that to minimize the overhead, for example in exponentiations, computing absolute values can always be postponed to a single ultimate step. This follows from the simple observation that

$$\begin{aligned}
|(|x|\cdot|y| \bmod p)| &= \begin{cases} |xy \bmod p|, & \text{if } x, y \in \mathbb{Z}_p^+, \\ |-xy \bmod p|, & \text{if } x \in \mathbb{Z}_p^+, y \in \mathbb{Z}_p^-, \\ |-xy \bmod p|, & \text{if } x \in \mathbb{Z}_p^-, y \in \mathbb{Z}_p^+, \\ |xy \bmod p|, & \text{if } x, y \in \mathbb{Z}_p^-. \end{cases} \\
&= |xy \bmod p|, \\
|(|x|^{-1} \bmod p)| &= \begin{cases} |x^{-1} \bmod p|, & \text{if } x \in \mathbb{Z}_p^+, \\ |-x^{-1} \bmod p|, & \text{if } x \in \mathbb{Z}_p^-. \end{cases} \\
&= |x^{-1} \bmod p|,
\end{aligned}$$

hold for all $x, y \in \mathbb{Z}_p^*$. This means that we can start with positive elements from $\mathbb{Z}_p^+$, apply the group operations (multiplication, inverse, division, exponentiation) in $\mathbb{Z}_p^*$, and ultimately map the result from $\mathbb{Z}_p^*$ back to $\mathbb{Z}_p^+$ by calculating a single absolute value. The overhead of working in $\mathbb{Z}_p^+$ is therefore always a single comparison and at most one negation, which is negligible in cryptographic applications that use considerably more expensive operations such as modular exponentiations.

Using the above property of the absolute value, it is simple to demonstrate that $(\mathbb{Z}_p^+, \otimes, \text{inv}, 1)$ satisfies the properties of a group:

- Associativity: $$\begin{aligned} x \otimes (y \otimes z) &= |x(|yz \bmod p|) \bmod p| \\ &= |x(yz \bmod p) \bmod p| = |xyz \bmod p| \\ &= |(xy \bmod p)z \bmod p| = |(|xy \bmod p|)z \bmod p| \\ &= (x \otimes y) \otimes z. \end{aligned} \qquad \blacksquare$$
- Inverse: $$\begin{aligned} x \otimes \text{inv}(x) &= |x(|x^{-1} \bmod p|) \bmod p| \\ &= |x(x^{-1} \bmod p) \bmod p| = |xx^{-1} \bmod p| = |e| = e. \end{aligned} \qquad \blacksquare$$
- Identity: $$\begin{aligned} x \otimes e &= |xe \bmod p| = |x| = x, \\ e \otimes x &= |xe \bmod p| = |x| = x. \end{aligned} \qquad \blacksquare$$

We call $\mathbb{Z}_p^+$ *multiplicative group of absolute values modulo* p. In Appendix A, a numerical example is given to demonstrate computations in $\mathbb{Z}_p^+$ and its application to ElGamal. Note that so far we have not imposed any restrictions on p other than assuming that $p > 2$ is an odd prime, and we do not know whether DL, CDH, or DDH are hard problems in this group.

Another way of proving that $\mathbb{Z}_p^+$ with the operations as defined above forms a group comes from defining $\mathbb{Z}_p^+$ as the *quotient group* $\mathbb{Z}_p^*/\mathbb{G}_2$, where $\mathbb{G}_2 = \{1, p-1\}$ denotes the trivial subgroup of $\mathbb{Z}_p^*$ of order 2. Since modular multiplication is commutative, it follows that $\mathbb{G}_2$ is a normal subgroup of $\mathbb{Z}_p^*$, which implies that $\mathbb{Z}_p^*/\mathbb{G}_2 = \{a\,\mathbb{G}_2 : a \in \mathbb{Z}_p^*\}$ with an operation defined as $(a\,\mathbb{G}_2)(b\,\mathbb{G}_2) = (ab)\,\mathbb{G}_2$ forms a group. Note that the $q = [\mathbb{Z}_p^* : \mathbb{G}_2] = \frac{p-1}{2}$ elements of $\mathbb{Z}_p^*/\mathbb{G}_2$ are the cosets $\{1, p-1\}, \{2, p-2\}, \ldots, \{q, q+1\}$ of $\mathbb{G}_2$, from which we obtain $\mathbb{Z}_p^+$ by simply selecting the smaller of the two values as coset representatives:

$$\mathbb{Z}_p^+ = \{\min(x, y) : \{x, y\} \in \mathbb{Z}_p^*/\mathbb{G}_2\} = \{1, \ldots, q\}.$$

This alternative definition of $\mathbb{Z}_p^+$, together with the above-mentioned group operation defined for the quotient group, leads directly to the group operation defined for $\mathbb{Z}_p^+$ at the beginning of this section.

2.3 Proving the Existence of an Isomorphism

To use $\mathbb{Z}_p^+$ for ElGamal, we must have good reasons to believe that DDH (and therewith CDH and DL) is a hard problem. In the special case of a safe prime $p = 2q + 1$, where q is also prime, we can demonstrate that DDH is equally hard in $\mathbb{Z}_p^+$ and $\mathbb{G}_q$ by showing that these groups are isomorphic and that the isomorphism can be computed efficiently in both directions.[6] Under this premise, an efficient DDH solver in $\mathbb{Z}_p^+$ would immediately imply an efficient DDH solver in $\mathbb{G}_q$ by applying the isomorphism forth and back. Since this conjecture is in contradiction with current beliefs that DDH is hard in the subgroup of quadratic residues, we can assume that DDH is also hard in $\mathbb{Z}_p^+$. In other words, $\mathbb{Z}_p^+$ and $\mathbb{G}_q$ are equally applicable to cryptographic applications.

Two groups G and H are called *isomorphic*, denoted by $G \cong H$, if a structure-preserving (bijective and homomorphic) mapping $\phi : G \longrightarrow H$ exists. For proving $\mathbb{Z}_p^+ \cong \mathbb{G}_q$, it is therefore sufficient to find a single candidate mapping $\phi : \mathbb{Z}_p^+ \longrightarrow \mathbb{G}_q$ (and therefore $\phi^{-1} : \mathbb{G}_q \longrightarrow \mathbb{Z}_p^+$) and to prove that the mapping is bijective and homomorphic. Our proposal is the following:

$$\phi(x) = x^2 \bmod p, \ \text{for } x \in \mathbb{Z}_p^+,$$
$$\phi^{-1}(y) = |\sqrt{y} \bmod p| = |y^{\frac{q+1}{2}} \bmod p|, \ \text{for } y \in \mathbb{G}_q.$$

[6] Note that p being a safe prime is not a necessary condition for finding an efficient isomorphism between the multiplicative group of absolute values and the group of quadratic residues modulo p. However, since q being prime ensures the absence of non-trivial subgroups, it is the most interesting case for cryptographic applications.

Note that, given this definition, the proposed mapping is efficiently computable in both directions, with essentially one modular multiplication for ϕ and one modular exponentiation for ϕ^{-1}.

A precondition for ϕ being a bijection is already met by the fact $q = |\mathbb{Z}_p^+| = |\mathbb{G}_q|$ is the order of both the domain and the codomain. What then remains to prove is that ϕ^{-1} inverts ϕ for all $x \in \mathbb{Z}_p^+$. From the fact that $x^q \equiv \pm 1 \pmod{p}$, depending on whether $x \in \mathbb{G}_q$ or $x \notin \mathbb{G}_q$, it follows that this is actually the case:

$$\begin{aligned}\phi^{-1}(\phi(x)) &= |(x^2 \bmod p)^{\frac{q+1}{2}} \bmod p| = |(x^2)^{\frac{q+1}{2}} \bmod p| \\ &= |x^{q+1} \bmod p| = |xx^q \bmod p| = |\pm x| = x. \qquad \blacksquare\end{aligned}$$

To prove that ϕ is homomorphic, we must show that $\phi(x \otimes y) = \phi(x)\phi(y) \bmod p$ holds for all $x, y \in \mathbb{Z}_p^+$:

$$\begin{aligned}\phi(x \otimes y) &= (|xy \bmod p|)^2 \bmod p \\ &= \begin{cases}(xy)^2 \bmod p, & \text{if } xy \bmod p \leq q, \\ (-xy)^2 \bmod p, & \text{if } xy \bmod p > q.\end{cases} \\ &= (xy)^2 \bmod p = (x^2 \bmod p)(y^2 \bmod p) \bmod p \\ &= \phi(x)\phi(y) \bmod p. \qquad \blacksquare\end{aligned}$$

Put together, this proves that $\mathbb{Z}_p^+ \cong \mathbb{G}_q$, and therefore we conclude that DDH is equally hard in $\mathbb{Z}_p^+$ and in $\mathbb{G}_q$. This means that we can recommend using $\mathbb{Z}_p^+$ for ElGamal without any restrictions or additional precautions. Note that due the symmetry between $\mathbb{Z}_p^+$ and $\mathbb{Z}_p^-$, a similar isomorphic group exists for $\mathbb{Z}_p^-$.

3 Discussion of Properties

If the group $\mathbb{Z}_p^+$ is used for ElGamal instead of $\mathbb{G}_q$, we benefit from the property that the new message space is compatible across different values of p in the sense that $\mathbb{Z}_p^+ \subset \mathbb{Z}_{p'}^+$ for $p < p'$. The absence of such a property for $\mathbb{G}_q$ is the main reasons for the practical disadvantages listed in Sect. 1.1. We can now review the topics from this list in the light of $\mathbb{Z}_p^+$:

- Testing membership $x \in \mathbb{Z}_p^+$ is very efficient, since only two comparisons $1 \leq x$ and $x \leq q$ are necessary (note that $1 \leq x$ is actually a signum test $\mathrm{sgn}(x) = 1$). Compared to computing $x^q \bmod p = 1$ or equivalently $\left(\frac{x}{p}\right) = 1$ for testing membership in $\mathbb{G}_q$, the cost of two comparisons is negligible, as we can see in the performance results discussed in Sect. 4.

- Since $\mathbb{Z}_p^+ = \{1, \ldots, q\}$ is a smooth message space in the sense that all consecutive elements between a lower and an upper limit are group members, the encoding of a general-purpose message $m \in \{0, 1\}^n$ of length $n < \|q\|$ is much simpler than in $\mathbb{G}_q$. In the most straightforward encoding, where the bits of m are simply interpreted as a binary number, only the special case of $m = 0$ (all n bits set to 0) is excluded by $\mathbb{Z}_p^+$. In applications where $m = 0$ is a possible message, one could either generally increase every m by 1 or substitute $m = 0$ by $m = q$. Both options can be regarded as a bijective mapping between $\mathbb{Z}_q$ and $\mathbb{Z}_p^+$. Therefore, different values for p only have an effect on the message length n, but not on the encoding of the messages itself.

- If a small message space is encoded into $\mathbb{Z}_p^+$ by defining an explicit mapping for each possible message, then this mapping can be defined independently of the choice of p. In the particular case, where the n voting options in an e-voting system are encoded as prime numbers, we can always use the exact same set of prime numbers, for example the n smallest prime numbers $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $\ldots$, p_n. Redefining this mapping when p changes is therefore no longer necessary.

- Another advantage in the same context is the increased capacity of a single message for encoding combinations of voting options by corresponding products of prime numbers, even if this is rarely a problem in practice. The advantage comes from the compactness of the encoding in the sense that the n smallest prime numbers lie closer to each other in $\mathbb{Z}_p^+$ and therefore require less bits than in $\mathbb{G}_q$. If voters can select up to $k < n$ different voting options in a k-out-of-n election, then the bit length $\|p\|$ may impose an upper limit for k depending on n. Tab. 1 shows these upper bounds $k_{\max}$ for $\|p\| \in \{2048, 3072\}$ and different values of n. The values are derived from the "worst case", in which the voter selects the k largest from the n smallest prime group members. The values given under $\ell_{\max}$ are the sizes of corresponding combined vote encodings. As one can see, the limits for $\mathbb{G}_q$ are approximately 10% smaller than the limits for $\mathbb{Z}_p^+$.

- The problem of selecting suitable group generators is much simpler in $\mathbb{Z}_p^+$, because any of the $q - 1$ values in the range $2 \leq g \leq q$ is a generator of $\mathbb{Z}_p^+$, and also of any larger group $\mathbb{Z}_{p'}^+$, with $p' > p$. The selection can therefore be fixed independently of the actual choice of p. For example, if k (non-independent) generators are needed in an application, then one could simply take $g_1 = 2$, $g_2 = 3$, $\ldots$, $g_k = k + 1$. If independent generators are needed, they must be chosen verifiably at random as for $\mathbb{G}_q$, but then they can be used universally across different groups (as long as p and p' are also picked verifiably at random).

This discussion shows that each of the practical disadvantages of using $\mathbb{G}_q$ for ElGamal turns into an advantage for $\mathbb{Z}_p^+$ at almost no cost.

	$\lVert p \rVert = 2048$ bits						$\lVert p \rVert = 3072$ bits					
	$\mathbb{G}_q$			$\mathbb{Z}_p^+$			$\mathbb{G}_q$			$\mathbb{Z}_p^+$		
n	$k_{\max}$	$\lVert p_n \rVert$	$\ell_{\max}$	$k_{\max}$	$\lVert p_n \rVert$	$\ell_{\max}$	$k_{\max}$	$\lVert p_n \rVert$	$\ell_{\max}$	$k_{\max}$	$\lVert p_n \rVert$	$\ell_{\max}$
100	(99)	11	856	(99)	10	729	(99)	11	856	(99)	10	729
200	(199)	12	1958	(199)	11	1703	(199)	12	1958	(199)	11	1703
300	176	13	2038	201	11	2046	285	13	3066	(299)	11	2765
400	167	13	2046	186	12	2039	256	13	3062	290	12	3065
500	161	13	2042	178	12	2041	245	13	3064	273	12	3066
600	157	14	2046	172	13	2037	238	14	3069	263	13	3070
700	153	14	2038	168	13	2041	232	14	3064	255	13	3064
800	151	14	2046	165	13	2047	228	14	3070	249	13	3062
900	148	14	2037	162	13	2045	224	14	3066	245	13	3070
1000	146	15	2039	159	13	2038	221	15	3070	241	13	3069
1200	143	15	2047	155	14	2037	215	15	3066	234	14	3061
1400	140	15	2040	152	14	2039	211	15	3065	229	14	3060
1600	138	15	2044	150	14	2047	207	15	3058	225	14	3061
1800	136	16	2040	147	14	2036	205	16	3069	222	14	3066
2000	134	16	2035	145	15	2034	202	16	3061	219	15	3065

Tab. 1: Prime number encoding of combined voting options in k-out-of-n elections. In cases without an upper limit for k other than $k < n$, $k_{\max} = n - 1$ is shown in parentheses (for example $k_{\max} = 99$ for $n = 100$). The values shown for $\mathbb{G}_q$ are approximate, because they depend slightly on the actual choice of p.

4 The Cost of Membership Testing

To underline our statements about the performance of different group membership tests, we conducted some experiments with different implementations of modular exponentiation and the Jacobi symbol (which is equivalent to the Legendre symbol when p is prime). The results of these experiments are shown in Tab. 2. It is interesting to observe that modular exponentiation in C (using the GMP library) is approximately 35% faster than in Java, whereas computing the Jacobi symbol in C is more than 99 times faster than in Java (using the Bouncy Castle library). Therefore, it seems that the Jacobi symbol implementation in Bouncy Castle is far from being optimal (it is only between 3 to 5 times faster than modular exponentiation). The same holds for the modular exponentiation implementation in the Javascript library verificatum-vjsc, which is more than 30 times slower than in Java and about 5 times slower than in Python.

Our measurements also show that the GMP implementation of the Jacobi symbol is the only option with negligible costs for conducting 10'000 group membership tests in $\mathbb{G}_q$. Even for 1 million membership tests, which GMP could handle approximately in 10 seconds for 2048-bits integers and in 20 seconds for 3072-bits integers, performing these tests seems not to grow into a major factor compared to other computations in corresponding applications. Note that a batch of exactly one million group elements results from the output

	$\|p\| = 2048$ bits			$\|p\| = 3072$ bits		
	$x \in \mathbb{Z}_p^+$	$x \in \mathbb{G}_q$		$x \in \mathbb{Z}_p^+$	$x \in \mathbb{G}_q$	
	$0 < x < q$	$\left(\frac{x}{p}\right) = 1$	$x^q \bmod p = 1$	$0 < x < q$	$\left(\frac{x}{p}\right) = 1$	$x^q \bmod p = 1$
C	< 1ms	98ms	23'224ms	< 1ms	186ms	72'992ms
Java	< 1ms	12'871ms	35'705ms	< 1ms	27'132ms	114'262ms
Python	< 1ms	15'447ms	243'561ms	< 1ms	34'762ms	691'568ms
Javascript	< 1ms	12'453ms	692'821ms	< 1ms	23'878ms	2'162'474ms

Tab. 2: Performance of membership testing in $\mathbb{Z}_p^+$ and $\mathbb{G}_q$ using different methods and programming languages. The measurements were conducted on a MacBook Pro (2.3 GHz 8-Core Intel Core i9) using a single-core process over a batch of 10'000 test vectors, each of which consisting of an integer x and a safe prime $p = 2q + 1$ of the required length of either 2048 or 3072 bits. For C, we used the functions `mpz_jacobi` and `mpz_powm` from the GMP library (verion 6.2.1). For Java, we used the build-in method `BigInteger::modPow` and the method `IntegerFunctions::jacobi` from the Bouncy Castle library (version 1.70). For Javascript, we used the functions `modPow` and `legendre` from Wikström's verificatum-vjsc library (version 1.1.1). And for Python, we used the `pow` and `jacobi_symbol` functions from the SymPy library (version 1.12).[7]

of a mixnet with 50'000 input ElGamal encryptions and 5 mixers (using Wikström's shuffle proof [9, 13]). In all considered cases, the cost for testing group membership in $\mathbb{Z}_p^+$ remains negligible, and it will only grow to approximately 20 milliseconds for 1 million tests.

5 Conclusion

In this paper, we have shown that the commonly used group $\mathbb{G}_q$ of quadratic residues modulo a safe prime should probably no longer be regarded as the best choice in practical applications of the ElGamal cryptosystem, which depends on the intractability of the DDH problem. We demonstrated that this group has several drawbacks, which make practical implementations more complicated, error-prone, and less efficient. Our proposal of using $\mathbb{Z}_p^+$ as an alternative group for ElGamal eliminates these drawbacks completely while preserving the intractability of the DDH problem. Therefore, to profit maximally from the advantages of $\mathbb{Z}_p^+$, we generally recommend the replacement of $\mathbb{G}_q$ by $\mathbb{Z}_p^+$ in applications of ElGamal. Existing implementations can be simplified accordingly.

[7] Note that the Python library SymPy provides three similar functions `jacobi_symbol`, `legendre_symbol`, and `is_quad_residue`, each of which with a different implementation. While `jacobi_symbol` implements an iterative version of the efficient $O(\log x \log p)$ algorithm from [10, Section 2.4.5], `legendre_symbol` and `is_quad_residue` both compute $x^{(p-1)/2} \bmod p$ if p is prime, i. e., their performance is equal to the `pow` function. Consequently, users of SymPy are likely to unintendedly pick the wrong function with sub-optimal performance. Problems like this can be be avoided when working with $\mathbb{Z}_p^+$.

A first implementation of $\mathbb{Z}_p^+$ can be found in the Java class `ZPLus.java` in the utilities sub-module of the OpenCHVote project.[8] In Version 1.3 of this library, this class replaces the implementation of $\mathbb{G}_q$ from previous versions. This replacement implied a number of simplifications at different places of the CHVote protocol specification and the OpenCHVote code base. Examples are the implementations of the algorithms GetPrimes and GetGenerators, which are now independent of any group parameters, and the removal of the algorithm GetRandomElement, which is now a special case of GetRandomInteger.

Acknowledgments

We thank Philipp Locher for his help in developing the initial idea of this approach and for providing the Javascript code to conduct the experiments with the verificatum-vjsc library. Our thanks also go to Pierrick Gaudry for pointing out the possibility of defining $\mathbb{Z}_p^+$ as a quotient group and to François Weissbaum and Florian Moser for proofreading an early draft of this paper. Finally, we would like to thank the reviewers for their valuable comments and exceptionally constructive suggestions for improvements.

Bibliography

[1] Digital signature standard (DSS). FIPS PUB 186-4, National Institute of Standards and Technology (NIST), 2013.

[2] D. Boneh. The decision Diffie-Hellman problem. In J. Buhler, editor, *ANTS-III, 3rd International Symposium on Algorithmic Number Theory*, LNCS 1423, pages 48–63, Portland, Oregon, USA, 1998.

[3] N. Chang-Fong and A. Essex. The cloudier side of cryptographic end-to-end verifiable voting: A security analysis of Helios. In W. Robertson and D. Balzarotti, editors, *ACSAC'16, 32nd Annual Conference on Computer Security Applications*, pages 324—335, Los Angeles, USA, 2016.

[4] K. Dorey, N. Chang-Fong, and A. Essex. Indiscreet logs: Diffie-Hellman backdoors in TLS. In A. Juels and P. Traynor, editors, *NDDS'17, 24th Annual Network and Distributed System Security Symposium*, San Diego, USA, 2017.

[5] M. El Laz, B. Grégoire, and T. Rezk. Security analysis of ElGamal implementations. In P. Samarati, S.De Capitani di Vimercati, M. S. Obaidat, and J. Ben-Othman, editors, *SECRYPT'20, 17th International on on Security and Cryptography*, pages 310–321, Paris, France, 2020.

[6] R. Fischlin, , and C. Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13:221–244, 2000.

[7] K. Gjøsteen. The Norwegian Internet voting protocol. In A. Kiayias and H. Lipmaa, editors, *VoteID'11, 3rd International Conference on E-Voting and Identity*, LNCS 7187, pages 1–18, Tallinn, Estonia, 2011.

[8] See `https://gitlab.com/openchvote/cryptographic-protocol`.

[8] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis. CHVote protocol specification – version 3.4. *IACR Cryptology ePrint Archive*, 2017/325, 2022.

[9] P. Locher and R. Haenni. A lightweight implementation of a shuffle proof for electronic voting systems. In E. Plödereder, L. Grunske, E. Schneider, and D. Ull, editors, *INFORMATIK 2014, 44. Jahrestagung der Gesellschaft für Informatik*, number P-232 in Lecture Notes in Informatics, pages 1391–1400, Stuttgart, Germany, 2014.

[10] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, USA, 1996.

[11] K. Peng. A hybrid e-voting scheme. In F. Bao, H. Li, and G. Wang, editors, *ISPEC'09, 5th Information Security Practice and Experience Conference*, LNCS 5451, pages 195–206, Xi'an, China, 2009.

[12] H. Renold, O. Esseiva, and T. Hofer. Swiss Post Voting System – System Specification – Version 1.3.1. Technical report, Swiss Post Ltd., Bern, Switzerland, June 2023.

[13] B. Terelius and D. Wikström. Proofs of restricted shuffles. In D. J. Bernstein and T. Lange, editors, *AFRICACRYPT'10, 3rd International Conference on Cryptology in Africa*, LNCS 6055, pages 100–113, Stellenbosch, South Africa, 2010.

A Numerical Example

To illustrate the proposed approach with a numerical example, we consider the groups obtained for $p = 23$ (safe prime) and $q = 11$:

$$\mathbb{G}_{11} = \{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\},$$
$$\mathbb{Z}_{23}^{+} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}.$$

In Tab. 3, we show the complete multiplication and exponentiation tables for both groups. The inverse elements can be observed in the multiplication tables, where each row and each column contains exactly one entry for the identity element 1, or in the last column ($y = 10$) of the exponentiation table (for example $2^{-1} = 12$ in $\mathbb{G}_{11}$ and $2^{-1} = 11$ in $\mathbb{Z}_{23}^{+}$).

The isomorphism $\phi(x) = x^2 \bmod 23$ as defined in Sect. 2.3 leads to the following map between the elements of $\mathbb{G}_{11}$ and $\mathbb{Z}_{23}^{+}$:

$x \in \mathbb{Z}_{23}^{+}$	1	2	3	4	5	6	7	8	9	10	11
$\phi(x) \in \mathbb{G}_{11}$	1	4	9	16	2	13	3	18	12	8	6

If we select $g = 2$ (element of both groups) as a common generator, then $(sk, pk_1) = (7, 13)$ would be a valid ElGamal key pair for $\mathbb{G}_{11}$, and $(sk, pk_2) = (7, 10)$ would be the

$\mathbb{G}_{11}$	1	2	3	4	6	8	9	12	13	16	18
1	1	2	3	4	6	8	9	12	13	16	18
2	2	4	6	8	12	16	18	1	3	9	13
3	3	6	9	12	18	1	4	13	16	2	8
4	4	8	12	16	1	9	13	2	6	18	3
6	6	12	18	1	13	2	8	3	9	4	16
8	8	16	1	9	2	18	3	4	12	13	6
9	9	18	4	13	8	3	12	16	2	6	1
12	12	1	13	2	3	4	16	6	18	8	9
13	13	3	16	6	9	12	2	18	8	1	4
16	16	9	2	18	4	13	6	8	1	3	12
18	18	13	8	3	16	6	1	9	4	12	2

$z = xy \bmod 23$

$\mathbb{Z}_{23}^{+}$	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11
2	2	4	6	8	10	11	9	7	5	3	1
3	3	6	9	11	8	5	2	1	4	7	10
4	4	8	11	7	3	1	5	9	10	6	2
5	5	10	8	3	2	7	11	6	1	4	9
6	6	11	5	1	7	10	4	2	8	9	3
7	7	9	2	5	11	4	3	10	6	1	8
8	8	7	1	9	6	2	10	5	3	11	4
9	9	5	4	10	1	8	6	3	11	2	7
10	10	3	7	6	4	9	1	11	2	8	5
11	11	1	10	2	9	3	8	4	7	5	6

$z = |xy \bmod 23|$

$\mathbb{G}_{11}$	0	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	16	9	18	13	3	6	12
3	1	3	9	4	12	13	16	2	6	18	8
4	1	4	16	18	3	12	2	8	9	13	6
6	1	6	13	9	8	2	12	3	18	16	4
8	1	8	18	6	2	16	13	12	4	9	3
9	1	9	12	16	6	8	3	4	13	2	18
12	1	12	6	3	13	18	9	16	8	4	2
13	1	13	8	12	18	4	6	9	2	3	16
16	1	16	3	2	9	6	4	18	12	8	13
18	1	18	2	13	4	3	8	6	16	12	9

$z = x^y \bmod 23$

$\mathbb{Z}_{23}^{+}$	0	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	7	9	5	10	3	6	11
3	1	3	9	4	11	10	7	2	6	5	8
4	1	4	7	5	3	11	2	8	9	10	6
5	1	5	2	10	4	3	8	6	7	11	9
6	1	6	10	9	8	2	11	3	5	7	4
7	1	7	3	2	9	6	4	5	11	8	10
8	1	8	5	6	2	7	10	11	4	9	3
9	1	9	11	7	6	8	3	4	10	2	5
10	1	10	8	11	5	4	6	9	2	3	7
11	1	11	6	3	10	5	9	7	8	4	2

$z = |x^y \bmod 23|$

Tab. 3: Multiplication and exponentiation tables for $\mathbb{G}_{11}$ and $\mathbb{Z}_{23}^{+}$.

corresponding key pair for $\mathbb{Z}_{23}^{+}$ with the same private key $sk = 7$ from $\mathbb{Z}_{11}$. If we chose $m = 8$ (element of both groups) as message to encrypt with randomization $r = 4$, then

$$\begin{aligned} e_1 &= (2^4 \bmod 23, 8{\cdot}13^4 \bmod 23) = (16, 6) \in \mathbb{G}_{11} \times \mathbb{G}_{11}, \\ e_2 &= (|2^4 \bmod 23|, |8{\cdot}10^4 \bmod 23|) = (7, 6) \in \mathbb{Z}_{23}^{+} \times \mathbb{Z}_{23}^{+}, \end{aligned}$$

are the resulting ElGamal ciphertexts. In both cases, we can perform the decryption using the private key $sk = 7$ to obtain the original plaintext message:

$$m = \begin{cases} 6/16^7 \bmod 23 = 6/18 \bmod 23 = 6 \cdot 9 \bmod 23 = 8, \\ |6/7^7 \bmod 23| = |6/5 \bmod 23| = |6 \cdot 9 \bmod 23| = 8. \end{cases}$$