# Security Analysis of ElGamal Implementations

Mohamad El Laz, Benjamin Grégoire, and Tamara Rezk
*Inria Sophia-Antipolis Méditerranée, France*
{*mohamad.el-laz, benjamin.gregoire, tamara.rezk*}*@inria.fr*

Keywords: ElGamal, DDH Assumption, Quadratic Residues, Voting Systems, Message Encoding.

Abstract: The ElGamal encryption scheme is not only the most extensively used alternative to RSA, but is also almost exclusively used in voting systems as an effective homomorphic encryption scheme. Being easily adaptable to a wide range of cryptographic groups, the ElGamal encryption scheme enjoys homomorphic properties while remaining semantically secure. This is subject to the upholding of the Decisional Diffie-Hellman (*DDH*) assumption on the chosen group. We analyze 26 libraries that implement the ElGamal encryption scheme and discover that 20 of them are semantically insecure as they do not respect the Decisional Diffie-Hellman (*DDH*) assumption. From the five libraries that do satisfy the *DDH* assumption, we identify and compare four different message encoding and decoding techniques.

## 1 INTRODUCTION

Throughout the last century, especially with the beginning of public key cryptography due to Diffie-Hellman (Diffie and Hellman, 1976), many cryptographic schemes have been proposed. Their security depends on mathematically complex problems such as integer factorization and discrete logarithm. In fact, it is thought that a cryptographic scheme is secure if it resists cryptographic attacks over a long period of time. On one hand, since certain schemes may take several years before being widely studied in depth, they become vulnerable as time passes. On the other hand, a cryptographic scheme is a provable one, if it resists cryptographic attacks relying on mathematical hypothesis.

Being easily adaptable to many kinds of cryptographic groups, the ElGamal encryption scheme enjoys homomorphic properties while remaining semantically secure (Goldwasser and Micali, 1982), provided that the Decisional Diffie-Hellman (*DDH*) assumption holds on the chosen group. While the homomorphic property forbids resistance against chosen ciphertext attacks, it is very convenient for voting systems (Cortier et al., 2015).

The ElGamal encryption scheme (ElGamal, 1985) is the most extensively used homomorphic encryption scheme for voting systems (see also Paillier (Paillier, 1999)). Moreover, ElGamal is the only homomorphic encryption scheme implemented by default in many hardware security modules (Volkamer, 2009; Orr and Liam, 2016). In order to be provably secure, the ElGamal encryption needs to be implemented on top of a group verifying the Decisional Diffie-Hellman (*DDH*) assumption. Since this assumption does not hold for all groups, one may have to wrap an encoding and a decoding phase to ElGamal to be able to have a generic encryption scheme. In this paper, our main goal is to study ElGamal encryption scheme libraries to identify which implementations respect the *DDH* assumption.

We manually analyze the source code of 26 libraries that implement the ElGamal encryption scheme in the wild. We focus our analysis on understanding whether the *DDH* assumption is respected in these implementations, ensuring a secure scheme in which no information about the original message could be leaked. The *DDH* assumption is crucial for the security of ElGamal because it ensures indistinguishability under chosen-plaintext attacks (*IND-CPA*). Without a group satisfying the *DDH* assumption, encryption mechanisms may leak one bit of information about the plaintext and endanger the security of the electoral system, thus threatening the privacy in an election. For instance, when considering an approval ballot with a *yes or no* vote, leaking one bit of information signifies a full leakage of the vote. One way to comply with the *DDH* assumption is by using groups of prime order. In particular, when adopting safe primes, one can ensure the existence of a *large* prime order subgroup (Milne, 2011) and restrict messages to belong to this subgroup. Mapping plaintexts

into subgroups is called message encoding. Such encoding necessitates to be efficient and precisely invertible to allow decoding after the decryption.

In Table 1, we give an overview of our results : out of 26 analyzed libraries, 20 are wrongly implemented because they do not respect the conditions to achieve *IND-CPA* security under the *DDH* assumption. This means that encryptions using ElGamal from any of these 20 libraries leak one bit of information (see Section 4).

From the 6 libraries which respect the *DDH* assumption, we also study and compare various encoding and decoding techniques. We identify four different message encoding and decoding techniques summarized in Table 2. Finally, we discuss the different designs and conclude which implementation is more efficient for voting systems.

## 2 RELATED WORK

*The ElGamal Scheme.* The ElGamal encryption scheme was introduced in 1985 by Taher ElGamal (ElGamal, 1985). It relies on Diffie-Hellman key exchange and is known to be semantically secure under the Decisional Diffie-Hellman (*DDH*) assumption where the discrete logarithm problem is hard to solve. In 2009, Barthe et al. (Barthe et al., 2009) proved that the ElGamal encryption scheme is secure against chosen-plaintext attacks (CPA) in the standard model assuming that the *DDH* problem is hard in the underlying group family by using the proof assistant Coq. The Cramer-Shoup cryptosystem (CS) (Cramer and Shoup, 1998) was developed by Ronald Cramer and Victor Shoup in 1998. It is a generalization of ElGamal's protocol and is provably secure against adaptive chosen ciphertext attacks (CCA), under the *DDH* assumption. Even though it is a modified version of ElGamal, the Cramer-Shoup cryptosystem cannot be used however as a substitute of ElGamal in voting systems. In fact, being resistant to CCA results in losing the homomorphic property of the scheme, which is fundamental for voting systems. In 2006, Benoît Chevallier-Mames et al. (Chevallier-Mames et al., 2006) proposed an ElGamal encryption alternative, using a new encoding-free technique. Their approach holds better performance than plain ElGamal without the necessity to map the message into a subgroup. The authors introduce the notion of the class function based on the difficulty of the Decisional Class Diffie-Hellman (DCDH) assumption. An essential improvement of the Encoding-Free version scheme is to avoid message conversion, providing a wider message space. ElGamal encoding-free is *IND-CPA*. However, to date, it is not known how to identify whether a group satisfies the DCDH assumption or not.

*Semantic Security.* The mental poker (Rivest et al., 1979) is the first protocol known to be vulnerable to attacks because its encryption scheme does not respect the *DDH* assumption. The game of mental poker is an ordinary poker game and communications between players are done via messages since it is a game without physical cards. Being exposed to attacks, the mental poker game can leak one bit of information about the cards by observing whether the encryption scheme preserves the quadratic residuosity of a number (Lipton, 1981). Consequently, in 1982, Goldwasser and Micali introduced the first probabilistic public-key encryption scheme (Goldwasser and Micali, 1982) which is provably secure under standard cryptographic assumptions. It is based on the intractability of Quadratic Residuosity Assumption modulo a composite $n$. Considering that the distribution of quadratic residues and quadratic non-residues is not the same, one restricts oneself to a subset where the number of quadratic residues is equal to the number of quadratic non-residues, and takes only the messages that are quadratic residues to prevent an attacker from gaining any information about the original message.

*Encoding Mechanisms.* What we call message encoding refers to the mechanism that maps a message into a specific group. An approach to encode a message is the hash-ElGamal encoding. This scheme consists of including a hash function during the encryption process. Let $h : \mathbb{G} \to \{0,1\}^l, w \to h(w)$ be a hash function mapping elements to $l$-bit strings. The encryption of a message $m \in \mathbb{M}$ where the message space is defined as $\mathbb{M} = \{0,1\}^l$ is then given by $(g^r, m \oplus h(y^r))$. This encoding mechanism solves the problem of leaking information about the original message but unfortunately, it cannot be used for voting systems as it loses its homomorphic property. Another option to encode messages is exponent-ElGamal encoding (Cramer et al., 1997). This technique takes advantage of a property where any element $w \in \mathbb{G} =< g >$ is uniquely represented as $w = g^z$ for some $z \in \mathbb{Z}/q\mathbb{Z}$. For any message $m \in \mathbb{Z}/q\mathbb{Z}$, the resulting encoding is $g^m$. The corresponding ciphertext is given by $(c_1, c_2) = (g^r, g^m y^r)$. In this case, the problem is in the decryption process: to retrieve the original message, the computation of the discrete logarithm in $\mathbb{G}$ is needed. Considering that the discrete logarithms are hard to solve in $\mathbb{G}$, this leads to limit the message space to a small set where the discrete logarithm problem is easy to solve by using brute force or Pollard's rho algorithm (Pollard, 1978).

Elliptic curve ElGamal is a different variant where a message $m$ is represented as a point on an elliptic curve, more accurately, as a point on a prime order subgroup. Elliptic curve cryptography (Koblitz et al., 2000; Miller, 1985) offers smaller key sizes resulting in gains in speed and memory, and benefits of the absence of sub-exponential algorithms that solve the elliptic curve discrete logarithm problem. However, encodings (Farashahi, 2014; Fadavi et al., 2018; Bernstein et al., 2013) mostly do not handle prime-order elliptic curves as there is no known polynomial time algorithm for finding a large number of points on an arbitrary curve. Furthermore, several encodings are hash-to-curve that are not invertible and therefore not compatible with group operation which destroys the homomorphic property (Faz-Hernandez et al., 2019).

*Application to E-voting.* We briefly discuss electronic voting systems that use ElGamal encryption. To get familiar with voting systems, we refer the interested reader to (Ne Oo and Aung, 2014; Cortier et al., 2016; Volkamer, 2009). E-voting systems are important as they expand the participation of voters and offer an efficient way to count votes. However, without employing secure systems, the use of voting systems would be meaningless. E-voting uses public-key cryptography: ElGamal is the most common used encryption algorithm as it enjoys multiplicative homomorphic properties that allows the addition on the ciphertexts in order to count ballots without revealing the identity of the voters. Additionally, ElGamal enables re-encryption which results in a different ciphertext containing the same information. (Paillier encryption scheme (Paillier, 1999) is a possible option as well, relying on the DCR assumption). Various studies demonstrated the importance of ElGamal as an encryption scheme for electronic voting systems (Haines et al., 2019; Adida, 2008). Additionally, several countries (e.g. Estonia (Kubjas et al., 2017), Norway (Puigalli and Guasch, 2012) and Russia (Babenko et al., 2017)) are using e-voting systems as a main mechanism for elections and are employing ElGamal to count and verify votes. Moreover, in a note of 15 November 2019, Pierrick Gaudry (Gaudry, 2019) reveals an attack about the Moscow voting system because it does not comply with the *DDH* assumption.

# 3 BACKGROUND

## 3.1 Notions and Preliminaries

The ElGamal encryption scheme is known to be *IND-CPA* secure under the decisional Diffie-Hellman assumption. In this section we recall some basic definitions and notations that will be used throughout this paper.

**Definition 3.1 (Decisional Diffie-Hellman)** *Given two distributions* $\mathrm{DDH}_0 = (g^x, g^y, g^{xy})$ *and* $\mathrm{DDH}_1 = (g^x, g^y, g^z)$ *where* $x, y, z$ *are randomly distributed in* $\mathbb{Z}_q$*, a cyclic group with generator* $g$*. It is hard to distinguish between* $\mathrm{DDH}_0$ *and* $\mathrm{DDH}_1$.

An encryption scheme is said to be IND-CPA secure if a polynomial time adversary choosing two plaintexts cannot distinguish between the resulting ciphertexts. If the adversary can distinguish between the ciphertexts better than guessing blindly, we say that the adversary achieves an advantage. The advantage of any efficient adversary is expressed as a negligible function of the security parameter in the formal definition of IND-CPA . For ElGamal, the security parameter is the key length.

To satisfy IND-CPA, an encryption scheme must necessarily be probabilistic, otherwise an adversary could trivially detect which message corresponds to the challenge ciphertext by simply encrypting one of the messages it has chosen and compare the resulting ciphertext with the challenge ciphertext.

## 3.2 The ElGamal Encryption Scheme

ElGamal (ElGamal, 1985) is an asymmetric encryption scheme, it enjoys homomorphic properties that are fundamental for the electronic voting systems. ElGamal scheme (see Algorithm 1) consists of three algorithms: *key generation*($\eta$), $\eta$ being a security parameter, *encryption* $E(m, pk)$ with $m$ being a plaintext and $pk$ a public key, and *decryption* $D(c, sk)$ where $c$ is a ciphertext and $sk$ is a private key.

**Algorithm 1** ElGamal Scheme

$\mathcal{KG}$: Pick randomly a secret key $x \in \mathbb{Z}_q$, then compute $y = g^x$ to obtain the public key.

$\mathcal{E}(m)$: Let $m \in G$ and $r \in \mathbb{Z}_q$ randomly selected. The resulting ciphertext is $c = (u, v) = (g^r, m.y^r)$.

$\mathcal{D}(c)$: To recover the plaintext, one computes $m = v \cdot u^{-x}$.

## 3.3 Security of ElGamal

A key point for the security of the ElGamal encryption scheme resides in the group $\mathbb{G}$ and its order. One should start by generating a pair of keys (public and private), then map the message into a group where the Decisional Diffie-Hellman assumption holds. Hence,

the difficulty consists in finding an efficient invertible group encoding procedure so that one can recover the original message when decrypting. The ElGamal cryptosystem operates in a finite cyclic group, which by convention is written multiplicatively. For the sake of simplicity, we will restrict our discussion to the group of integers from $\{1\}$ to $\{p-1\}$ under multiplication mod $p$ for some prime $p$, commonly denoted $\mathbb{Z}_p^*$ and subgroups of $\mathbb{Z}_p^*$ of prime order. Moreover, we will also use $|g|$ to denote the order of an element $g$ in $\mathbb{Z}_p^*$ and $< g >$ to denote the cyclic subgroup of $\mathbb{Z}_p^*$ generated by $g$. Unless otherwise noted, assume multiplications and exponentiations involving elements of $\mathbb{Z}_p^*$ are done mod $p$. As if all subgroups of a cyclic group are cyclic and if $G = \langle g \rangle$ is cyclic, then for every divisor $d$ of $|G|$ there exists exactly one subgroup of order $d$ which may be generated (Rotman, 1999). One may rely on this property to form a unique subgroup of quadratic residues elements. To achieve this goal, the idea is to use a Sophie Germain prime (Pollard, 1978): it is a safe prime $p$ of the form $2q+1$ where $q$ is also prime. Safe primes of that form are important for modulo groups as they guarantee the existence of a subgroup of prime order. For ElGamal, using a safe prime $p$, where the order is $p-1=2q$ permits to form a subgroup of prime order $q$ that forms the message space we need in order to encrypt messages. One may take advantage of the Lagrange theorem (Pollard, 1978) that states that in a finite group $G$, the order of any subgroup $H$ divides the order of the group, to conclude that the prime order subgroup has no subgroups being prime. Finally, the message space must be restrained to this prime order subgroup.

***Quadratic Residues.*** To make the ElGamal cryptosystem IND-CPA secure, the Decisional Diffie-Hellman assumption must be respected. As a matter of fact, one needs to find which type of groups satisfies the underlying assumption. A good technique is to restrict the messages to form the subgroup of prime order $q$ of quadratic residues. In what follows we will introduce further explanations and examples to better understand the role of quadratic residues for the security of the ElGamal encryption scheme.

**Definition 3.2 (Quadratic Residue)** *An element $a \in \mathbb{Z}_n^*$ is said to be a **Quadratic Residue** modulo n if there exists $x \in \mathbb{Z}_n^*$, such that $x^2 \equiv a (mod\ n)$. Every such x is called a square root of a modulo n. If no such x exists, then a is called a **Quadratic Non-Residue** modulo n. We denote the set of all quadratic residues modulo n by $QR_n$ and the set of all quadratic non-residues by $QNR_n$*

Quadratic residues are exactly those elements which can be written of the form $g^i$ where $i$ is even: $\{g^2, ..., g^{p-1}\}$ are distinct quadratic residues, while $\{g, gg^2, ..., gg^{p-1}\}$ are quadratic non-residues. There exists an efficient algorithm based on Euler's criterion for deciding quadratic residuosity in $\mathbb{Z}_p^*$, with $p$ prime:

**Definition 3.3 (Euler's Criterion)** *Let $p$ be an odd prime. Then $a \in \mathbb{Z}_p^*$ is a quadratic residue modulo $p$ iff $a^{\frac{p-1}{2}} \equiv 1$ (mod p).*

Therefore, by restricting all the messages to be Quadratic residues in a safe prime group, a polynomial time adversary cannot distinguish between elements as all the elements are then quadratic residues. Hereinafter, we will be using the Legendre symbol based on Euler's criterion for its convenient notation that reports the quadratic residuosity of $a$ mod $p$.

**Definition 3.4 ( Legendre symbol)** *Let $p$ be an odd prime , $a$ an integer, such that $gcd(a,p) = 1$. The **Legendre symbol** (LS) is defined to be*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & if \quad a \in QR_p \\ -1 & if \quad a \in QNR_p \end{cases}$$

# 4 BREAKING ELGAMAL

Let $p$ be a prime and $g \in \mathbb{Z}_p^*$. Given a public key $g^x$, the ElGamal encryption scheme encrypts a message $m \in \mathbb{Z}_p$ by computing $(g^r, m.g^{xr})$, with $r$ chosen randomly in $\mathbb{Z}_p^*$. Using the private key $x$, decryption can be done by first computing $(g^{xr})$ and then dividing to retrieve $m$. The cryptosystem is not semantically secure when $g$ is a generator of $\mathbb{Z}_p^*$, as some information about the plaintext is leaked. Specifically, the Legendre symbol of $(m.g^{xr})$ uncovers the Legendre symbolof the message $m$. In order to prove that ElGamal is *IND-CPA* under the *DDH* assumption, one may choose $g$ to be the generator of the group in which the *DDH* assumption holds and restrict the message space to this group. This way, the system is semantically secure as given $(g^x)$ and $(g^r)$, the secret value $(g^{xr})$ cannot be distinguished from a random element in the group. Therefore, $(m.g^{xr})$ cannot be distinguished from a random element and an attacker cannot learn any information about the original message. In what follows, we will show an example on how to break the *DDH* assumption and ElGamal encryption scheme for $\mathbb{Z}_p^*$ groups.

**Example 4.1 (Breaking the *DDH* Assumption)** *In order to break the* DDH *assumption, one should be able to distinguish between two distributions having elements randomly distributed in a group.*

*Let $p = 2q+1 = 11$ be the group on which we want to perform the attack (to ease the comprehension of the example, we consider small parameters size and not secure). Being prime, $p$ has $\{p-1\}$ elements, half being quadratic residues (QR) and the other half being quadratic non-residues (QNR). Given a tuple $(g^x, g^y, g^c)$ it is hard to decide whether $c = xy$ or $c = z$, where $z$ is randomly generated. By LS, we can check the quadratic residuosity of the elements $g^x$, $g^y$ and $g^c$: $\binom{x}{p} = 1$ if $x$ is a square (i.e., quadratic residue), then $\binom{g^x}{p} = 1$ if $x$ is even. Therefore, $g^x$ is quadratic residue if $x$ is a quadratic residue too. In addition, if $x$ or $y$ are even, then $xy$ is even and $g^{xy}$ is quadratic residue. Taking advantage of all the before-mentioned notions, we will give a detailed example on how to break the* DDH *assumption. A tuple is a valid* DDH *tuple if $xy \equiv c \bmod p$ and can be written as $(g^x, g^y, g^{xy})$.*
*Let $p = 11$, the challenge is to distinguish whether $(4,5,9)$ is $\mathrm{DDH}_0$ or $\mathrm{DDH}_1$ in $G_{11}$. For Euler's criterion, $a \in \mathbb{Z}_p^*$ is a QR iff $a^{\frac{p-1}{2}} \equiv 1 \pmod p$.*

$$4^5 \equiv 1\, mod 11 \quad 5^5 \equiv 1\, mod 11 \quad 9^5 \equiv 1\, mod 11$$

*By testing the quadratic residuosity of the three elements, one can notice that all of them are quadratic residues. In this case, we cannot distinguish between $xy$ and $z$. Since the multiplication between two elements that are quadratic residues, results in an quadratic residue element, it might be that the third element of the tuple belongs to* $\mathrm{DDH}_0$ *or* $\mathrm{DDH}_1$*. Consequently, we are not able to break the* DDH *assumption.*
*In this second example, the challenge is to distinguish whether $(4,5,8)$ is $(g^x, g^y, g^{xy})$ or $(g^x, g^y, g^z)$ in $G_{11}$.*

$$4^5 \equiv 1\, mod 11 \quad 5^5 \equiv 1\, mod 11 \quad 8^5 \not\equiv 1\, mod 11$$

*One can notice that the third element of the tuple is not a quadratic residue. Being both quadratic residues, the multiplication between $g^x = 4$ and $g^y = 5$ must result in an quadratic residue element. However, $g^c = 8$ results being a non quadratic residue element. In this case, we are able to distinguish between $xy$ and $z$ by ensuring that $(4,5,8)$ is $\mathrm{DDH}_1$ and break the* DDH *assumption as a consequence. We emphasize on the importance of choosing a safe prime group $p$, to be able to work in the subgroup of prime order $q$ by restricting the elements to form the subgroup of quadratic residues. This guarantees the difficulty of such attacks on the* DDH *assumption.*

**Example 4.2** ***(Breaking the ElGamal Scheme using a QR generator)*** *Being in a group in which the* DDH *does not hold may leak information about the original plaintext. An attacker able to calculate the quadratic residuosity of an encrypted message, can learn one bit of information about the original message. An attacker can check whether the encryption of a message is a QR or not and therefore deduce whether the original message is a QR too. In fact, by taking $g^{xr} \in QR$ and encrypting the message as $\mathcal{E}(m,k) = (g^r, m.g^{xr})$, one can notice that if $m.g^{xr} \in QR$ then $m \in QR$.*
*Consider the group $G_{11}$, let $x = 4 \in sk$, $r = 2 \in rnd$, $g = 3 \in QR$ and $y = g^x = 4 \in pk$.*
*In the above example, the group $G_{11}$ has $\{1,3,4,5,9\}$ as a subgroup of quadratic residues. Actually, encrypting a message $m \in QR$ with a public key $pk \in QR$, results always in an encryption $\mathcal{E}(m,k) \in QR$. Thus, by using $G_{11}$ and taking messages belonging to its entire message space, one endangers the security of the scheme as he allows QR and QNR messages to be encrypted. An attacker able to calculate the quadratic residuosity of an element could learn one bit of information about the original message by performing the following attack:*

*(a) $g \in QR$, then $(g^{xr})$ is QR.*
*(b) Check if $\binom{\mathcal{E}(m,k)}{p}$ is QR or not.*
*(c) If $\binom{\mathcal{E}(m,k)}{p} \in QR$, the attacker can learn that the message $m \in QR$, as the multiplication between QR elements result always in a QR element.*
*(d) If $\binom{\mathcal{E}(m,k)}{p} \in QNR$, then the attacker can learn that the message $m \in QNR$.*

*Let us check $\mathcal{E}(\underline{1},5) = (9,\underline{5})$:*

*(a) $\binom{3}{11} = 1 \Rightarrow g \in QR$, $\binom{3^8}{11} = 1 \Rightarrow (g^{xr}) \in QR$.*
*(b) $\binom{5}{11} = 1 \Rightarrow \mathcal{E}(1,5) \in QR$.*
*(c) Being $m.g^{xr} \in QR$ leaks the quadratic residuosity of $m$. In fact $\binom{1}{11} = 1 \Rightarrow m \in QR$.*

*The previous example could also be adopted for QNR messages. In summary, the feasibility of calculating the quadratic residuosity of an element in a modulo prime groups, may leak information about the original message on top of groups that do not respect* DDH*.*

# 5 STUDY OF LIBRARIES

We focus our study on manually checking whether the underlying groups in ElGamal implementations satisfy the Decisional Diffie-Hellman assumption. A summary of our results regarding 26 analyzed libraries can be found in Table 1.
We split our work into three tasks to measure the correctness of the implementations. We first verify that the implementations use safe primes, next

we check if they adopt quadratic residue generators to generate subgroups in which the *DDH* assumption holds. Finally, we analyze the message encoding techniques deployed to map the messages into the before-mentioned subgroups. This study led us to notice that a large number of the considered libraries are not *IND-CPA* secure as the encryption may leak at least one bit of information on the plaintext: 20 libraries do not respect the *DDH* assumption. Moreover, among these 20 libraries, 10 do not use a safe prime. In what follows, we describe in details the problems found in the investigated libraries using the following classification:

**(A)** *Libraries that do not respect the* DDH *assumption.* There are 20 libraries in this category. In this class we further classify the libraries that do not respect the *DDH* assumption due to 3 different concerns: libraries that do not deploy a safe prime, libraries that do not adopt a quadratic residue generator, and libraries that do not use a correct message encoding technique to map the messages into the intended subgroup.

**(B)** *Libraries that do respect the* DDH *assumption.* There are 6 libraries in this category. However, they do not all use the same encoding technique. Thus, we describe in detail the 4 different encoding techniques, discussing their advantages and drawbacks.

***Libraries' Relevance.*** This paper will provide a brief description on the relevance of some of the chosen libraries for analysis. Belenios (Belenios, 2016) has been deployed on an online platform and it is used in more than 200 elections, both in academia and in education. Similarly, Helios (Helios, 2008) is used for the election of the International Association for Cryptographic Research members board, the ACM general elections, the election of UCLouvain president and for other student elections. On the other hand, the Estonian voting system (Estonia, 2017) has been used for the European Parliament elections, local government council elections and the election of the president of the Republic. Swisspost (Swisspost, 2018) then offers voting system services for cantons and municipalities in Switzerland, while Verificatum (Verificatum, 2017) is used in binding elections, student body elections and intra-party elections.

## 5.1 Libraries That Do Not Respect *DDH*

In this section, we will present the implementations that do not respect the conditions to achieve an *IND-CPA* secure ElGamal scheme. Working on groups modulo $p$, a secure ElGamal scheme has to adopt safe primes, a quadratic residue generator, and a message encoding technique to map the messages into a subgroup that respects the Decisional Diffie-Hellman assumption. Twenty out of twenty six libraries violate one or more of the before-mentioned conditions: all the libraries in this section do not employ message encoding techniques.

***Lack of Safe Prime.*** In this category, 10 libraries (Diaz, 2017; Alves, 2015; Sidorov, 2016; Lee, 2017; Wang, 2017; Pankratiew, 2018; Pellegrini, 2017; Musat, 2017; Libgcrypt, 2013; Moscow, 2019a) do not use safe primes. Instead, these implementations focus on generating large numbers and checking their primality. This method does not guarantee the generation of a safe prime. In fact, a safe prime of the form $p = 2q+1$ where $q$ is also a large prime, is essential as it forms a large prime subgroup of order $q$. Conversely, using an arbitrary large prime results in a $p-1$ group order, that can be decomposed into small prime order subgroups. Hence, small prime order subgroups are exposed to subgroup attacks in which the discrete logarithm is easy to compute by using the Pohlig-Hellman algorithm (Pohlig and Hellman, 1978) or the Pollard's rho algorithm (Pollard, 1978).

***Lack of QR Generators.*** In this category, we discuss the libraries that do not use quadratic residue generators. Among the 20 libraries that do not respect *DDH*, 5 libraries (Nasr, 2015; Ridhuan, 2016; Elgamir, 2016; Pycrypto, 2012; Ioannou, 2014) use a safe prime $p = 2q+1$ but do not choose a quadratic residue generator. Using a safe prime without a quadratic residue generator does not guarantee a subgroup of prime order $q$ in which the *DDH* assumption stands. In what follows we show an example on the feasibility of learning information about the plaintext when we do not employ a quadratic residue generator.

**Example 5.1** *(**Breaking ElGamal without a QR generator**) To break ElGamal without a QR generator, it is sufficient to break the underlying assumption. The* DDH *assumption does not hold in $\mathbb{Z}_p^*$ if $g$ is a generator of $\mathbb{Z}_p^*$. This is because the Legendre symbol of $g^a$ reveals whether $a$ is even or odd. Given $(g^a, g^b, g^{ab})$, one can compute the LS and compare the least significant bit of a, b and ab, which allows to distinguish between $g^{ab}$ and a random element group. Having a distinguisher against* DDH *means having a distinguisher against ElGamal and therefore break ElGamal.*

***Lack of Encoding.*** We point out the relevance of message encoding mechanisms as a crucial requirement in ElGamal scheme. Despite generating a safe prime and choosing a quadratic residue generator, 5 libraries (Botan, 2018; Riddle, 2014; Norvegia, 2017;

Table 1: An overview on the 26 analyzed libraries (where T1, T2, T3 and T4 are listed in Table 2) .

| Library | Safe prime | *QR* Generator | Encoding | Decoding | *DDH* |
|---|---|---|---|---|---|
| Belenios (Belenios, 2016) | ✓ | ✓ | T3 | T3 | **Yes** |
| Botan (Botan, 2018) | ✓ | ✓ | ✗ | ✗ | **No** |
| Cryptology (Nasr, 2015) | ✓ | ✗ | ✗ | ✗ | **No** |
| Elgamal-api (Diaz, 2017) | ✗ | ✗ | ✗ | ✗ | **No** |
| ElGamal-cipher.py (Alves, 2015) | ✗ | ✗ | ✗ | ✗ | **No** |
| ElGamalExt (Sidorov, 2016) | ✗ | ✗ | ✗ | ✗ | **No** |
| ElGamal.h (Lee, 2017) | ✗ | ✗ | ✗ | ✗ | **No** |
| ElGamal.hs (Ridhuan, 2016) | ✓ | ✗ | T3 | T3 | **No** |
| Elgamal.hpp (Wang, 2017) | ✗ | ✗ | ✗ | ✗ | **No** |
| Elgamal.java (Pankratiew, 2018) | ✗ | ✗ | ✗ | ✗ | **No** |
| Elgamal-lib.c (Pellegrini, 2017) | ✗ | ✗ | ✗ | ✗ | **No** |
| Elgamal.py (Musat, 2017) | ✗ | ✗ | ✗ | ✗ | **No** |
| Elgamal.py (Riddle, 2014) | ✓ | ✓ | ✗ | ✗ | **No** |
| Elgamir (Elgamir, 2016) | ✓ | ✗ | ✗ | ✗ | **No** |
| Estonia (Estonia, 2017) | ✓ | ✓ | T1 | T1 | **Yes** |
| Helios (Helios, 2008) | ✓ | ✓ | T2 | T2 | **Yes** |
| Libgcrypt (Libgcrypt, 2013) | ✗ | ✗ | ✗ | ✗ | **No** |
| Microsoft (Microsoft, 2019) | ✓ | ✓ | T3 | T3 | **Yes** |
| Moscow 07-19 (Moscow, 2019a) | ✗ | ✗ | ✗ | ✗ | **No** |
| Moscow 09-19 (Moscow, 2019b) | ✓ | ✓ | T4 | T4 | **Yes** |
| Norway (Norvegia, 2017) | ✓ | ✓ | ✗ | ✗ | **No** |
| PyCrypto (Pycrypto, 2012) | ✓ | ✗ | ✗ | ✗ | **No** |
| PyCryptodome (Pycryptodome, 2018) | ✓ | ✓ | ✗ | ✗ | **No** |
| RSA-ElGamal (Ioannou, 2014) | ✓ | ✗ | ✗ | ✗ | **No** |
| Swisspost (Swisspost, 2018) | ✓ | ✓ | ✗ | ✗ | **No** |
| Verificatum (Verificatum, 2017) | ✓ | ✓ | T1 | T1 | **Yes** |

Pycryptodome, 2018; Swisspost, 2018) do not use a message encoding to map the messages into a valid subgroup. However, by adopting the standard encryption scheme of ElGamal, the message space is not restricted to the expected subgroup. This imply that even in the presence of a safe prime and consequently the presence of a subgroup of prime order generated by a quadratic residue generator, the message to encrypt is not mapped into the designed subgroup. Hence, an attacker can gain knowledge about the original message and expose the entire scheme to total insecurity. To better understand the importance of using a message encoding method, we display an attack on how to break a scheme that does not map messages into the intended subgroup (see example in Section 4).

## 5.2 Libraries That Do Respect *DDH*

In this section, we will present the implementations that respect the *DDH* assumption and therefore implement an *IND-CPA* secure ElGamal scheme. As mentioned in the previous section, a well implemented library should adopt a safe prime, a quadratic residue generator, and a message encoding technique. Only 6 out of the 26 analyzed libraries (Belenios, 2016; Estonia, 2017; Helios, 2008; Microsoft, 2019; Moscow, 2019b; Verificatum, 2017) respect all of the previously mentioned conditions. In the following paragraphs, we will discuss in detail the message encoding techniques implemented in these libraries. In particular, we can categorize four different techniques.

### 5.2.1 Limited Message Space and $q$-exponentiations

**The Estonian and Verificatum** In this paragraph, we will present two libraries that implement ElGamal using the same technique: the Estonian voting system and Verificatum. The Estonian government relies on Internet voting in a significant way for national elections. While the Estonian voting system (Estonia, 2017) is implemented in Java, Verificatum (Verificatum, 2017), which implements provably secure cryptography libraries for electronic voting systems, is implemented in JavaScript. To comply with

the *IND-CPA* security of ElGamal, these two implementations adopt a safe prime, and generate the subgroup of prime order in which the *DDH* assumption holds. Both implementations allow messages $m$ to be any integer from $[1, p-1]$. Hence, before encrypting, one checks if $m$ is a *QR* by checking $m^q \equiv 1 \bmod p$:

**Proof 1** *Euler's criterion states that $a \in \mathbb{Z}_p^*$ is a quadratic residue modulo p iff $a^{\frac{p-1}{2}} \equiv 1$ (mod p). Being $q = \frac{p-1}{2}$, then $a^{\frac{p-1}{2}} = a^q \equiv 1$ (mod p).*

If the equivalence is confirmed then $m$ is encrypted as a *QR*, else the message is rejected and an error is raised (see Listing 1). In fact, these two implementations take as an input messages in $\mathbb{Z}_p^*$ and rejects half of the messages that are not *QR* instead of encoding them (also see Helios in Subsection 5.2.2). Besides that, using $q$-exponentiations to encrypt messages can be optimized as we will explain in Section 6.

```
switch (legendre(m)) {
case 1:
   return new ModPGroupElement(this, m);
case -1:
  throw new IllegalArgumentException("Can not
encode as quadratic residue");
   }
// Negate if not a quadratic residue.
var value = new LargeInteger(bytesToUse);
return new ModPGroupElement(this, value);
```

Listing 1: Limited space and $q$-exponentiations ElGamal encoding (Verificatum, 2017).

### 5.2.2 Encoding with $q$-exponentiations

**Helios** Helios (Helios, 2008), is a known library implemented in Python. It is used for voting systems and can be manipulated to meet the needs of the users. Helios, is vulnerable to ballot stuffing as a dishonest bulletin board could add ballots without anyone noticing (Belenios, 2016). Being *IND-CPA* secure, ElGamal in Helios is implemented by generating a safe prime $p = 2q+1$ where $p$ and $q$ are both large primes. It then selects a generator $g$ of the subgroup of prime order $q$. Before encrypting a message $m$, a mapping to the prime order subgroup is done. As the implementation allows the message $m$ to be any integer from $[0, p-1]$, one computes $m_0 = m+1$ (to avoid picking $m = 0$) and checks whether $m_0$ is a *QR*. If ${m_0}^q \equiv 1 \bmod p$ then it outputs $m_0$ which belongs to the *QR* elements (as in Subsection 5.2.1), else, it outputs $-m_0$. Please note that $-m_0 \bmod p$ belongs also to the *QR* elements. Being a safe prime $p$ and $p \equiv 3 \bmod 4$, ensures the fact that 1 is a square element and $-1$ is a non-square element. This is essential in turning a non-square element $m$ into a square element $-m$. After decryption, one obtains $m$ and checks if $m \leq q$. If $m \leq q$ then $m_0 = m$ otherwise $m_0 = -m$. To recover the message, one computes $m = m_0 - 1$ (see Listing 2).

**Proof 2** *This is because $x^2 \equiv (-x)^2$ mod p. So the squares of the first half of the nonzero numbers mod p give a complete list of the nonzero quadratic residues mod p. If p is an odd prime, the residue classes of $\{1^2, 2^2, ..., (\frac{p-1}{2})^2\}$ are distinct and give a complete list of the quadratic residues modulo p. So there are $\frac{p-1}{2}$ residues and $\frac{p-1}{2}$ non-residues. This gives a complete list as $x^2$ and $(p-x)^2$ are equivalent mod p:*

$$\begin{aligned} x^2 \equiv y^2 \ mod\ p &\Leftrightarrow p | x^2 - y^2 \\ &\Leftrightarrow p | (x-y)(x+y) \\ &\Leftrightarrow p | (x-y) \ or \ p | (x+y) \end{aligned}$$

*which is impossible if x and y are two different members of the set.*

As we will see in the Section 6, it is possible to reduce to 2 the $q$-exponentiations, and therefore obtain a more efficient encoding process.

```
if (encode_m) {
      var y = m.add(BigInt.ONE);
      var test = y.modPow(pk.q, pk.p);
      if (test.equals(BigInt.ONE)) {
        this.m = y;
      } else
        this.m = y.negate().mod(pk.p); }
```

Listing 2: ElGamal encoding with $q$-exponentiations (Helios, 2008).

### 5.2.3 Hard Decoding

**Belenios and Microsoft** Belenios (Belenios, 2016) is a verifiable voting system built upon Helios. It can be used to organize elections and perform verification. Contrary to Helios, Belenios provides eligibility verifiability as anyone can check that ballots are coming from legitimate voters: each voter receives a private credential, while the election server receives only the corresponding public credential. Therefore, even if the election server is compromised, no ballot can be added. Microsoft Election guard (Microsoft, 2019) is a library that verifies voting ballots. Concerning ElGamal encryption scheme a message $m$ is encoded as $g^m$ where $g$ is the *QR* generator of the prime order subgroup: every element written in the form of $g^m$ belongs to the subgroup generated by $g$. The choice of using the exponential version of ElGamal is to benefit from turning the multiplicative homomorphism of ElGamal into an additive one. After decryption, one should compute the discrete logarithm of $g^m$ in order

to recover the initial message $m$. This is possible by using Pollard-lambda algorithm or brute force only if $m$ is taken from a small subset and not from the entire interval $\{0,...,q-1\}$. Being in a subgroup of prime order $q$, where $q$ is a large prime, it is not possible to decompose the subgroup in smaller subgroups (Lagrange Theorem (Pollard, 1978)) since the computation of the discrete logarithm is unfeasible in general (see Listing 3).

```
type factor = elt partial_decryption
let eg_factor x {alpha; _} =
  let zkp = "decrypt|" ^ G. to_string (g **~ x) ^
"|" in
        alpha **~ x,
    fs_prove [| g; alpha |] x (hash zkp)
let check_ciphertext c =
    Shape. forall (fun {alpha; beta} -> G.check
alpha && G.check beta) c
```

Listing 3: Hard decoding implementation (Belenios, 2016).

#### 5.2.4 Encoding with 2-exponentiations

**Moscow Voting System** For the elections of September 2019, the Russian government decided to employ an electronic voting system (Moscow, 2019a) to elect governors for local parliaments in Moscow. In July 2019, the source code, developed by the Moscow Department of Information Technology, was made public to test its vulnerabilities. At that time, the Moscow voting system was discovered to be subjected to two attacks by researchers (Gaudry, 2019). In the first test, the researchers exploited the fact that the keys used are small: three keys of 256 bits were used. Discrete logarithms defined by small primes are easy to calculate in feasible time. Therefore, one can compute the discrete logarithm and recover the secret keys used for decryption. Moreover, one can decrypt messages employing the same time as a legitimate possessor of secret keys. After reporting this issue, the developers of Moscow voting system increased the key size to 1024 bits. However, a second test was made to verify the security of the modified version. In this version, the message space was not restricted to the subgroup of quadratic residues as any message was allowed to be encrypted. By relying on subgroup attacks, one can get enough information about the voter's choice and indeed can reveal one bit of information about the original message (see Section 4). Therefore the Decisional Diffie-Hellman assumption did not hold and the system leaked strong information. Two days before the elections, the developers modify the source code (Moscow, 2019b) and adopt an efficient method to secure their voting system. To map a message $m$ into the *QR* subgroup, it is sufficient to square the message: $m \rightarrow m^2$ (see Listing 4).

**Proof 3** *The quadratic residue theorem states that $a \in QR$ if $\exists\, x$ s.t. $x^2 \equiv a(p)$. Let $m = x$, then $m^2 \in QR$ if $m^2 \equiv m^2(p)$, which can be trivially satisfied.*

After decryption, one obtains $m$ by calculating its modular square root: $m = c^{\frac{q+1}{2}}$ (Nishihara et al., 2009). The algorithm computes the square root of $c$ iff $p \equiv 3 \bmod 4$, which is always the case when using a safe prime. As a matter of fact, the underlying group in this last version respects the *DDH* assumption. We will discuss the adopted method in Section 6.

```
const sessionKey = trimBigInt(xoredRandomBigInt,
this.moduleP.bitLength() - 1);
const squaredData = dataAsBI.modPow(new BigInt('2
'), this.moduleP);
const a = this.generatorG.modPow(sessionKey, this
.moduleP).toString();
const b = this.publicKey
              .modPow(sessionKey, this.moduleP)
              .multiply(squaredData)
               .remainder( this.moduleP)
              .toString();
```

Listing 4: ElGamal encoding with 2-exponentiations (Moscow, 2019b).

## 6 COMPARISON OF ENCODINGS

In the previous section, we have seen 4 different encoding techniques of ElGamal that comply with the *DDH* assumption. In Table 2, we summarize the four techniques by giving a general overview on the encoding and the decoding processes.

**T1** The first message encoding technique (T1) checks whether a message $m$ is a *QR* or not by checking the following equivalence: $m^q \equiv 1\ (p)$. If the equivalence holds, then the message is encrypted, otherwise an error is raised and the message is rejected. The decoding operation is simple as one outputs directly the message $m$.

**T2** The second message encoding technique (T2) uses the same method as the previous one, but instead of raising an error and rejecting the messages, it maps the message as $-m$. For what concerns the decoding process, one first checks if $m < q$ and output $m$, otherwise it outputs $-m$.

**T3** The third message encoding technique (T3) maps a message $m$ as $g^m$. The decoding mechanism is hard in general and can be only applied on a small subset of messages in which the computation of discrete logarithm can be solved by brute force.

Table 2: Message Encoding Comparison.

| | T1: Limited msg space & $q$-expon. | T2: $q$-expon. | T3: m-expon. | T4: 2-expon. |
|---|---|---|---|---|
| Encoding | $m^q \equiv 1\ (p)? \ m : \text{error}$ | $m^q \equiv 1\ (p)? \ m : -m$ | $g^m$ | $m^2$ |
| Decoding | $m$ | $m < q\ ?: \ m : -m$ | $log_g\ (g^m)$ | $m^{\frac{q+1}{2}}$ |

**T4** Concerning the fourth message encoding technique (T4), one maps a message $m$ as $m^2$ into the subgroup of order $q$. This squaring technique is sufficient to map any message as a *QR* element. In addition, it is efficient as it needs only 2 exponentiations for encoding any message. To decode, one computes the square root by modular exponentiation of $m$: $m^{\frac{q+1}{2}}$ to recover the original message.

Whereas in T4, the encoding technique is faster than T1 and T2 (2-exponentiations is more performant than $q$-exponentiations as $q$ is large), the decoding process is T1 and T2 is faster. However, for what concerns electronic voting systems, usually several encryptions are made and only one decryption is needed to reveal the result of an election. We conclude that T4 is more efficient to apply to electronic voting systems. Additionally, as reported in the note on Moscow voting systems (Gaudry, 2019), this technique is efficient since the decryption ($q$-exponentiations) is usually done on high-end servers, while the encryption (2-exponentiations) is done on the voter's device.

In addition, T1, T2, T3, and T4 implement a *QR* generator using $q$-exponenentiations since they check its quadratic residuosity by calculating $g^q(p)$. However, one can simply implement a quadratic residue generator by using only 2-exponentiations instead of using $q$-exponentiations ($q$-exponentiations are used in (Belenios, 2016; Helios, 2008; Microsoft, 2019; Moscow, 2019b; Verificatum, 2017) for the quadratic residue generator). But clearly, a more direct and efficient way to calculate the generator is by fixing it in advance. For example $g = 4$ if we are working in $\mathbb{Z}_p^*$. Note that the performance of the encoding is more important than the performance of the generator calculation, which occurs only once at the initial phase of the voting process. Moreover, the performance of the decoding is also less important than the encoding in a voting process as discussed previously.

We provide a reference implementation in Ocaml (Rémy, 2000) (see Listing 5) in which we apply the encoding and decoding process as in T4. In addition, in our implementation, the generation of the quadratic residue generator differs from all the other implementations as we use 2-exponentiations instead of $q$-exponentiations.

```
(*Generate a safe prime of the form p = 2q + 1*)
let rec random_safe_prime nbits =
  let q = sample (nbits - 1) in
  let q = Z.nextprime q in
  let p = Z.succ (Z. shift_left q 1) in
  if Z.probab_prime q 10 <> 0
  && Z.probab_prime p 10 <> 0 then p
  else random_safe_prime nbits

(*Generate a QR generator*)
let generator pbits p =
  let q = Z. shift_right p 1 in
  let g = Z.succ (sample_le pbits (Z.sub p (Z.
of_int 2))) in
  Z.powm g (Z.of_int 2) p

(*Define the group*)
let sample_group pbits =
  let p = random_safe_prime pbits in
  let g = generator pbits p in { pbits = pbits }

(*Encode the message as a QR element*)
let encode gr m = Z.powm (Z.succ m) (Z. of_int 2)
 gr.p

(*Encryption*)
let encrypt gr pk m =
  if ((Z.leq Z.zero m) && (Z.lt m (Z.pred (q gr))
)) then
    let r = sample_le (gr. pbits - 1) (q gr) in
    (Z.powm gr.g r gr.p, mulm gr (Z.powm pk r gr.
p) (encode gr m))
  else raise (Invalid_argument "ElG_encryption")

(*Decryption*)
let decrypt gr sk (u,v) =
  let mult = Z.mul (Z.pred (q gr)) sk in
  let modulo = Z.powm u mult gr.p in
  let dec = mulm gr v modulo in
  decode gr dec

(*Decoding*)
let decode gr m =
  let p = gr.p in
  let q = q gr in
  let r = Z.powm m (Z. shift_right (Z.succ q) 1) p
  in
  let m = if Z.leq r q then r else (Z.sub p r) in
  (Z.pred m)
```

Listing 5: Our Ocaml implementation using a *QR* Generator & 2-exponentiations for an efficient message encoding.

# 7 CONCLUSION

During our analysis, we have discovered a number of ElGamal scheme implementations that are not *IND-CPA* secure since they do not respect the *DDH* assumption. On one hand, some implementations do not employ safe primes, an essential condition to form subgroups of large prime order in which the *DDH* assumption holds. On the other hand, other implementations do not apply message encoding mechanisms or use Quadratic Residue Generators. As a consequence, 20 out of the 26 analyzed libraries may leak one bit of information about the original message and therefore, may endanger the validity of an election. Finally, after comparing four different message encoding techniques that satisfy the *DDH* assumption, we conclude which implementation is most convenient for voting systems. We focused the current study on manually analyzing the *IND-CPA* security of open source code libraries of ElGamal encryption scheme. However, it is also possible to check the *IND-CPA* (in-) security when source code is not available. In fact, by applying the technique discussed in the Example 4.2 of Section 4, one can black-box test applications. In particular, such tests can be applied to ElGamal encryptions obtained by Hardware Security Modules (*HSM*) (Volkamer, 2009; Orr and Liam, 2016), which are used e.g. in the Estonian I-voting system (Springall et al., 2014). We leave this as future work.

# REFERENCES


Adida, B. (2008). Helios: Web-based open-audit voting. In *USENIX '08*, pages 335–348.

Alves, P. (2015). Public source code of library n.5. https://github.com/pdroalves.

Babenko, L., Pisarev, I., and Makarevich, O. B. (2017). A model of a secure electronic voting system based on blind intermediaries using russian cryptographic algorithms. In *SIN '17*, pages 45–50.

Barthe, G., Grégoire, B., and Béguelin, S. Z. (2009). Formal certification of code-based cryptographic proofs. In *POPL '09*, pages 90–101.

Belenios (2016). Public source code of library n.1. https://github.com/glondu/belenios.

Bernstein, D. J., Hamburg, M., Krasnova, A., and Lange, T. (2013). Elligator: elliptic-curve points indistinguishable from uniform random strings. In *ACM SIGSAC '13*, pages 967–980.

Botan (2018). Public source code of library n.2. https://github.com/randombit/botan.

Chevallier-Mames, B., Paillier, P., and Pointcheval, D. (2006). Encoding-free elgamal encryption without random oracles. In *PKC '06*, pages 91–104.

Cortier, V., Fuchsbauer, G., and Galindo, D. (2015). Beleniosrf: A strongly receipt-free electronic voting scheme. *IACR*, 2015.

Cortier, V., Galindo, D., Küsters, R., Müller, J., and Truderung, T. (2016). Sok: Verifiability notions for e-voting protocols. In *IEEE, SP '16*, pages 779–798.

Cramer, R., Gennaro, R., and Schoenmakers, B. (1997). A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT '97*, pages 103–118.

Cramer, R. and Shoup, V. (1998). A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO '98*, pages 13–25.

Diaz, A. (2017). Public source code of library n.4. https://github.com/vrnvu/elgamal-ap.

Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Trans. Inf. Theory '76*, pages 644–654.

ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory '85*, pages 469–472.

Elgamir (2016). Public source code of library n.14. https://github.com/d5c5ceb0/elgamir.

Estonia (2017). Public source code of library n.15. https://github.com/vvk-ehk/ivxv.

Fadavi, M., Farashahi, R. R., and Sabbaghian, S. (2018). Injective encodings to binary ordinary elliptic curves. In *SAC '18*, pages 434–449.

Farashahi, R. R. (2014). Hashing into hessian curves. *IJACT '14*, pages 139–147.

Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R. S., and Wood, C. A. (2019). Hashing to Elliptic Curves. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-05. Technical report.

Gaudry, P. (2019). Breaking the encryption scheme of the moscow internet voting system. *CoRR '19*.

Goldwasser, S. and Micali, S. (1982). Probabilistic encryption and how to play mental poker keeping secret all partial information. In *ACM '82*, pages 365–377.

Haines, T., Goré, R., and Tiwari, M. (2019). Verified verifiers for verifying elections. In *ACM SIGSAC*, CCS '19, pages 685–702.

Helios (2008). Public source code of library n.16. https://github.com/benadida/helios-server.

Ioannou, O. (2014). Public source code of library n.24. https://github.com/oorestisime.

Koblitz, N., Menezes, A., and Vanstone, S. A. (2000). The state of elliptic curve cryptography. *Des. Codes Cryptogr. '00*, pages 173–193.

Kubjas, I., Pikma, T., and Willemson, J. (2017). Estonian voting verification mechanism revisited again. *IACR*, 2017.

Lee, R. (2017). Public source code of library n.7. https://github.com/rayli-bot/modulus-calculation.

Libgcrypt (2013). Public source code of library n.17. https://github.com/gpg/libgcrypt.

Lipton, R. J. (1981). How to Cheat at Mental Poker. In *Proceeding of AMS short course on Cryptology '81*.

Microsoft (2019). Public source code of library n.18. https://github.com/microsoft/electionguard-verifier.

Miller, V. S. (1985). Use of elliptic curves in cryptography. In *CRYPTO '85*, pages 417–426.

Milne, J. S. (2011). Fields and galois theory (v4.22).

Moscow (July, 2019a). Public source code of library n.19. https://github.com/moscow-technologies.

Moscow (September, 2019b). Public source code of library n.20. https://github.com/moscow-technologies.

Musat, A. (2017). Public source code of library n.12. https://github.com/andreeamusat/.

Nasr, H. (2015). Public source code of library n.3. https://github.com/ananasr/cryptology.

Ne Oo, H. and Aung, A. (2014). A survey of different electronic voting systems. *IJSER '14*.

Nishihara, N., Harasawa, R., Sueyoshi, Y., and Kudo, A. (2009). A remark on the computation of cube roots in finite fields. *IACR '09*, page 457.

Norvegia (2017). Public source code of library n.21. https://www.regjeringen.no/en/topics/elections-and-democracy/den-norske-valgordningen/the-norwegian-electoral-system/id456636.

Orr, D. and Liam, K. (2016). SAC '15. Lecture Notes in CS '16.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99*, pages 223–238.

Pankratiew, A. (2018). Public source code of library n.10. https://github.com/n1ghtf1re/public-key-ciphers.

Pellegrini, J. (2017). Public source code of library n.11. http://aleph0.info/jp/software/elgamal.

Pohlig, S. C. and Hellman, M. E. (1978). An improved algorithm for computing logarithms over gf(p) and its cryptographic significance (corresp.). *IEEE Trans. Inf. Theory*, 24:106–110.

Pollard, J. M. (1978). Monte carlo methods for index computation $({mod} p)$.

Puigalli, J. and Guasch, S. (2012). Cast-as-intended verification in norway. In *EVOTE '12*, pages 49–63.

Pycrypto (2012). Public source code of library n.22. https://github.com/dlitz/pycrypto.

Pycryptodome (2018). Public source code of library n.23. https://github.com/legrandin/pycryptodome.

Rémy, D. (2000). Using, understanding, and unraveling the ocaml language. from practice to theory and vice versa. In *APPSEM '00*, pages 413–536.

Riddle, R. (2014). Public source code of library n.13. https://github.com/ryanriddle/elgamal.

Ridhuan, I. (2016). Public source code of library n.8. https://github.com/ilyasridhuan/elgamal.

Rivest, R. L., Shamir, A., and Adleman, L. M. (1979). Mental poker. *The Mathematical Gardner '79*, pages 120–126.

Rotman, J. (1999). *An Introduction to the Theory of Groups*. Graduate Texts in Mathematics '99.

Sidorov, V. (2016). Public source code of library n.6. https://github.com/bazzilic/elgamalext.

Springall, D., Finkenauer, T., Durumeric, Z., Kitcat, J., Hursti, H., MacAlpine, M., and Halderman, J. A. (2014). Security analysis of the estonian internet voting system. In *ACM SIGSAC '13*, pages 703–715.

Swisspost (2018). Public source code of library n.25. https://gitlab.com/swisspost/evoting-solution.

Verificatum (2017). Public source code of library n.26. https://github.com/verificatum.

Volkamer, M. (2009). *Evaluation of Electronic Voting*. Lecture Notes in Business Inf. Proc. '09.

Wang, K. (2017). Public source code of library n.9. https://github.com/wangkepfe/cryptography.