An Attack on the Encryption Scheme of the Moscow Internet Voting System

Alexander Golovnev*

August 24, 2019

Abstract

The next Moscow City Duma elections will be held on September 8th with an option of Internet voting. Some source code of the voting system is posted online for public testing. Pierrick Gaudry recently showed that due to the relatively small length of the key, the encryption scheme could be easily broken. This issue has been fixed in the current version of the voting system. In this note we show that the new implementation of the ElGamal encryption system is not semantically secure. We also demonstrate how this newly found security vulnerability can be potentially used for counting the number of votes cast for a candidate.

1 Introduction

The Internet voting system developed for the Moscow City Duma elections had a public testing during the month of July. Some of the system’s code was posted online [git19], and the organizers asked the public to test several attack scenarios [Pub19]. The system is poorly documented, but from the source code and brief descriptions of the system [Kri19], we know that it uses the Etherium blockchain and ElGamal encryption. In one of the attack scenarios, the organizers publish a challenge consisting of the public key and some encrypted messages.

Recently Pierrick Gaudry [Gau19a] discovered that the implemented encryption system used a concatenation of three ElGamal encryptions each with a key of length 256 bits. Gaudry showed that since the keys were too short, each private key could be retrieved in minutes. Moreover, Gaudry provided a very short and beautiful script computing the secret keys from the public keys. In the same note, Gaudry remarked that the implemented version of ElGamal worked in groups of even order, which means that it leaked a bit of the message. Both of these issues have been fixed in the current version of the Internet voting system. Gaudry [Gau19b] acknowledged that these specific issues have been fixed, but he is still doubtful about the security of the system. (In particular, he says [Gau19b] that he cannot properly test the system due to the

*Harvard University

https://github.com/moscow-technologies/blockchain-voting/blob/1d4f346881e961420a28e8a3a3d64719c4ddc628/encryption-keys/keys/public-key.json#L1-L5

and

https://github.com/moscow-technologies/blockchain-voting/blob/1d4f346881e961420a28e8a3a3d64719c4ddc628/encryption-keys/data/encrypted-datums.csv#L1-L10
lack of documentation, and is concerned about potential flaws caused by the recent big changes fixing the key length issue.)

In this note we show that the new implementation of the encryption system also leaks a bit of the message. This is caused by the usage of ElGamal where the message is not mapped to the cyclic group under consideration. We show that this flaw potentially can be used for counting the number of votes cast for a candidate, which is illegal (until the end of the election period).

After the publication of this note, Pierrick Gaudry told us that he did warn the developers of the system that they should make sure that the system design does not allow attackers to encrypt elements outside of the group. As we show in this note, even the messages generated by the system (let alone an attacker) do not necessarily belong to the group.

2 Attack

The current implementation of the electronic voting system uses the ElGamal public key encryption scheme. The source code is available in [git19], and the encoding procedure is provided in Appendix A. Below we summarize the current implementation.

Let \( p = 2q + 1 \) and \( q \) be primes of length about 1024 bits, and let \( Q_p \) be the group of quadratic residues modulo \( p \). Note that \( |Q_p| = (p−1)/2 = q \). The system picks a \( g \in Q_p \) generating \( Q_p \). Let \( pk \in Q_p \) and \( sk \in \{1, \ldots, q\} \) be the public and secret key respectively, and \( pk = g^{sk} \). A message \( m \in \mathbb{Z}_p^* \) is encoded as a pair

\[
E_{pk,sk}(m) = (g^r, pk^r \cdot m),
\]

where \( r \in \{1, \ldots, q\} \) is a random number.

The problem with the current implementation is that \( m \) is allowed to be any integer from \( \{1, \ldots, q−1\} \) which is naturally mapped to an element of \( \mathbb{Z}_p^* \). For semantic security (under the Decisional Diffie-Hellman assumption), the message \( m \) should instead be hashed to one of the \( q \) elements of the group \( Q_p \) generated by \( g \). In the case when \( m \) is not necessarily picked from the group of quadratic residues, the Decisional Diffie-Hellman assumption does not hold. Moreover, such a system is not semantically secure.

If \( m \in Q_p \) is a quadratic residue, then for every choice of randomness of the encryption algorithm \( E_{pk,sk}(m) = (c_1, c_2) \), the second component of its encoding \( c_2 \) is a quadratic residue. Indeed, if \( g, m \in Q_p \), then \( g = x^2, m = y^2 \) for \( x, y \in \mathbb{Z}_p^* \), which implies that

\[
c_2 = pk^r \cdot m = g^{rsk} \cdot y^2 = (x^{rsk}y)^2 \in Q_p.
\]

Similarly, if \( m \) is not a quadratic residue, then \( pk^r \cdot m \) is not a quadratic residue either.

Due to Euler’s criterion, there is an efficient way to distinguish the two cases: \( x \in \mathbb{Z}_p^* \) for a prime \( p > 2 \) is a quadratic residue if and only if \( x^{(p−1)/2} = 1 \). Since this exponentiation can be performed in time \( \text{poly}(\log p) \), this gives an efficient distinguishing attack.

In order to see that the implemented code indeed applies ElGamal to elements outside of the group generated by \( g \), one can apply Euler’s criterion to the published encrypted messages. In Appendix B we provide Python code showing that exactly five out of the ten published messages are quadratic residues modulo \( p \).
3 Example

Due to the lack of documentation and only some code available online, we cannot say for sure which parts of ballots are encrypted. The following lines of code suggest that only the “deputy id” field might be encrypted:

```javascript
var dataToEncrypt = parseInt($(this).data('value'));
...
return signer.getSignedTransaction(votingId, dataToEncrypt, entropy, encryptor);
```

The “deputy id” field stores the id of the voter’s chosen candidate. While the attack distinguishes two identical ballots with different votes with high probability regardless of which parts of the ballot are encrypted, for simplicity in this example we will assume that only the “deputy id” field is encrypted. Also, for this example we take the currently published group $\mathbb{Z}_p^*$ where

$$p = \text{10062759081450625618037903678618826196600591242500860802791085970455088}
29615914188038720723057459046019130152450978128758867982127126946624453
23678201384359740027439588690880234391145675099291004487668846511981135
30933109486902124540395785614572628133031351548262091859360232929939444
1379077427748668682254003.$$

If the number of candidates is $k \leq 4$, and the deputy ids are integers from 1 to $k$, then the adversary can easily count the number of votes cast for candidate number 2. (In particular, if there are only two candidates, then the adversary can count the total number of votes for each of the candidates).

Indeed, since 1, 3 and 4 are quadratic residues modulo $p$, and 2 is not, it suffices to raise the encrypted message to the power $(p−1)/2$ to check whether the vote is cast for candidate number 2. In Appendix C we give Python code emulating the implemented encryption system, and give an example of how to count the number of encryptions of the message 2 among encryptions of messages from 1 to 4.

We emphasize that this attack will distinguish (with high probability) two messages that differ only in their vote, regardless of the values of the deputy ids and which fields of the ballots are encrypted. Let $k$ be the number of candidates. Note that exactly half of the elements of $\mathbb{Z}_p^*$ are quadratic residues. Assuming the uniform distribution of the plain messages we have that the probability of distinguishing a vote for one candidate from the votes for other candidates is $0.5$ for $k = 2$, and is $\frac{k}{2^k}$ for $k > 2$.

---

2See the following links for the code on github: [https://github.com/moscow-technologies/blockchain-voting/blob/1d4f348681e961420a28e8a3a3d64719c4ddc628/voting-form/src/ballot/static/js/forms/mgik/election.js#L108-L123](https://github.com/moscow-technologies/blockchain-voting/blob/1d4f348681e961420a28e8a3a3d64719c4ddc628/voting-form/src/ballot/static/js/forms/mgik/election.js#L108-L123) and [https://github.com/moscow-technologies/blockchain-voting/blob/1d4f348681e961420a28e8a3a3d64719c4ddc628/voting-form/src/ballot/common/module_tpl/election/default/show.tpl#L59](https://github.com/moscow-technologies/blockchain-voting/blob/1d4f348681e961420a28e8a3a3d64719c4ddc628/voting-form/src/ballot/common/module_tpl/election/default/show.tpl#L59)
4 Suggested Fix

The described security vulnerability is a major issue of the implemented cryptographic scheme. We note that the provided attack does not recover the secret key as required by the public testing scenario [Pub19], but rather breaks the system without recovering the secret key.

On the other hand, we believe that there might be an easy way to fix this issue. Since the current implementation already assumes that the message $0 < m < q = (p-1)/2$ (see line 13 in the code block in Appendix [A]), one can use a trivial hashing procedure to map each such message $m$ to the group of quadratic residues.

In the encoding procedure, we suggest to check whether $m(p-1)/2 \equiv -1 \mod p$. If the equality holds, then $m$ is not a quadratic residue, but $-m$ is. Therefore, one can set $m \leftarrow p - m$ and continue the encoding procedure.

In the decoding procedure, an analogous step will be required. If the decoded message $m \geq q$, then it also should be negated ($m \leftarrow p - m$).

Acknowledgements

I would like to thank Pierrick Gaudry and Noah Stephens-Davidowitz for their comments on an earlier version of this note.

References


A Current Implementation of ElGamal

Below we provide the current implementation of the ElGamal encryption in the Internet voting system:

```javascript
async encrypt(data, entropy) {
  if (!data) {
    throw new Error('Data must present!');
  }

  if (!entropy) {
    throw new Error('Entropy must present!');
  }

  const dataAsBI = new BigInt(data.toString());
  const entropyAsBI = new BigInt(entropy.toString());

  if (dataAsBI.compareTo(this.Q) >= 0) {
    throw new Error('Data to encrypt can not be bigger or equal that (P-1)/2!');
  }

  if (entropyAsBI.compareTo(BigInt.ONE) <= 0) {
    throw new Error('Entropy for Session Key must be Integer bigger than 1!');
  }

  if (entropyAsBI.compareTo(this.moduleP) >= 0) {
    throw new Error('Entropy for Session Key can not be bigger or equal Basis Module P!');
  }

  const randomBigInt = await getRandomBigInt(BigInt.ONE,
                                             this.moduleP.subtract(BigInt.ONE));
  const xoredRandomBigInt = randomBigInt.xor(entropyAsBI);
  const sessionKey = trimBigInt(xoredRandomBigInt, this.moduleP.bitLength() - 1);

  const sharedKey = this.publicKey.modPow(sessionKey, this.moduleP);

  const a = this.generatorG.modPow(sessionKey, this.moduleP).toString();
  const b = sharedKey
    .multiply(dataAsBI)
    .remainder(this.moduleP)
    .toString();

  return { a, b };}
```

[https://github.com/moscow-technologies/blockchain-voting/blob/1d4f346881e961420a28e8a3a3d64719c4ddc628/smart-contracts/packages/crypto-lib/src/elGamal.js#L84-L125](https://github.com/moscow-technologies/blockchain-voting/blob/1d4f346881e961420a28e8a3a3d64719c4ddc628/smart-contracts/packages/crypto-lib/src/elGamal.js#L84-L125)
Published Encrypted Messages are Not Quadratic Residues

In this Appendix we use the provided public key and encrypted messages to show that not all messages are quadratic residues in $\mathbb{Z}_p^*$. Here $c$ is the set of the second components of the encrypted messages (that is, each element of the set is $pk^r \cdot m$ where $m$ is some plain message, $pk$ is the public key, and $r$ is a random number). The code shows that only five out of ten elements are quadratic residues (give 1 when raised to the power $(p-1)/2$).

\[
p = 1006275908145065261803790367861882619660509124525008608207210859704550508296159141880387207230754690460191301524050978128758667982127126946264453236782013843\]

\[
q = \frac{(p-1)}{2}
\]

\[
c = [869110015064974622517826385673193618336889788136366496464373338293547389090443974481927922632846987233406362466500520250274346790605883881689706232630528605813829505598477741255501704989450676046755496365836531412743455509369413797345489309461174072514309856175754908122473242156485917832632031302495649, 329945787158463156253342824645839128113015193084449494471583157721729726449185921614274535705666629897986418517050201640312479742720107307075201610948340065358918749914616187769795518055519137612754566546769812037644437522489395754148894267685714188203805416972085863674666599830313728802786163926222734813980, 256054513991066206276662873102021964641362456242148409311459772958496440670168435783150805854518407777279459383097915161681981096625659057692081413077819709806694723689969137957383923173034953045148344118833747707653328275151838997505982992061479564793810225632159787641001956266937123881264751108978786232383202, 30936197551567176269685847042355224083428717175564186238222958588525166667621175580599792978902003702782586887327647891989007741022633330080555036874265367994310710000423997302275417239840497165392742967694429586447995013570761082087769265474707032687773614733617427076860121227155152924933175729529510619903504396708512011344065, 9014495276593656973350536509417605368364753755146175994563182331901683130353994704341622299458002075261525937564517555548559901874042232932422286496051239260442729637671756134870576660535584273031857918168518893590358640489290557062040553070391901730975254660082170709937953363208336695605308414504363789714782355, 91764719158344453102657171361954468495190520510854708634288270771461294908650888052304165093402099109542879591722926613847539079059578167881799170526245000292973902363787636681934271263388852857592304132784015338462708883925877915981265205628726986176578705759761248346470413913994]

https://github.com/moscow-technologies/blockchain-voting/blob/1d4f348681e961420a28e8a3a3d64719c4ddc628/encryption-keys/keys/public-key.json#L1-L5

and https://github.com/moscow-technologies/blockchain-voting/blob/1d4f348681e961420a28e8a3a3d64719c4ddc628/encryption-keys/data/encrypted-datums.csv#L1-L10
Here we show that with high probability one can distinguish encryptions of one message from the encryptions of other messages. In this example we use the currently provided public key for the ElGamal system. We generate a hundred of random encryptions of the messages from 1 to 4. Then without the knowledge of the secret key, we correctly recover the number of encryptions of the message 2.

```python
for i in range(len(c2)):
    print(pow(c2[i], q, p))
```

---

**C Example**

- **Example**: Here we show that with high probability one can distinguish encryptions of one message from the encryptions of other messages. In this example we use the currently provided public key for the ElGamal system. We generate a hundred of random encryptions of the messages from 1 to 4. Then without the knowledge of the secret key, we correctly recover the number of encryptions of the message 2.

```python
for i in range(len(c2)):
    print(pow(c2[i], q, p))
```
def encrypt(data):
    if (data >= q):
        print('Data to encrypt can not be bigger or equal that (P-1)/2!')
    sessionKey = random.randint(1, p-1)
    sharedKey = pow(pk, sessionKey, p)
    a = pow(g, sessionKey, p)
    b = sharedKey * data % p
    return (a, b)

# number of candidates is 4, their ids are {1, 2, 3, 4}.

count = [0] * 5
encryptions = []
for i in range(100):
    candidate = random.randint(1, 4)
    count[candidate] += 1
    encryptions.append(encrypt(candidate))

for i in range(1, len(count)):
    print('Number of votes for candidate %d is %d' % (i, count[i]))

notQResidues = 0
for (a, b) in encryptions:
    if pow(b, q, p) != 1:
        notQResidues += 1

print('
Number of encrypted votes for candidate 2 is %d' % notQResidues)