Homomorphic Tallying for the Estonian Internet Voting System

Arnis Parsovs[1,2]

[1]Software Technology and Applications Competence Centre, Estonia
[2]University of Tartu, Institute of Computer Science, Estonia

August 3, 2016

Abstract

In this paper we study the feasibility of using homomorphic tallying in the Estonian Internet voting system. The paper analyzes the security benefits provided by homomorphic tallying, the costs introduced and the required changes to the voting system. We find that homomorphic tallying has several security benefits, such as improved ballot secrecy, public verifiability of the vote tallying server and the possibility for observers to recalculate the tally without compromising ballot secrecy. The use of modern elliptic curve cryptography allows homomorphic tallying to be implemented without a significant loss of performance.

1 Introduction

Currently, as of the 2015 Parliamentary Elections, Estonian i-voters cast their votes by encrypting the candidate number using the RSA-OAEP encryption scheme and the National Electoral Committee's (NEC) 2048-bit RSA public key. The 256-byte ciphertext is digitally signed using the voter's ID card and sent to the Vote Forwarding Server (VFS), where the digital signature is verified, the signed vote is timestamped and the signed and timestamped vote is forwarded to the Vote Storage Server (VSS) for storage. In the tallying phase, the VSS removes digital signatures from votes and exports ciphertext-only votes to removable media. The media is connected to the offline Vote Tallying Server (VTS) and ciphertexts are sent to the Hardware Security Module (HSM) for decryption. The candidate numbers obtained are tallied and the voting result is announced.

In this paper we study a homomorphic tallying scheme where the current RSA vote encryption scheme is replaced with an additively homomorphic ElGamal encryption scheme and the casting and tallying of votes is modified as follows. The voter casts his vote by producing as many ElGamal ciphertexts as there are candidates in the candidate list. All ciphertexts are encryptions of 0, except for the ciphertext that corresponds to the candidate for whom the voter wants to vote. This ciphertext is an encryption of 1. Every ciphertext is supplemented with cryptographic proof proving that the ciphertext is an encryption of a Boolean value – 0 or 1. This prevents the voter from casting more than one vote for a candidate. The ciphertexts are supplemented with one more proof, proving that the sum of the corresponding plaintexts is 1. This prevents the voter from voting for more than one candidate. The voter signs his district number, the ordered ciphertexts and proofs, and sends the digitally signed vote to the VFS. In addition to verifying the digital signature, the VFS also checks cryptographic proofs inside the vote to verify that the voter has voted only for one candidate (without being able to distinguish the particular candidate). The vote is timestamped and forwarded to the VSS for storage. In the tallying phase, the VSS removes digital signatures from the votes and multiplies all the ciphertexts corresponding to the same candidate. Due to the homomorphic property of the underlying encryption scheme, the multiplication of ciphertexts produces a single ciphertext that contains the number of votes cast for a particular candidate. The aggregated ciphertexts are written onto removable media. The media is connected to the VTS, the VTS sends ciphertexts to the HSM for decryption and shows the number of votes cast for the candidate.

The homomorphic tallying scheme described above is not new. The scheme was introduced in 1997 by Cramer et al. [1] and has been used in the Helios open-audit voting system [2] for years. The contribution of this paper is an analysis of the deployment of homomorphic tallying in the context of Estonian Internet voting, where the performance of the protocol is improved by the use of elliptic curve cryptography.

The paper is structured as follows. Section 2 lists the security benefits of homomorphic tallying compared to the current i-voting scheme used in Estonia, Section 3 describes implementation details along with the benchmark of 256-bit elliptic curves, Section 4 provides efficiency-related statistics from past elections, Section 5 analyzes required changes to the i-voting system, Section 6 discusses the benefits of homomorphic tallying over mix-net-based schemes and Section 7 concludes the paper.

2 Security Benefits

Switching to the homomorphic tallying scheme provides several security benefits. In this section we discuss the benefits in order of significance.

2.1 Public Verifiability of VTS and HSM

In the homomorphic tallying scheme the task of the VTS is simple: to obtain aggregated ciphertext decryptions from the HSM and display the number of votes cast for a candidate. Since the aggregated ciphertexts and corresponding plaintexts cannot be linked back to individual voters, the data can be made public and anyone can verify the correctness of the VTS and the HSM. In order to achieve that, the HSM along with plaintext must return cryptographic proof of a correct decryption. Thus, in case of homomorphic tallying the functionality of the VTS is simplified and the correctness of the VTS and the HSM[1] can be publicly verified.

2.2 Enabler for End-to-End Verifiability

If the correctness of the VTS and the HSM can be publicly verified, then the only major verifiability feature still missing is to verify if the aggregated ciphertexts submitted to the VTS are produced from correctly produced and signed ciphertexts, effectively providing counted-as-cast verifiability[2]. This step could also be made publicly verifiable by publishing the signed votes; however, this approach introduces two security risks.

[1]The proof of correct decryption could be used also in the current scheme, but since the individual decryptions must be kept secret, the proof can be verified only by the VTS.

[2]The verification of ballot box integrity, i.e., proof that all of the cast votes are counted, also needs to be provided.

- **Vote-buying risk.** By observing published votes, a vote-buyer would be able to verify whether or not the vote-seller's vote is revoked or included in the final tally. By publishing the vote verification QR code [3], the voter would be able to present a cryptographic proof about the candidate for whom he voted.
- **Vote privacy risk.** The votes are encrypted using cryptographic algorithms that may be broken after a few decades. For this reason, access to encrypted votes should be limited and the ciphertexts should be destroyed as soon as the results are tallied[3].

Considering these risks, access to signed votes should be limited. Limited public verifiability can be achieved if at least some members of the public (e.g., election observers) are allowed to perform the verification on their hardware. The verification can be performed on the premises of the NEC, where organizational measures are used to prevent data leakage.

2.3 Stronger Ballot Secrecy

Homomorphic tallying improves ballot secrecy, since individual votes are never decrypted. Although at the moment tallying is implemented so that individual vote decryptions are kept secret, there is still a moment where the VTS and the HSM obtain plaintext that can be linked back to the voter. In case of homomorphic tallying the VTS and the HSM operate on aggregated ciphertexts that cannot be linked to individual voters.

2.4 No Invalid Votes

In the past three elections an invalid vote was observed in the tallying phase [4]. The vote contained unexpected plaintext. This gave rise to a variety of speculations on whether the invalid vote was caused by a software bug or whether the vote was intentionally spoiled by encrypting an invalid value [5, Section 3.1]. In case of homomorphic tallying such invalid votes can be identified by the VFS and rejected. If at any time the decision is made to allow casting an empty ballot, an additional "empty ballot" candidate can be introduced or the ciphertext sum proof can be replaced with proof that the sum of ciphertexts is either 0 or 1, thus allowing a voter to cast a vote for no candidate.

2.5 Enabler for Threshold Decryption

The use of the ElGamal cryptosystem allows for efficient distributed key generation and decryption [6]. Currently, the election RSA private key is stored in a single HSM that can be activated by M out of N hardware keys issued to NEC members. The use of a private key split between independent parties would significantly improve ballot secrecy, although it would require overcoming additional organizational challenges. The use of homomorphic tallying also decreases the load on the HSM, since the number of decryptions required depends on the number of candidates and not on the number of votes cast.

[3]It is possible to argue that since votes are sent to the election server over a public Internet connection, different parties are able to capture and store the encrypted traffic forever, which makes this unconditional ballot secrecy requirement unachievable in any case.

3 Implementation Details

3.1 Additive Homomorphic Cryptosystem

As noted before, the ElGamal encryption scheme in additive homomorphic setting, also known as exponential or lifted ElGamal, is used. The ElGamal ciphertext consists of the tuple $(c_1, c_2) = (g^r, h^r g^i)$, where g is the group generator, h is the public key, r is encryption randomness and i is the integer value to be encrypted. To recover a message from the decrypted plaintext, a discrete logarithm must be solved, but since the message space is small (bound by the maximum number of votes that a single candidate can receive), the time required to recover the message is insignificant.

To improve efficiency and decrease the size of the vote, ElGamal over an elliptic curve group is used. Since it is desirable to have a security level of 128 bits that should stay secure at least until 2030 [7], a 256-bit prime order curve (equivalent to a 3072-bit RSA) must be selected. This means that a compressed point on the curve will require 33 bytes (the last byte encoding the sign of the Y coordinate), and thus the full ElGamal ciphertext will consume 66 bytes.

3.2 Zero-knowledge Proofs

As explained above the homomorphic tallying scheme requires three types of proofs: proof of correct decryption (to verify the correctness of the HSM), proof that the ciphertext contains an encryption of either 0 or 1, and proof that the sum of the ciphertexts is 1 (to guarantee that the voter is able to cast only a single vote only for a single candidate).

Proof of correct decryption. The HSM receives an aggregated ElGamal ciphertext tuple (c_1, c_2), which contains the sum of votes given for a candidate. In addition to decrypted plaintext m, the HSM must return zero-knowledge proof proving that plaintext m was correctly obtained by taking c_1 to some exponent x and the result was factored out from c_2 to obtain plaintext m. However, the HSM must also prove that the exponent x used to obtain m is indeed the corresponding private key value from the election key pair, and thus the HSM proves in the same proof that the publicly known public key h is equal to the generator g taken to this exponent x. Thus, the HSM proves without disclosing private key x that $g^x = h$ and ${c_1}^x = \frac{c_2}{m}$. The Chaum-Pedersen proof of discrete logarithm equality [8] is used to prove this statement. The proof is made non-interactive by using the strong form [9] of the Fiat-Shamir transformation [10] and SHA-256 as a hash function.

As a result, complete proof consists of ElGamal ciphertext (c_1, c_2) (66 bytes), plaintext m (32 bytes), challenge c (32 bytes) and response r (32 bytes).

To verify the proof, the commitment values a, b are recovered from c by $a = \frac{g^r}{h^c}$ and $b = \frac{{c_1}^r}{{\frac{c_2}{m}}^c}$ and it is verified that $c = H(g||h||c_1||\frac{c_2}{m}||a||b)$.

Note that since the additive ElGamal scheme is used, the number of votes given for the candidate must be recovered from the plaintext m by brute-forcing exponent i until $g^i = m$, in which case i will be the number of votes given for the candidate.

Proof that the ciphertext encrypts 0 or 1. To prove that the ciphertext contains either $(g^r, h^r g^0)$ or $(g^r, h^r g^1)$ for some randomness r, disjunctive Chaum-Pedersen proof [11] is used. The proof is converted into non-interactive proof using the strong form of Fiat-Shamir transformation and SHA-256 as a hash function.

A complete proof consists of ElGamal ciphertext (c_1, c_2) (66 bytes), challenges c_a, c_b (64 bytes) and responses r_a, r_b (64 bytes) – in total 194 bytes.

The proof is verified by first recovering challenge and commitments: $c = c_a + c_b$, $a_0 = \frac{g^{r_a}}{{c_1}^{c_a}}, a_1 = \frac{h^{r_a}}{{c_2}^{c_a}}, b_0 = \frac{g^{r_b}}{{c_1}^{c_b}}, b_1 = \frac{h^{r_b}}{(c_1/g)^{c_b}}$ and checking that $c = H(g||h||c_1||c_2||a_0||a_1||b_0||b_1)$.

Proof that the sum of ciphertexts is 1. Instead of using zero-knowledge proof to prove that the sum of ciphertexts is 1, the prover can provide the decryption of aggregated ciphertexts. The voter includes in the vote r_{sum} – the sum of individual randomness values used in encryptions (note that r_{sum} does not leak any information about the individual randomness values). The verifier adds up individual ciphertexts and checks whether the aggregated ciphertext is equal to the encryption of 1 using r_{sum} as randomness. The size of r_{sum} is 32 bytes.

3.3 Choice of Elliptic Curve

We evaluated the three most popular 256-bit prime order curves: NIST P-256, secp256k1 and Ed25519. The main evaluation criterion was speed provided by available cryptographic libraries implementing the curve. The speed of implementation is important, since the verification of proofs is an intensive task, computationally. Since private key operations will be performed in the HSM, the side channel protections provided by the benchmarked implementations are of no importance. In fact, increased speed-up could be gained if the libraries provided implementations that did not consider protection against side channel attacks.

NIST P-256. The curve also known as prime256v1 and secp256r1 has been standardized by the NIST and the SECG and is the most widely supported curve in the industry. The curve is used also in the Estonian Mobile-ID for ECDSA digital signatures [12, Section A.2.5]. We benchmarked two `P-256` implementations available in OpenSSL 1.0.2h. Intel's optimization (i) [13] and Google's optimization (g) [14] that can be enabled with flag `enable-ec_nistp_64_gcc_128`.

secp256k1. The curve has been standardized by the SECG and this is the curve used in Bitcoin for transaction signing. The curve has mathematical properties that allow performing point operations faster than in P-256. We benchmarked two secp256k1 implementations – implementation available in OpenSSL 1.0.2h and implementation libsecp256k1 -r699 [15] used by the official Bitcoin software. Implementation libsecp256k1 was compiled with the parameters `USE_NUM_GMP`, `USE_FIELD_5X52_ASM`, `USE_FIELD_INV_NUM`, `USE_SCALAR_INV_NUM`, `USE_SCALAR_4X64` and `USE_ENDOMORPHISM` that were found to provide the best result.

Ed25519. The curve was designed for speed and security against side channel attacks. The curve is birationally equivalent to the ECDH optimized Curve25519, the difference being that Ed25519 supports efficient point arithmetic with the Y coordinate. We benchmarked the 64-bit optimized implementation ed25519-donna -r81 [16]. The implementation was compiled with the parameters `HAVE_UINT128` and `ED25519_64BIT` that were found to provide the best result.

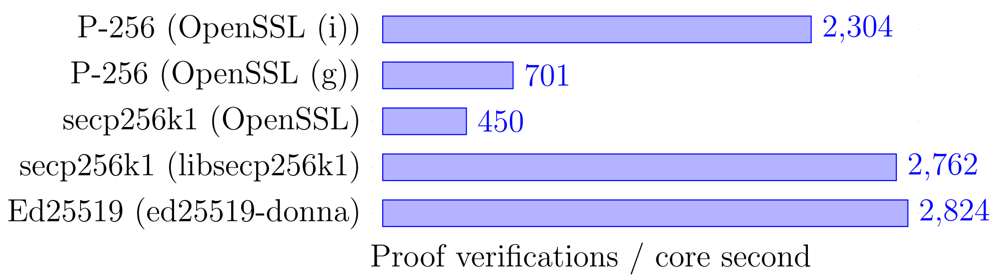

Figure 1: Proof verification speed of 256-bit curve implementations.

Implementations libsecp256k1 and ed25519-donna have been designed for efficient digital signatures using ECDSA and edDSA, respectively. In order to use the implementation for general-purpose point operations, the required functionality had to be constructed using low-level API calls. For the libraries benchmarked in this paper, we have published Python bindings [17] which provide a uniform interface for general-purpose elliptic curve point operations.

In the benchmark we measured the speed of disjunctive Chaum-Pedersen proof verifications (proof that the ciphertext encrypts 0 or 1). Single proof verification involves 2 curve point decompressions, 8 compressions, 6 multiplications, 2 base multiplications, 7 additions, 4 inversions and one SHA256 hash calculation. In all implementations the hash was calculated using OpenSSL.

The benchmark was run on a single core of an Intel i5 6260U@2.90GHz processor embedded in Intel NUC Mini PC NUC6i5SYH. The code was compiled with GCC 5.3.1 using optimization option `-O2`. The results are shown in Figure 1.

We see that the performance of secp256k1 and Ed25519 curve implementation is similar, while curve P-256 implementation is 0.82 times slower. Since the difference in performance is not particularly significant, the choice of curve P-256 might be preferable due to its availability in cryptographic libraries and support by HSM vendors.

4 Relevant Statistics

The efficiency of homomorphic tallying depends on several parameters such as candidates in the candidate list, the total number of candidates and the total number of i-votes cast. Here we present relevant statistics from the last three elections.

KOV2013. In the 2013 Municipal Elections [18], out of 1,086,935 eligible voters, 133,808[4] voters cast their vote using i-voting. In municipal elections voters can vote for candidates who run in their municipality. The total number of candidates running for elections was 14,784. The smallest number of candidates in a municipality was 12 and the largest 437. The weighted average of candidates in the candidate list returned to i-voters was 166.

The maximum number of i-votes given to a single candidate was 3,263. The maximum number of votes per minute submitted to election servers was 134 votes per minute [19, Slide 17].

[4]The number of votes received is usually negligibly larger because of re-voting and the number of votes counted is usually negligibly smaller because some are revoked by a paper vote.

EP2014. In the 2014 European Parliament Elections [20], out of 902,873 eligible voters, 103,151 voters cast their vote using i-voting. All voters chose a candidate from a single country-wide list of 88 candidates.

The maximum number of i-votes given to a single candidate was 17,268. The maximum number of votes per minute submitted to election servers was 86 votes per minute [21].

RK2015. In the 2015 Parliamentary Elections [22], out of 899,793 eligible voters, 176,491 voters cast their vote using i-voting.

In the parliamentary elections voters can vote for candidates who run in their district. The total number of candidates running for elections was 872. The smallest number of candidates in a district was 49 and the largest was 115. The weighted average of candidates in the candidate list returned to i-voters was 80.

The maximum number of i-votes given to a single candidate was 6,423. The maximum number of votes per minute submitted to election servers was 159 votes per minute [23, Slide 43].

From these statistics we can see that the maximum number of votes that has to be stored if all eligible voters i-vote would be $\approx$ 1,100,000, while the current number is $\approx$ 130,000. We can also see that municipal elections are the most complex: there, the total number of candidates is $\approx$ 15,000 and the largest candidate list contains $\approx$ 500 candidates.

5 Required Changes and Efficiency

In this section we list the required changes to the components of the i-voting system and analyze their impact on efficiency.

5.1 Voting Client Application

Instead of encrypting a single candidate number using the 2048-bit RSA, the Voting Client Application (VCA) has to create as many encryptions with proofs as there are candidates in the candidate list. The production of votes in the worst-case-scenario candidate list of 500 candidates takes 0.5 seconds on a single core of the benchmark machine using curve P-256.

In case of the worst-case-scenario candidate list of 500 candidates, the size of the vote would increase by 95 KB ($194 * 500 + 32 = 97,032$ bytes), which is an amount that can be sent to the VFS even over a low-speed mobile Internet connection.

For vote verification, the VCA would have to replace the 20-byte RSA-OAEP encryption padding in the QR code with the 32-byte ElGamal randomness that was used to encrypt the ciphertext that contains the encryption of 1.

5.2 Vote Verification Application

In the current scheme, the Vote Verification Application (VVA) decodes the QR code and extracts the vote identifier and RSA-OAEP randomness used to encrypt the candidate number. Using the vote identifier, the VVA obtains from the VSS the candidate list and the encrypted vote. Using RSA-OAEP randomness, candidate numbers in the candidate list are encrypted one by one, until the produced ciphertext matches the encrypted vote received from the VSS [3]. This brute-force search applied to a candidate list of 500 candidates takes around 5.5 seconds on an Android 2.3.6 device with ARMv6 832 MHz CPU using Bouncy Castle v1.54.

For the homomorphic tallying scheme, the RSA-OAEP randomness in the QR code (40 hexadecimal characters) must be replaced with ElGamal randomness used to encrypt the value 1 (64 hexadecimal characters). To find the candidate, a single ElGamal encryption must be made and the produced ciphertext must be used to find its position in the list of ciphertexts received from the VSS.

While such a single ElGamal encryption using curve P-256 on an Android device described above takes only 0.2 seconds, instead of the fixed 256-byte RSA ciphertext, the VVA must download ciphertext whose size depends on the number of candidates in the candidate list. In the worst-case-scenario, when the candidate list contains 500 candidates, the vote size is 95 KB. To reduce the size to 32 KB, the VSS can return votes without proofs. The vote size can be further reduced to 16 KB by including in the vote only the second member c_2 from the ElGamal ciphertext tuple (c_1, c_2). We can skip the verification of c_1 by observing that the VCA controlled by an attacker cannot come up with randomness r (encoded in the QR code) and randomness r' (used for encrypting 0) such that $c_2 = h^r g^1 = h^{r'} g^0$, unless the attacker can solve a discrete logarithm problem or unless the attacker knows the election private key[5].

To sum up, the homomorphic tallying scheme decreases a varying number of brute-force encryptions to a single encryption, but doubles the download size (assuming the vote and the candidate list are of a similar size).

5.3 Vote Forwarding Server

In addition to digital signature verification, the VFS will also have to verify the proofs included in the vote. A single core of the benchmark machine can process 4.6 votes per second, assuming the worst-case-scenario, where each vote contains 500 proofs, using curve P-256 (see Figure 1).

Taking into account that the VFS will have a faster processor with several cores, that an average vote only contains 170 proofs, and that the maximum number of votes cast per minute so far has only been 159 (in RK2015), then even a single instance of the VFS should be able to easily handle the current stream of votes without introducing a noticeable delay.

5.4 Vote Storage Server

The task of the VSS is to store the votes until elections are over. Assuming that the average candidate list contains 170 candidates, the size of the vote increases by 32 KB ($194 * 170 + 32 = 33,012$ bytes), which in total requires an additional 4 GB of storage in case of 130,000 i-votes cast and 34 GB of storage if all $\approx$ 1,100,000 eligible voters cast their i-vote. While these storage requirements are far from being too high for today's hardware, the increase in vote storage is a challenge for the daily vote backup procedure, which up to now has been implemented by manually writing votes onto a DVD.

After the end of the voting period, the VSS has to produce aggregated ciphertexts by adding up all ciphertexts corresponding to the same candidate. This involves two curve point decompressions and two point additions per ElGamal ciphertext. Assuming that the average candidate list contains 170 candidates and 130,000 i-votes are cast, the summation takes 27 minutes on a single core of the benchmark machine using curve P-256, excluding the overhead introduced by file access and read operations. The size of aggregated ciphertexts is 1 MB, assuming that 15,000 candidates are running for elections.

[5] An attacker who knows the ElGamal private key x can come up with values $r = 0$ and $r' = x^{-1}$ that will satisfy the equation, since $h^0 g^1 = g^{x^{x^{-1}}} g^0$. Therefore, the VVA must reject QR codes with $r = 0$.

5.5 Vote Tallying Server

As mentioned in Section 2.1, the task of the VTS becomes simpler. The VTS is provided with a list of candidates and the corresponding aggregated ciphertexts. The VTS obtains a decryption from the HSM and outputs the number of votes cast for the candidate.

Since the number of votes i cast for a candidate is encoded in plaintext in the form g^i, the VTS has to recover the exponent by solving a discrete logarithm. Since the search space is small, the exponent can be found trivially. So far, the largest number of i-votes cast for a candidate has been 17,268 (in EP2014). It takes only 0.2 seconds to test 17,000 successive exponents on a single core of the benchmark machine using curve P-256.

5.6 Procedures

To use the verifiability features provided by the scheme, two new procedures have to be implemented.

Public verification of VTS and HSM. To verify the correctness of the VTS and the HSM, the NEC would have to publish a file, which contains for every candidate aggregated ciphertexts (66 bytes) and proof of correct decryption (96 bytes). This file can be published on the NEC website, since it leaks no information about individual votes.

Verification of signed votes. To verify that aggregated ciphertexts were produced from correctly formed and signed votes, procedures have to be established for providing observers with access to signed votes for verification using their hardware and software. As a possible measure to prevent data leakage, the verification can be done on the premises of the NEC on the condition that after verification, the hardware onto which the votes were loaded is left on the premises of the NEC for destruction.

Assuming that 130,000 votes are cast and the average candidate list contains 170 candidates, the NEC would have to provide verifiers with a 4 GB medium. The proof verification process will take 2 hours and 40 minutes on a single core of the benchmark machine using curve P-256, plus 5 minutes for 2048-bit RSA signature verifications, plus 27 minutes to produce aggregated ciphertexts (see Section 5.4). The output of the verification procedure is a hash of aggregated ciphertexts, which should match the hash of the aggregated ciphertexts that are published on the NEC website.

As a minimum, the NEC should implement the verification using a machine that has never been connected to the Internet, and, if there is no interest from observers to verify signed votes, buy the verification service from an external entity (e.g., an auditing firm).

6 Related Schemes

The same level of verifiability can be achieved using mix-net-based schemes, where signature-less ciphertexts are rerandomized and decryptions of rerandomized ciphertexts are made public. While a mix-net-based scheme would be more efficient [24, Section 5.2], the proof of correct shuffling used in a mix-net scheme is cryptographically more complex than the disjunctive Chaum-Pedersen proof used in the homomorphic tallying scheme. The cryptographic simplicity of the scheme is important in case of a national voting scheme, since it is desirable that the voting scheme be understandable to and verifiable by as wide an audience as possible.

Another side effect of mix-net-based schemes is an additional mixing party that has to be trusted to not compromise ballot secrecy. Several independent mixing parties can be used in a chain to reduce the risk of ballot secrecy compromise, but the cost is complicating the scheme and introducing additional organizational challenges.

The simple ballot style used in Estonia makes it possible to use homomorphic tallying. For more complicated ballots, the mix-net-based scheme would be the only option.

7 Conclusions

The configuration of the Estonian electoral system allows the Estonian i-voting system to benefit from security and verifiability features provided by a relatively simple homomorphic tallying scheme. The scheme provides counted-as-cast verifiability, improving efficiency in the tallying phase, with the cost of increasing the ballot size by 32 KB on average.

Acknowledgements. This work has been supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science (EXCS), the Estonian National Electoral Committee (NEC) and the Estonian Doctoral School in Information and Communication Technology (IKTDK).

References

[1] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT'97*, pages 103–118, Berlin, Heidelberg, 1997. Springer-Verlag.

[2] Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association.

[3] Sven Heiberg and Jan Willemson. Verifiable Internet Voting in Estonia. In *6th International Conference on Electronic Voting 2014, (EVOTE 2014), October 28-31, 2014, Bregenz, Austria*, pages 23–28, 2014.

[4] Estonian National Electoral Committee. Statistics about Internet Voting in Estonia, 2015. `http://vvk.ee/voting-methods-in-estonia/engindex/statistics`.

[5] Sven Heiberg, Peeter Laud, and Jan Willemson. The Application of I-Voting for Estonian Parliamentary Elections of 2011. In Aggelos Kiayias and Helger Lipmaa, editors, *VOTE-ID*, volume 7187 of *Lecture Notes in Computer Science*, pages 208–223. Springer, 2011.

[6] Torben Pryds Pedersen. *Advances in Cryptology — EUROCRYPT '91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings*, chapter A Threshold Cryptosystem without a Trusted Party, pages 522–526. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[7] NIST. Recommendation for Key Management Part 1: General (Revision 3), July 2012. `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf`.

[8] David Chaum and Torben P. Pedersen. Wallet Databases with Observers. In ErnestF. Brickell, editor, *Advances in Cryptology CRYPTO 92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer Berlin Heidelberg, 1993.

[9] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer Berlin Heidelberg, 2012.

[10] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 186–194, London, UK, UK, 1987. Springer-Verlag.

[11] Ronald Cramer, Ivan Damgrd, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In YvoG. Desmedt, editor, *Advances in Cryptology CRYPTO 94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer Berlin Heidelberg, 1994.

[12] AS Sertifitseerimiskeskus. Certificates on the personal identification documents of the Republic of Estonia. v5.0, 2015. `https://sk.ee/upload/files/SK-CPR-ESTEID-EN-v5_0-20150101.pdf`.

[13] Shay Gueron and Vlad Krasnov. Fast prime field elliptic-curve cryptography with 256-bit primes. *J. Cryptographic Engineering*, 5(2):141–151, 2015.

[14] Emilia Käsper. Fast elliptic curve cryptography in openssl. In *Proceedings of the 2011 International Conference on Financial Cryptography and Data Security*, FC'11, pages 27–39, Berlin, Heidelberg, 2012. Springer-Verlag.

[15] Pieter Wuille. Optimized C library for EC operations on curve secp256k1, SVN commit r699, May 2016. `https://github.com/bitcoin-core/secp256k1`.

[16] Andrew Moon. Implementations of a fast Elliptic-curve Digital Signature Algorithm, SVN commit r81, March 2015. `https://github.com/floodyberry/ed25519-donna`.

[17] Arnis Parsovs. Python bindings for general purpose elliptic curve point operations, May 2016. `https://github.com/user8547/fast-ecc-python`.

[18] Estonian National Electoral Committee. Municipal Elections 2013 Results, 2013. `http://kov2013.vvk.ee/`.

[19] Tarvi Martens. I-voting developments, 2013. `https://sk.ee/upload/files/AK2013_Tarvi%20Martens.pdf`.

[20] Estonian National Electoral Committee. European Parliament Elections 2014 Results, 2014. `http://ep2014.vvk.ee/detailed-en.html`.

[21] Tarvi Martens. Personal communication, October 2014.

[22] Estonian National Electoral Committee. Parliamentary Elections 2015 Results, 2015. `http://rk2015.vvk.ee/voting-results.html`.

[23] Tarvi Martens. On Internet Voting, 2015. `http://vvk.ee/public/RK2015/Slides_for_Election_Observers.pdf`.

[24] Oksana Kulyk and Melanie Volkamer. Efficiency comparison of various approaches in e-voting protocols. In *Financial Cryptography, 1st Workshop on Advances in Secure Electronic Voting Schemes (VOTING'16)*, February 2016.