Two Simple Code-Verification Voting Protocols*

Helger Lipmaa

[1] Cybernetica AS, Estonia
[2] Tallinn University, Estonia

Abstract. Norwegian nationwide Internet voting will make use of a setting that we will call the code-verification voting. The concrete protocol that will be used in Norway was proposed by Scytl and improved by Gjøsteen. As we show, Gjøsteen's protocol has several undesirable properties. In particular, one of the online servers shares the secret key with the offline tallier. Even without considering that, the coalition of two online voting servers can breach voter privacy. We propose two new code-verification voting protocols. The first protocol separates the secret keys, and is as efficient as Gjøsteen's protocol. The second protocol provides voter privacy against the coalition of online voting servers but is somewhat less efficient. While the new protocols are more secure than the protocol that is going to be used in the Norwegian nationwide Internet voting, they are based on the same setting, so as to minimize the required infrastructural changes.
Keywords. Code-verification voting, Internet voting, malicious voter PC.

1 Introduction

Several nations are moving towards using Internet voting to elect their political representatives. One can use any of standard cryptographic protocols, coupled with relevant organizational means, to secure Internet voting against malicious voting servers. Instead, the weakest link is the voter PCs that may be corrupted by various malware. It is important to achieve Internet voting security even in such a case, without considerably changing the user experience or introducing unreasonable assumptions that may hamper accessibility (like requiring all voters to perform expensive mental arithmetics, entering long pseudorandom numbers, or to own special PIN-calculators; we omit citations).

Heiberg, Lipmaa and Van Laenen [HLV10] proposed a concrete Internet voting setting where the voters can verify that their votes reached the central voting servers even when the voter PCs are malicious. In this *code-verification voting* setting (that will be used in the Norwegian nationwide Internet voting), there are two extra out-of-the-band channels, a prechannel and a postchannel, that are completely independent of the voter PCs. Before the Internet voting period starts, voter v obtains via the prechannel a code sheet that lists all candidates c with the corresponding voter-dependent random integrity check codes $\mathsf{Code}_v[c]$. After that, during the actual Internet voting, the voter v selects a concrete candidate number c^*. He inputs c^* to his PC by using a suitable user interface, and the PC sends encrypted c^* to the vote collector. The correspondence between c^* and the actual candidate is publicly known, and thus there is no need for the voter to enter a long pseudo-random string as in say code voting [Cha01]. It was strongly felt by Norwegian authorities that the usability advantages of code-verification voting are more important than the security advantages of code voting.

The vote collector executes a suitable cryptographic protocol with another central server, the messenger, after which the messenger obtains the value $\mathsf{Code}_v[c^*]$. Finally, the messenger sends the integrity check code $\mathsf{Code}_v[c^*]$ to the voter v over the postchannel (for example, by sending an SMS). In parallel, the vote collector stores all encrypted values of c^*. To limit the efficiency of coercion, the code-verification setting also supports revoting, after which the vote collector stores the encrypted value of the "new" version of c^*. Due to the possibility of revoting, the postchannel must be independent of the voter PCs. At the end of the allotted Internet voting period, the vote collector forwards encrypted votes to the (offline) tallier, who then decrypts and tallies the votes. One can use various existing cryptographic protocols to ensure the correctness of the last step. See [HLV10] for more discussion about the setting.

Two different code-verification voting protocols have been proposed thus far. In [HLV10], Heiberg, Lipmaa and Van Laenen proposed a protocol where the codes $\mathsf{Code}_v[c]$ are uniformly random. The voter PC sends an encrypted candidate number c to the vote collector, who applies a so called proxy oblivious transfer protocol [HLV10] to the ciphertext, obtaining a ciphertext of $\mathsf{Code}_v[c]$, and sends the result to the messenger. The computational complexity of the proxy oblivious transfer protocol is linear in the number Γ of candidates, and becomes relatively

* Draft, June 16, 2011. The result dates back to Spring 2010.

inefficient for already say $\Gamma > 5$. In particular, [HLV10] did not specify how to make their protocol secure against malicious voting servers, assuming that the vote collector is semihonest; security against a malicious vote collector can be achieved by using standard yet relatively inefficient techniques. Moreover, in this protocol it is possible that the online servers (the vote collector and the messenger) collaborate to breach the voter privacy; in this case, implementing a secure tallier (by say using mixnets) would be a clear overkill.

Norwegian nationwide e-voting will use alternative code-verification protocol by Gjøsteen [Gjø10].[1] In this protocol, the integrity check codes are not random, but pseudorandom. More precisely, every code $\mathsf{Code}_v[c]$ is computed as a composition of three pseudorandom functions, respectively "owned" by the voter PC, vote collector and by the messenger. Due to this, Gjøsteen manages to design an efficient code-verification voting protocol with computational complexity that does not depend on the number of candidates. As a tradeoff, Gjøsteen's protocol requires a setup phase, where a number of servers, who share the secrets of all voter PCs, the vote collector, and the messenger, precompute the integrity check codes so as they can be forwarded to voters over the prechannel. (This phase is not precisely specified in [Gjø10].)

To increase efficiency, Gjøsteen's protocol uses a few additional tricks. First, the coalition of two online servers (the vote collector and the messenger) shares the secret key of the tallier (an offline server, that can be implemented in a distributed fashion by using say mixnets). This may become a serious accountability problem, especially since the coalition of online servers can completely breach voter privacy.[2] If the secret keys are separated, then the Gjøsteen's protocol automatically becomes less efficient (intuitively, this is due to the fact that as in the protocol of [HLV10], then the voter PC has to encrypt the candidate number by using two different public keys, and then prove in zero-knowledge that this was done correctly). Moreover, even without sharing the tallier's secret key, the coalition of two online servers, the vote collector and the messenger, can breach the voter privacy, since they can verify whether the output of a voter PC is equal to the application of the first pseudorandom function to any concrete candidate. See Sect. 3 for more discussion on both existing protocols.

Our Contributions. We propose two new code-verification voting protocols that are both more secure than Gjøsteen's protocol. The first protocol does not provide privacy against the coalition of two malicious online voting servers, but is computationally more efficient than the second protocol due to a more efficient non-interactive zero-knowledge (NIZK) proof. It is however somewhat more secure than Gjøsteen's protocol since the online servers do not share tallier's secret key. The second protocol guarantees said privacy, while being computationally less efficient. In both cases, we adapt Gjøsteen's main optimization (that is, the integrity check codes are computed as superpositions of independent pseudorandom function families), but we construct the whole protocol in a way that facilitates efficient construction of NIZK proofs that are needed to achieve security in malicious model.

As the code-verification protocols from [HLV10,Gjø10], the new protocols use the Elgamal cryptosystem [Elg85] over a prime-order cyclic group $\mathbb{G}$. The choice of Elgamal is motivated by practical considerations: first, by its homomorphic properties that facilitate efficient protocol design, second by its efficiency (particularly over elliptic curves), and third, since it is sufficiently standard to be supported by Hardware Security Modules. We also need an arbitrary secure pseudorandom function family, and an arbitrary secure signature scheme. In the second protocol, we also need Pedersen's commitment scheme. Moreover, as in [HLV10,Gjø10], we only require two online voting servers, the vote collector and the messenger.

In the new code-verification voting protocols, integrity check code is computed by two (and not three, as in the protocol of [Gjø10]) parties, the voter PC and the messenger. On the other hand, the messenger and the tallier have independent keys (unlike in [Gjø10]), and in the second new protocol, the coalition of the vote collector and the messenger cannot decrypt the votes (unlike both in [HLV10] and [Gjø10]). It it also easy to implement the messenger by using more than one 1 servers, such that every server applies its own pseudorandom function to the output of the previous one, thus decreasing dependency on a single messenger server.

Both new protocols separate the keys of the online servers and the tallier, thus making it reasonable to implement the tallier in a distributed way. (If the online servers share tallier's secret key, then using secure multi-party computation to implement tallier's operations would clearly be an overkill.) The main difference between the two new protocols is that the first protocol does not protect voter privacy against the coalition of two online servers, the vote collector and the messenger, while the second does. On the other hand, the first protocol is about twice as

[1] This protocol was designed by Scytl, and then modified by Gjøsteen. However, since [Gjø10] does not clarify the precise contribution of Scytl, we will call it Gjøsteen's protocol.

[2] Norwegian government seems to be aware also of this problem, see [Bul10, slide 18], where it is stated that to alleviate this issue, the servers will be separated logically, geographically (by 600 km) and organizationally (in two different authorities, under two different ministries).

Protocol	Voter PC	Vote Collector	Messenger	Setup phase
[HLV10]	$(7\gamma + 10) \cdot e + 1 \cdot s$	$(2\Gamma + 6\gamma + 8) \cdot e + 1 \cdot v + 1 \cdot s$	$\Gamma \cdot e + 1 \cdot v$	No
[Gjø10]	$3 \cdot e + 1 \cdot s$	$8 \cdot e + 1 \cdot v + 1 \cdot s$	$10 \cdot e + 1 \cdot v$	Yes
Sect. 4	$12 \cdot e + 1 \cdot s$	$9 \cdot e + 1 \cdot v + 1 \cdot s$	$10 \cdot e + 2 \cdot v$	Yes
Sect. 5	$16 \cdot e + 1 \cdot s$	$17 \cdot e + 1 \cdot v + 1 \cdot s$	$18 \cdot e + 2 \cdot v$	Yes

Table 1. The computational complexity of the protocols from [HLV10], [Gjø10] and of the two protocols from the current paper. The numbers for [HLV10] do not include the cost of a NIZK proof from the vote collector to the messenger. The protocol from [Gjø10] assumes that the messenger and the tallier share a common secret/public key, which makes the protocol about twice more efficient than it would be otherwise.

efficient as the second protocol. See Sect. 4 for a precise description—including the security analysis—of the first protocol. The differences between the first and the second protocol are outlined in Sect. 5.

We present short and informal proofs that both new protocols (if applied in combination with a secure cryptographic shuffle and mixnets) are secure against any single malicious party (the voter PC, the vote collector, the messenger, and the tallier). More precisely, if the voter PC is malicious, the protocol remains correct. If the vote collector or the messenger is malicious, then the protocol remains correct and private. If the tallier is malicious and the shuffle is secure, then the protocol remains correct. The protocol is not universally verifiable, since it involves revoting to protect against coercion. However, all voters can verify that their own last vote was included. We also show that if both online voting servers (the vote collector and the messenger) are malicious and collaborate, then the second (but not the first) new protocol remains private.

Efficiency comparison of the protocols from [HLV10], [Gjø10] and the current paper is given in Table 1. The numbers are given by a vote attempt, and thus we have not counted in the cost of the subprotocol between the vote collector and the tallier (which may say include a cryptographic shuffle in all 3 cases), since this is done after the end of the Internet voting period. Here, the constant before e/v/s denotes the number of exponentiations, signature verifications and signings, respectively. Throughout this paper, Γ is the number of candidates, and $\gamma = \lceil \log_2 \Gamma \rceil$. In the case of [HLV10], we have not included the cost of the likely expensive NIZK proof that the vote collector follows the proxy oblivious protocol. On the other hand, the protocol of [HLV10] includes another NIZK proof that the candidate number c^* that the voter PC encrypted belongs to the set of valid candidates; the rest of the protocols have no such NIZK proof.

In the case of [Gjø10], we emphasize that Gjøsteen's protocol achieves somewhat better efficiency due to the fact that the messenger and the tallier share the public/secret key, and thus there is no need to construct and verify a NIZK proof that the voter PC has encrypted the same value with two public keys. As mentioned before, we find that this solution is bad from the accountability standpoint.

Possible criticisms. Recall that Gjøsteen's protocol will be used in Norwegian nationwide Internet voting. While it is considerably more efficient than the code-verification protocol from [HLV10], it also suffers from a number of weaknesses. Our paper addressed some of the weaknesses (while being practically as efficient), but it still shares some others. E.g., as in the case of Gjøsteen's protocol, *all* voters share the same secret key. The difference is that in our protocol, if the coalition of two online servers want to breach voter privacy, they have to obtain that key on top of their own secret keys, while in Gjøsteen's protocol, the servers can just mutually share their own secret keys.

Another important shortcoming is the need for the voters to obtain the table of verification codes before the start of the Internet voting period. Again, this weakness is shared with Gjøsteen's protocol, and the Norwegian authorities have already worked on corresponding trusted infrastructure to alleviate this problem [Bul10].

It is an interesting open problem to eliminate the two above mentioned weaknesses without blowing up the computation. Our work can ben seen as a step in this direction.

2 Preliminaries

Notation. All logarithms are on basis 2. k is the security parameter, we assume that $k = 80$. $x \leftarrow X$ denotes assignment; if X is a set or a randomized algorithm, then $x \leftarrow X$ denotes a random selection of x from the set or from the possible outputs of X as specified by the algorithm. In the case of integer operations, we will explicitly mention the modulus, like in $z \leftarrow a + b \mod q$. Γ denotes the number of candidates.

Hash Functions and Random Oracle Model. Function $H : A \to B$ is a hash function if $|B| < |A|$. We usually need to assume that H is a random oracle. I.e., the value of $H(x)$ is completely unpredictable if one has not seen $H(x)$ before. In practice, one would instantiate H with a strong cryptographic hash function like SHA2 or the future winner of the SHA3 competition. While there exist cryptographic protocols and primitives which are secure in the random oracle model but which are insecure given any "real" function [CGH98], all known such examples are quite contrived.

Signature Schemes. A signature scheme $\mathsf{SC} = (\mathsf{Gen^{sc}}, \mathsf{Sign}, \mathsf{Ver})$ is a triple of efficient algorithms, where $\mathsf{Gen^{sc}}$ is a randomized key generation function, Sign is a (possibly randomized) signing algorithm and Ver is a verification algorithm. A signature scheme is EUF-CMA (existentially unforgeable against chosen message attacks) secure, if it is computationally infeasible to generate a valid signature to a message that was not queried from the oracle, given access to an oracle who signs messages chosen by the adversary. In the new protocols, any of the well-known EUF-CMA secure signature schemes can be used. However, since Internet voting is most probably going to use the existing PKI infrastructure of the relevant country, the most prudent approach is to rely on whatever signature scheme has been implemented in the corresponding ID-cards.

Public-Key Cryptosystems. Let $\mathsf{PKC} = (\mathsf{GenPkc}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key cryptosystem, where GenPkc is a randomized key generation algorithm that on input 1^k, for randomizer r, outputs a new secret/public key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{GenPkc}(1^k)$, Enc is a randomized encryption algorithm with $c = \mathsf{Enc}_{\mathsf{pk}}(m; r)$, and Dec is a decryption algorithm with $\mathsf{Dec}_{\mathsf{sk}}(c) = m'$. If $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{GenPkc}(1^k)$, then $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m; r)) = m$ for all valid m and r. We denote $\mathsf{Enc}_{\mathsf{pk}}(m; r)$ for a randomly chosen r by $\mathsf{Enc}_{\mathsf{pk}}(m)$. A public-key cryptosystem is CPA-secure if the ciphertext distributions corresponding to any two plaintext messages are computationally indistinguishable.

In the Elgamal cryptosystem [Elg85], one fixes a cyclic group $\mathbb{G}$ of prime order $2^{2k+1} > q > 2^{2k}$, together with a generator g of $\mathbb{G}$. Then, $\mathsf{GenPkc}(1^k)$ generates a random $\mathsf{sk} \leftarrow \mathbb{Z}_q$, and sets $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$. On input $m \in \mathbb{G}$, the encryption algorithm generates a new random $r \leftarrow \mathbb{Z}_q$, and sets $\mathsf{Enc}_{\mathsf{pk}}(m; r) := (m \cdot \mathsf{pk}^r, g^r)$. On input $c = (c_1, c_2) \in \mathbb{G}^2$, the decryption algorithm outputs $m' \leftarrow c_1 / c_2^{\mathsf{sk}}$. Elgamal is multiplicatively homomorphic. That is, $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m_1; r_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(m_2; r_2)) = m_1 \cdot m_2$ for all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{GenPkc}(1^k)$, and all m, r_1, r_2. The Elgamal cryptosystem is CPA-secure under the decisional Diffie-Hellman assumption.

Commitment Schemes. Let $\mathsf{COM} = (\mathsf{GenCom}, \mathsf{Com}, \mathsf{Open})$ be a commitment scheme, where GenCom is a randomized key generation algorithm that on input 1^k outputs a new public key $\mathsf{pk} \leftarrow \mathsf{GenCom}(1^k)$, Com is a randomized encryption algorithm with $(c, d) = \mathsf{Com}_{\mathsf{pk}}(m; r')$ where c is the commitment and d is the decommitment value, and Open is an opening algorithm with $\mathsf{Open}_{\mathsf{pk}}(c, d) = m'$. It is required that if $\mathsf{pk} \leftarrow \mathsf{GenCom}(1^k)$, then $\mathsf{Open}_{\mathsf{pk}}(\mathsf{Com}_{\mathsf{pk}}(m; r')) = m$ for all valid m and r'. We denote $\mathsf{Com}_{\mathsf{pk}}(m; r)$ for a randomly chosen r also just as $\mathsf{Com}_{\mathsf{pk}}(m)$. A (statistically hiding) commitment algorithm is required to be statistically hiding (that is, the distributions of commitments of any two messages are statistically close) and computationally binding (no polynomial-time adversary can open a commitment to two different messages, except with a negligible probability).

The Pedersen commitment scheme [Ped91] works in the same setting as the Elgamal cryptosystem, in a group $\mathbb{G}$ of prime order q, with generator g. Here, $\mathsf{GenCom}(1^k)$ generates a random $\mathsf{pk} \leftarrow \mathbb{G}$. On input $m \in \mathbb{G}$, the commitment algorithm generates a random $r \leftarrow \mathbb{Z}_q$, and sets $\mathsf{Com}_{\mathsf{pk}}(m; r) := (m \cdot \mathsf{pk}^r, r)$. Thus, $m \cdot \mathsf{pk}^r$ is the commitment value, and $d = r$ is the decommitment value. On input (c, r), the opening algorithm outputs $\mathsf{Open}_{\mathsf{pk}}(c, r) = c / \mathsf{pk}^r$. If the decisional Diffie-Hellman assumption holds, then the Pedersen commitment scheme is statistically hiding and computationally binding.

Non-Interactive Zero-Knowledge Proof of Knowledge. Let L be an arbitrary NP-language, and let $R = \{(x, y)\}$ where $x \in L$ and y is the corresponding NP-witness. A Σ-protocol (P_1, V_1, P_2, V_2) for a relation R is a three-message protocol between a prover and a verifier (both stateful), such that (1) the prover and verifier have a common input x, and the prover has a private input y, (2) the prover sends the first (P_1) and the third (P_2) message, and the verifier sends the second message V_1, after which the verifier either rejects or accepts (by using V_2), (3) the protocol is public-coin: i.e., the verifier chooses her response V_1 completely randomly from some predefined set, (4) the protocol satisfies the security properties of correctness, special honest-verifier zero-knowledge (SHVZK), and special soundness. A protocol run is the tuple $(x; \mathfrak{i}, \mathfrak{c}, \mathfrak{r})$ where $(\mathfrak{i}, \mathfrak{c}, \mathfrak{r})$ are the three messages of this protocol. A protocol run is *accepting*, if an honest verifier accepts this run, on having input x and seeing the messages $\mathfrak{i}$, $\mathfrak{c}$, and $\mathfrak{r}$.

More precisely, a Σ-protocol is *correct* if for any $(x, y) \in R$, an honest verifier accepts all runs with an honest prover. A Σ-protocol has the property of *special soundness* if one can construct an efficient simulator that, given any two accepting runs $(x; \mathfrak{i}, c_1, \mathfrak{r}_1)$ and $(x; \mathfrak{i}, c_2, \mathfrak{r}_2)$ with $c_1 \neq c_2$, outputs a y such that $(x, y) \in R$. A Σ-protocol has the property of SHVZK if there exists an efficient simulator that, given as input an arbitrary $x \in L$ (without corresponding y), can construct accepting runs $(x; \mathfrak{i}^*, \mathfrak{c}^*, \mathfrak{r}^*)$ such that (a) the simulator starts by choosing $\mathfrak{c}^*$ and

$\mathfrak{r}^*$, and only then computes $\mathfrak{i}^*$, and (b) the resulting distribution $(x; \mathfrak{i}^*, \mathfrak{c}^*, \mathfrak{r}^*)$ is computationally indistinguishable from the distribution $(x; \mathfrak{i}, \mathfrak{c}, \mathfrak{r})$ of runs between an honest prover and an honest verifier.

Based on an arbitrary Σ-protocol, one can build a non-interactive zero-knowledge (NIZK) proof of knowledge in the random oracle model, by using the Fiat-Shamir heuristic [FS86]. I.e., given $(x, y) \in R$ and a random oracle H [BR93], the corresponding NIZK proof of knowledge π is equal to $(\mathfrak{i}, \mathfrak{c}, \mathfrak{r})$, where $\mathfrak{i} \leftarrow P_1(x, y)$, $\mathfrak{c} \leftarrow H(param, x, \mathfrak{i})$, and $\mathfrak{r} \leftarrow P_2(x, y, \mathfrak{c})$, where $param$ is the set of public parameters (like the description of the underlying group, etc). See [CDS94]. We use the next common notation. A NIZK proof of knowledge $\mathrm{PK}(\alpha, \cdots : R(\dots))$ is for relation R, where the prover has to prove the knowledge of variables denoted by Greek letters $\alpha, \dots$. All other variables are known to both the prover and the verifier. E.g., $\mathrm{PK}(\mu, \rho : y = \mathsf{Enc}_{\mathsf{pk}}(\mu; \rho) \wedge \mu \in \{0, 1\})$ denotes a NIZK proof of knowledge that the prover knows a Boolean μ and some ρ such that $y = \mathsf{Enc}_{\mathsf{pk}}(\mu; \rho)$.

3 Previous Code-Verification Voting Protocols

The Heiberg-Lipmaa-Van Laenen Protocol. We provide a longer description of the Heiberg, Lipmaa and Van Laenen [HLV10] code-verification protocol in App. A.

The Heiberg-Lipmaa-Van Laenen protocol provides security against any single party. More precisely, it achieves privacy against a malicious vote collector or a messenger (but not against their coalition, or against the voter PC). It provides correctness in the presence of either a malicious voter PC, a semihonest vote collector (correctness against a malicious vote collector is achievable, but costly), and a semihonest messenger, but not against their coalitions. One can achieve additional privacy and correctness by including a cryptographic shuffle. See [HLV10] for more details.

Gjøsteen's Protocol. Recently, Gjøsteen [Gjø10] proposed a more efficient Internet voting protocol in the same setting, where the (online) computational complexity of the protocol does not depend on Γ. In Gjøsteen's protocol, the values $\mathsf{Code}_v[c]$ are not random, but computed as a superposition of three pseudorandom functions: one pseudorandom function by the voter PC, another one (exponentiation by a random integer y_v) by the vote collector (named the ballot box in [Gjø10]), and the third one by the messenger (named the receipt generator in [Gjø10]). Thus, the final pseudorandom mapping $c \mapsto \mathsf{Code}_v[c]$ is a composition of those three pseudorandom functions. Due to this trick, one does not have to use the costly proxy oblivious transfer protocol, and in particular, the computational complexity of Gjøsteen's protocol does not depend on Γ. The revoting part is handled similarly as in [HLV10]. Currently, it seems that Gjøsteen's protocol will be used in Norwegian Internet voting.

As a drawback compared to the Heiberg-Lipmaa-Van Laenen protocol, Gjøsteen's protocol requires an offline setup phase, where a number of trusted servers—the process is not specified in [Gjø10]—compute the values $\mathsf{Code}_v[c]$ for every c and v. To do this, the setup servers must possess all secrets, required to evaluate all three pseudorandom functions. On the one hand, presence of such a setup phase seems to be necessary for the Gjøsteen's optimization to work. On the other hand, the setup servers have to be highly secured, and they also must have secure access to the prechannel but also secure channels to all voter PCs, the vote collector, and the messenger, to forward them the corresponding secrets. We emphasize however, that since the setup phase is an offline step, one can use time-consuming cryptographic techniques (without going into details, secure multi-party computation) to secure the operation of the setup servers. To secure the channels, one must use organizational means decoupled with strong cryptography. As in [Gjø10], we will omit further discussion of the setup phase. The Norwegian government seems to be aware of these problems, see [Bul10, slide 16], where it is stated that pre-election sharing of secrets is cumbersome and vulnerable. However, they see it as a necessary evil. *The setup phase will also be present in the new code-verification voting protocols of the current paper.*

As another drawback, in Gjøsteen's protocol it is assumed that the coalition of two online servers (the messenger and the vote collector) share tallier's secret key, which makes the protocol less accountable since the coalition of malicious online servers can sometimes accuse honest tallier, and vice versa. Another obvious attack is when a dishonest vote collector sends the votes to the dishonest messenger for decryption. (The same attack can also be applied to the Heiberg-Lipmaa-Van Laenen protocol.) Thus, the privacy can be breached even when the tallier is honest. This is a serious problem in practice, since in the real Internet voting systems, the vote tallying should be performed offline, and thus it is not so easy to attack the tallier. Moreover, the functions of the tallier can be distributed by using say mix-nets, and the computational cost of this is not so important since tallying is done after the Internet voting period ends. However, the vote collector and the messenger are online entities, and distributing them would be computationally costly. Thus, we emphasize that in an Internet voting protocol, malicious *online* servers should not be able to breach the voter privacy.

Interestingly, even if the messenger and the tallier did not share the same key pair, it is still possible for the coalition of online servers to breach the voter privacy, since together they can compute the output $y = f(c)$

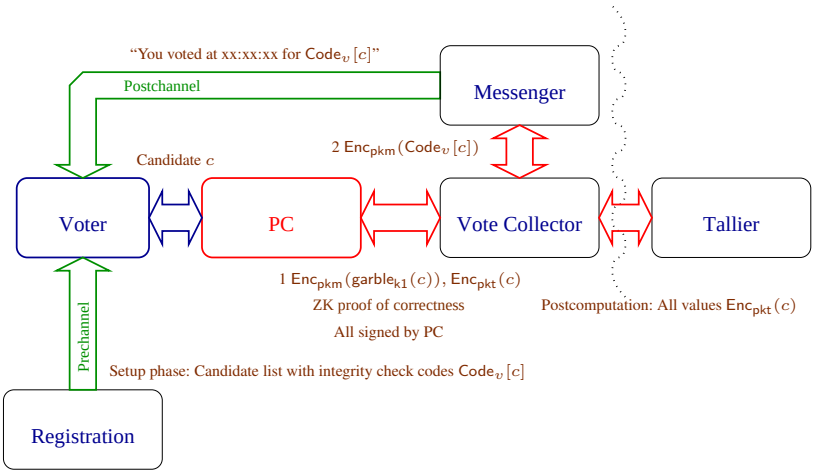

Fig. 1. The first new code-verification protocol, simplified

of the first pseudorandom function (the one, computed by the voter PC) on c, and they can just check whether $y = f(c^*)$ for some candidate c^*. One could try to solve this problem by assuming that every voter PC has a different pseudorandom function f_v, not known by the online servers, but known by the tallier. However, in this case the tallier would obtain voter-dependent values $f_v(c)$ which is bad from the privacy viewpoint.

We now present a slightly more precise description of Gjøsteen's protocol:

1. The voter's PC Elgamal-encrypts (the garbled) vote $f(c)$ (the function f is not precisely specified in [Gjø10]) by using a product (Elgamal) public key $\mathsf{pkt} = \mathsf{pkvc} \cdot \mathsf{pkm}$ of public keys of the vote collector and the messenger. The result is $\mathsf{Enc}_{\mathsf{pkt}}(f(c)) = \mathsf{Enc}_{\mathsf{pkm}\cdot\mathsf{pkvc}}(f(c); r) = (f(c)\mathsf{pkt}^r, g^r)$ for random r.
2. The vote collector decrypts the ciphertext partially, by applying the knowledge of her secret key skvc, and then takes the resulting ciphertext $\mathsf{Enc}_{\mathsf{pkm}}(f(c)) \leftarrow (f(c)\mathsf{pkt}^r/(g^r)^{\mathsf{skvc}}, g^r) = (f(c)\mathsf{pkm}^r, g^r)$ to the power x_v, where x_v is a voter-dependent secret key. He sends $\mathsf{Enc}_{\mathsf{pkm}}(f(c)^{x_v}) = (f(c)^{x_v}\mathsf{pkt}^{rx_v}, g^{rx_v})$ to the messenger.
3. The messenger decrypts the message, and obtains $f(c)^{x_v}$. He applies another pseudorandom function h to this, obtaining $h(f(c)^{x_v})$.

To achieve security in the malicious model, the second step has to be accompanied by a NIZK proof of correctness. This NIZK proof is somewhat expensive — while it was not specified in [Gjø10], by our calculations, it requires the prover (the voter PC) to perform 5 exponentiations, and the verifier to perform 9 exponentiations. In the case the keys of the tallier and the messenger were separated to tackle the mentioned accountability problem, it seems that the vote collector would also have to partially decrypt (and present a NIZK proof of correctness) the ciphertext that the voter PC meant to send to the tallier. Finally, Gjøsteen's paper [Gjø10] does not specify the secret keys, or what pseudorandom functions will really be used. In the new protocols, we have tried to be more precise on such accounts.

4 First New Code-Verification Protocol

In this section, we propose the first new code-verification protocol. It is more efficient than the second one, but it does not provide privacy against the coalition of online servers. A *simplified* version of this protocol is depicted by Fig. 1.

Setting. Let $\mathsf{garble} = \{\mathsf{garble}_{\mathsf{k1}}\}$ be a public pseudorandom function family where every $\mathsf{garble}_{\mathsf{k1}}$ is a function from the set of all candidates $\{c\}$ to $\mathbb{G}$. We implement $\mathsf{garble}_{\mathsf{k1}}$ as $\mathsf{garble}_{\mathsf{k1}}(c) := g^{\mathsf{AES}_{\mathsf{k1}}(c)}$, where g is a fixed generator of $\mathbb{G}$. Here, $\mathsf{k1}$ is not known by the messenger and the tallier[3], while it is known to the voter PCs. All voter PCs v have a secret key $x_v \leftarrow \mathbb{Z}_p$. The voter PCs publish the values $h_v \leftarrow g^{x_v}$. Let prf be another public pseudorandom function family (in practice, AES followed by truncation) from $\mathbb{G}$ (in practice, from some one-to-one bit-representation of the elements of $\mathbb{G}$) to the set of codes. The messenger also has a secret key $\mathsf{k2}$ for prf.

[3] This can be achieved by organizational security. Similar assumptions are made in [Gjø10].

Setup phase. As in [Gjø10], the new protocols need a setup phase. During the setup phase—before the Internet voting starts—, some trusted third parties compute jointly all the integrity check codes. Since this process must finish before Internet voting starts, one can use any of the existing secure multi-party computation based protocols, by using a set of independent setup servers, in this phase. (See [Gjø10] for a discussion. As already mentioned, the Norwegian government is aware of the accompanying problems, and has accepted them due to the lack of efficient setup-less solutions.) More precisely, one integrity check code $\mathsf{Code}_v[c]$ is computed as $\mathsf{Code}_v[c] = \mathsf{prf}_{\mathsf{k2}}(h_v^{\mathsf{AES}_{\mathsf{k1}}(c)})$. Thus, the setup servers have to know (in a secret-shared form) the values $\mathsf{k1}$, h_v and $\mathsf{k2}$. Note that $h_v \leftarrow g^{x_v}$ for a random secret x_v. The setup servers securely forward the values $\mathsf{Code}_v[c]$ to the party who later sends the codes to the voters over prechannel, and also send $\mathsf{k1}$ to all potential voter PCs, $\mathsf{k2}$ to the messenger, h_v and x_v to the voter v's PC. The key $\mathsf{k1}$ must remain secret from the tallier, but has to be sent to the auditors (see the description of the tallier's operation later in this section).

Protocol: One Internet Vote Attempt. Voter v casts a vote for candidate $c \in \mathbb{Z}$, that is, inputs c to the PC by using some user interface, not necessarily as a number. The voter PC prepares two encryptions of the vote c, as $E_t \leftarrow \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathsf{AES}_{\mathsf{k1}}(c)}) = \mathsf{Enc}_{\mathsf{pkt}}(\mathsf{garble}_{\mathsf{k1}}(c))$ and $E_m \leftarrow \mathsf{Enc}_{\mathsf{pkm}}(h_v^{\mathsf{AES}_{\mathsf{k1}}(c)}) = \mathsf{Enc}_{\mathsf{pkm}}(\mathsf{garble}_{\mathsf{k1}}(c)^{x_v})$, where pkt is the tallier's public key and pkm is the messenger's public key. It then generates a non-interactive zero-knowledge proof of knowledge for

$$\pi = \mathrm{PK}(\mu_1, \mu_2, \rho_1, \rho_2 : E_t = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu_1}; \rho_1) \wedge E_m = \mathsf{Enc}_{\mathsf{pkm}}(h_v^{\mu_1}; \rho_2) \wedge h_v = g^{\mu_2}) \ , \tag{1}$$

that is, that the decryption of E_m (under the secret key of the messenger) is equal to the x_vth power of the decryption of E_t (under the secret key of the tallier), and in addition it knows the value x_v. The voter's PC generates a random number sid (session ID of this protocol instance).

It then signs $\sigma \leftarrow \mathsf{Sign}_v(sid, \mathbf{1}, v, E_t, E_m, \pi, time_v)$ (where $time_v$ is the time of voting according to the PC's clock, and $\mathbf{1}$ is included as a unique marker that separates this signature from signatures given in other purposes), and sends the result

$$(sid, \mathbf{1}, v, E_t, E_m, \pi, time_v, \sigma)$$

to the vote collector.

The vote collector verifies the signature and the NIZK proof. Upon success, the vote collector computes signature $\sigma' \leftarrow \mathsf{Sign}_{vc}(sid, \mathbf{2}, v, E_m, \pi', time_{vc})$ (where $time_{vc}$ is the time when the vote collector received this vote[4]) and sends

$$(sid, \mathbf{2}, v, E_t, E_m, \pi, time_v, \sigma, \pi', time_{vc}, \sigma')$$

to the messenger. The messenger verifies both signatures σ and σ', and the NIZK proof π. Upon success, he decrypts E_m, obtaining $\mathsf{garble}_{\mathsf{k1}}(c)^{x_v} = h_v^{\mathsf{AES}_{\mathsf{k1}}(c)}$. (Thus, the messenger should either not know $\mathsf{k1}$ or any of the values h_v.) The messenger, who has another secret key $\mathsf{k2}$, then computes $c' \leftarrow \mathsf{prf}_{\mathsf{k2}}(\mathsf{garble}(c)^{x_v})$, where prf is some fixed and public pseudorandom function family. The messenger also has a lexicographically ordered table of all values $H(\mathsf{prf}_{\mathsf{k2}}(\mathsf{garble}(z)^{x_v})) = H(\mathsf{Code}_v[z])$ for all possible candidates z. He checks that $H(c')$ belongs to this table (which takes $\Theta(\log \Gamma)$ non-cryptographic operations by using binary search, where Γ is again the number of candidates, or $\Theta(1)$ non-cryptographic operations by using a hash table). If it does not, then he complains to an arbiter/auditor that the voter PC cheated. This comparison step gives us some additional security, compared just to the case where the messenger sends out the SMS even when the candidate was invalid. It also saves an SMS message in the case the voter PC has encoded a bogus candidate — this can be important to avoid DDOS attacks. If such issues are not considered important, the comparison step can be omitted.

If messenger's verifications succeed, it sends $(c', time_m)$ back to the voter by the postchannel, where $time_m$ is messenger's current time. The voter checks that $c' = \mathsf{Code}_v[c]$ where c is a candidate he voted for at time moment approximately equal to $time_m$. If this verification is unsuccessful, the voter revotes by using either a computer, or goes later to the voting station for paper voting.

In parallel to message 2, the vote collector will store

$$(sid, v, E_t, E_m, \pi, \pi', time_v, time_{vc}, \sigma, \sigma') \ .$$

If a previous such entry for the same v existed, it will be overwritten.

Postcomputation. At the end of the Internet voting period, the vote collector will forward all stored entries, with the entire file (i.e., the collection of all stored entries) signed once by the vote collector, to the tallier. To eliminate the trust in vote collector, this step must be implemented by using a vote shuffle protocol, see e.g. [Gro03,GL07].)

[4] In actual e-voting, one can add extra steps in the case where $time_v$ and $time_{vc}$ differ too much. We will not consider this issue in this paper, and just assume that voter PC's clock is synchronized with that of the vote collector and the messenger.

Tallier's Operation. After receiving all (encrypted) votes from the vote collector, a semihonest tallier verifies two signatures (σ and σ') per voter, and then decrypts all values E_t, obtaining values $d = \mathsf{garble}_{\mathsf{k1}}(\cdot)$ for some candidates. To provide security against malicious tallier, one can use standard cryptographic techniques (mixnet followed by NIZK proofs of correct decryption). As mentioned before, this step is outside of the scope of the current paper. The mixnets and corresponding NIZK proofs are however standard in cryptographic literature. See [Gjø10] for discussion.

She will then tally the votes, and then send the tally $\{(\sharp votes, \mathsf{garble}_{\mathsf{k1}}(c))\}$ for garbled candidates to the auditors (human beings). The auditors have a (say in a printed) sorted list $\{(\mathsf{garble}_{\mathsf{k1}}(c), c)\}$ for all candidates c, and thus can associate the tallies with real candidates c, getting a list $\{(\sharp votes, c)\}$. This list is finally published.

The final step of the tallier can be implemented differently, depending on the desired trade-off between the security, the cost of implementation and the usability. In the first alternative, there is another server (independent of the tallier), who has access to the table $\{(\mathsf{garble}_{\mathsf{k1}}(c), c)\}$. This server receives the list $\{(\sharp votes, \mathsf{garble}_{\mathsf{k1}}(c))\}$ from the tallier, and then outputs the list $\{(\sharp votes, c)\}$. That step requires very little computational power if we trust this server — otherwise, the server might need to output a NIZK proof of correct operation. This alternative has better usability (since human beings do not have to associate the lists), essentially the same security (since here the tallier does not know the actual candidates), but one more server compared to the solution of the previous paragraph. In the second alternative, the tallier actually has access to the list $\{(\mathsf{garble}_{\mathsf{k1}}(c), c)\}$ and thus can compute the tally $\{(\sharp votes, c)\}$ by herself. This alternative has better usability (since human beings do not have to associate the lists), the same cost but somewhat worse security (since if the tallier is not honest, she can ignore votes cast for candidates she does not like) compared to the solution in the previous paragraph. That problem can be alleviated by raising computational cost by letting the server to present a (relatively costly) NIZK proof of correct operation.

4.1 Non-Interactive Zero-Knowledge Proof

We need a NIZK proof of knowledge for $\mathrm{PK}(\mu_1, \mu_2, \rho_1, \rho_2 : E_t = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu_1}; \rho_1) \wedge E_m = \mathsf{Enc}_{\mathsf{pkm}}(h_v^{\mu_1}; \rho_2) \wedge h_v = g^{\mu_2})$. The interactive version of this NIZK (that is, a Σ-protocol) is as follows (assuming that g is a group generator, $k \geq 80$ is the security parameter, and q is the order of the group):

1. The prover generates $m_1, m_2, r_1, r_2 \leftarrow \mathbb{Z}_q$, and sets $\mathfrak{i}_1 \leftarrow \mathsf{Enc}_{\mathsf{pkt}}(g^{m_1}; r_1)$, $\mathfrak{i}_2 \leftarrow \mathsf{Enc}_{\mathsf{pkm}}(h_v^{m_1}; r_2)$, $\mathfrak{i}_3 \leftarrow g^{m_2}$. He sends $(\mathfrak{i}_1, \mathfrak{i}_2, \mathfrak{i}_3)$ to the verifier.
2. The verifier sends $\mathfrak{c} \leftarrow \{0,1\}^k$ to the prover.
3. The prover sends $\mathfrak{r}_1 \leftarrow m_1 + \mathfrak{c} \cdot \mu_1$, $\mathfrak{r}_2 \leftarrow r_1 + \mathfrak{c} \cdot \rho_1$, $\mathfrak{r}_3 \leftarrow r_2 + \mathfrak{c} \cdot \rho_2$, $\mathfrak{r}_4 \leftarrow m_2 + \mathfrak{c} \cdot \mu_2$ to the verifier.
4. The verifier accepts iff $\mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}_1}; \mathfrak{r}_2) = \mathfrak{i}_1 \cdot E_t^{\mathfrak{c}}$, $\mathsf{Enc}_{\mathsf{pkm}}(h_v^{\mathfrak{r}_1}; \mathfrak{r}_3) = \mathfrak{i}_2 \cdot E_m^{\mathfrak{c}}$ and $g^{\mathfrak{r}_4} = \mathfrak{i}_3 \cdot h_v^{\mathfrak{c}}$.

The proof that this Σ-protocol satisfies the properties of correctness, special soundness and SHVZK is standard.

Lemma 1. *The above Σ-protocol is correct and satisfies the properties of special soundness and special honest-verifier zero-knowledge.*

Proof. CORRECTNESS. Clearly, $\mathfrak{i}_1 \cdot E_t^{\mathfrak{c}} = \mathsf{Enc}_{\mathsf{pkt}}(g^{m_1}; r_1) \cdot \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{c} \cdot \mu_1}; \mathfrak{c} \cdot \rho_1) = \mathsf{Enc}_{\mathsf{pkt}}(g^{m_1 + \mathfrak{c} \cdot \mu_1}; r_1 + \mathfrak{c} \cdot \rho_1) = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}_1}; \mathfrak{r}_2)$, $\mathfrak{i}_2 \cdot E_m^{\mathfrak{c}} = \mathsf{Enc}_{\mathsf{pkm}}(h_v^{m_1}; r_2) \cdot \mathsf{Enc}_{\mathsf{pkm}}(h_v^{\mathfrak{c} \cdot \mu_1}; \mathfrak{c} \cdot \rho_2) = \mathsf{Enc}_{\mathsf{pkm}}(h_v^{m_1 + \mathfrak{c} \cdot \mu_1}; r_2 + \mathfrak{c} \cdot \rho_2) = \mathsf{Enc}_{\mathsf{pkm}}(h_v^{\mathfrak{r}_1}; \mathfrak{r}_3)$ and $\mathfrak{i}_3 \cdot h_v^{\mathfrak{c}} = g^{m_2 + \mathfrak{c} \cdot \mu_2} = g^{\mathfrak{r}_4}$.

SPECIAL SOUNDNESS. First, assume that $\mathfrak{i}_1 \cdot E_t^{\mathfrak{c}} = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}_1}; \mathfrak{r}_2)$ and $\mathfrak{i}_1 \cdot E_t^{\mathfrak{c}'} = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}'_1}; \mathfrak{r}'_2)$, where $\mathfrak{c} \neq \mathfrak{c}'$. Then $E_t^{\mathfrak{c} - \mathfrak{c}'} = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}_1 - \mathfrak{r}'_1}; \mathfrak{r}_2 - \mathfrak{r}'_2)$, and thus $E_t = \mathsf{Enc}_{\mathsf{pkt}}(g^{(\mathfrak{r}_1 - \mathfrak{r}'_1)/(\mathfrak{c} - \mathfrak{c}')}; (\mathfrak{r}_2 - \mathfrak{r}'_2)/(\mathfrak{c} - \mathfrak{c}'))$. Thus, given two accepting views, one can find $\mu_1 = (\mathfrak{r}_1 - \mathfrak{r}'_1)/(\mathfrak{c} - \mathfrak{c}')$, and $\rho_1 = (\mathfrak{r}_2 - \mathfrak{r}'_2)/(\mathfrak{c} - \mathfrak{c}')$, such that $E_t = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu_1}; \rho_1)$.

Second, assume that $\mathfrak{i}_2 \cdot E_m^{\mathfrak{c}} = \mathsf{Enc}_{\mathsf{pkm}}(h_v^{\mathfrak{r}_1}; \mathfrak{r}_3)$ and $\mathfrak{i}_2 \cdot E_m^{\mathfrak{c}'} = \mathsf{Enc}_{\mathsf{pkm}}(h_v^{\mathfrak{r}'_1}; \mathfrak{r}'_3)$. Then $E_m^{\mathfrak{c} - \mathfrak{c}'} = \mathsf{Enc}_{\mathsf{pkm}}(h_v^{\mathfrak{r}_1 - \mathfrak{r}'_1}; \mathfrak{r}_3 - \mathfrak{r}'_3)$, or $E_m = \mathsf{Enc}_{\mathsf{pkm}}(h_v^{\mu_1}; \rho_2)$ for $\mu_1 = (\mathfrak{r}_1 - \mathfrak{r}'_1)/(\mathfrak{c} - \mathfrak{c}')$ (as before) and $\rho_2 = (\mathfrak{r}_3 - \mathfrak{r}'_3)/(\mathfrak{c} - \mathfrak{c}')$.

Third, assume that $\mathfrak{i}_3 \cdot h_v^{\mathfrak{c}} = g^{\mathfrak{r}_4}$ and $\mathfrak{i}_3 \cdot h_v^{\mathfrak{c}'} = g^{\mathfrak{r}'_4}$ for $\mathfrak{c} \neq \mathfrak{c}'$. Then $h_v^{\mathfrak{c} - \mathfrak{c}'} = g^{\mathfrak{r}_4 - \mathfrak{r}'_4}$, or $h_v = g^{\mu_2}$ for $\mu_2 = (\mathfrak{r}_4 - \mathfrak{r}'_4)/(\mathfrak{c} - \mathfrak{c}')$.

SPECIAL HONEST-VERIFIER ZERO-KNOWLEDGE. The simulator can first choose all values $\mathfrak{c}, \mathfrak{r}_1, \ldots, \mathfrak{r}_4$ randomly, and then choose $\mathfrak{i}_1$, $\mathfrak{i}_2$ and $\mathfrak{i}_3$ such that they satisfy the verification. For example, $\mathfrak{i}_3 \leftarrow h_v^{-\mathfrak{c}} \cdot g^{\mathfrak{r}_4}$. Clearly, the view created by the simulator is indistinguishable from the view in an actual run, given that the verifier is honest. □

Thus, one can use the Fiat-Shamir heuristic to transform it to a NIZK proof of knowledge. If Elgamal is used, the prover (that is, the voter PC) needs to perform 6 exponentiations, and the verifier (the vote collector) has to perform 9 exponentiations. In the actual code-verification voting protocol of this section, $m = \mathsf{garble}_{\mathsf{k1}}(c)$.

4.2 Implementation

We consider the implementation of Elgamal by conservatively using 283-bit elliptic curves over binary fields; this corresponds to the security provided by 141-bit long symmetric keys. According to `http://www.shamus.ie/index.php?page=Elliptic-Curve-point-multiplication`, an elliptic point multiplication (i.e., exponentiation in Elgamal) takes on average 3.56 milliseconds. We note that in Estonian Internet voting in 2009, with slightly more than 100 000 total Internet voters, 4 500 votes were cast during the peak hour. See `http://www.vvk.ee/public/pics/EP09kokkuakt.jpg` for a distribution of the number of votes per hour during that Internet voting period.

The voter PC executes 6 exponentiations to compute E_t and E_m, 6 exponentiations to compute π, and 1 signing. Thus in total it executes 12 exponentiations (≈ 42.72 milliseconds), and signs 1 signature.

The vote collector executes a signature verification, 9 exponentiations to verify π, and signs one message. For exponentiations alone, he spends $7 \cdot 3.56 \approx 32.04$ milliseconds, and thus can handle—ignoring the time that is required to sign and verify signatures—more than 112 000 votes per hour.

The messenger verifies two signatures, verifies one NIZK proof (9 exponentiations), and performs 1 exponentiation to decrypt E_m, computes prf (insignificant) and then $\log_2 \Gamma$ non-cryptographic operations (insignificant). Thus, when we only count the $9 + 1 = 10$ exponentiations, the messenger can handle more than 101 000 votes a hour, and its throughput is in practice restricted by the ability of sending messages on the postchannel. In the case the latter throughput is not sufficiently high, one can run several messengers in parallel.

After the Internet voting period ends, in the simplest case the vote collector will compute only one more signature. The tallier will perform 1 exponentiation (decryption) and one signature verification per voter, and then (say) sorts the values. This is very close to the optimal (1 million exponentiations can be done in about one hour), but in the case of very large scale elections, one might still need a Hardware Security Module. However, as said, the vote collector ought to perform cryptographic shuffle, and also the tallier's operation should be secured by using mixnets and a suitable NIZK proof of correctness. Since this step is done after the Internet voting period ends, the exact computational efficiency of this step is not such a big concern.

4.3 Security Guarantees

Security against Malicious Voter PC: In the code-verification voting protocol of this section, the voter PC sees the candidate name c, and thus the protocol does not guarantee privacy against malicious PC. It is unclear how to avoid this without changing the voting user interface dramatically (e.g., by requiring the voter to enter long random numbers, as in code voting). On the other hand, the voter PC can disrupt the protocol by not forwarding the voter's candidate (which will be detected by the voter due to the lack of a returning message from the postchannel), or sending inconsistent information (which will be detected by the NIZK proofs and/or wrong return code).

Even if the voter PC could gain both the read and write access to the postchannel between the messenger and all voters (but without corrupting the messenger), creating a correct integrity check code requires it to compute $\mathsf{prf}_{\mathsf{k2}}$ on a previously unseen value $h_v^{\mathsf{AES}_{\mathsf{k1}}(c)}$, which is intractable, assuming that prf is a secure pseudorandom function family and $\mathsf{k2}$ is only known to the messenger. The only bad case is if the voter PC has both the read and the write access to the postchannel, and the voter votes according to pattern A-B-A (i.e., after voting for candidate A, the same voter votes for candidate B and then again for candidate A). Now, the voter PC could send B to the vote collector, while sending the integrity check code for A over the postchannel. (See [HLV10] for discussion.) To avoid this attack, it is required that the postchannel is independent of the voter PCs (voter PCs have neither read or write access to the postchannel) — e.g., SMS. Thus we have informally proven the next lemma.

Lemma 2. *The code-verification protocol of this section guarantees correctness against malicious voter PCs, assuming that* prf *is a secure pseudorandom function family, the used signature scheme is secure, H is a random oracle, the postchannel is independent from the voter PCs and that the voter verifies the messages obtained from the postchannel.*

Note that verification by voters is necessary in any case, and is not too difficult for voters. Because the biggest danger in the case of Internet voting is massively distributed malware that addresses a large number of voters,

then to guarantee the correctness of Internet voting with "large probability", it is sufficient that a sufficiently large fraction of voters verifies the correctness of codes. We omit further discussion.

Security against Malicious Vote Collector: One can easily see that the next lemma holds.

Lemma 3. *Privacy against a malicious vote collector is guaranteed since Elgamal is semantically secure. The correctness of the vote collector's first operation (sending of an encrypted check code to the messenger) is guaranteed trivially, given that the used signature scheme is secure, H is a random oracle, and that the messenger performs all required verifications.*

The new voting protocol of this paper does however *not* guarantee that the vote collector forwards correct ciphertexts to the tallier. Without revoting, this could be achieved by using a standard cryptographic shuffle protocol [Gro03,GL07]. With revoting, this proof will be much very complex, since the vote collector has to somehow prove that the shuffled vote was the last one from every e-voter. To prevent coercion, it is unreasonable to assume anyhow that an Internet voting protocol provides universal verifiability. Thus, we can assume that there is an internal (incorruptible) auditor (say, the tallier itself), who can verify the correctness of the shuffle, but without knowing whether the vote collector included the last e-vote from every voter. On the other hand, one should allow the voters to later go physically to a (paper-)voting station, and verify whether in their own concrete case *their* last e-vote was used.

Security against Malicious Messenger: Malicious messenger can obviously always refuse sending codes on postchannel, or send codes when not asked to. In such cases the voter will be alerted, but will not be completely sure of the origin of the problem. Sending correct codes corresponding to a wrong candidate (say, for c) is only possible when the messenger has already seen the same voter voting for exactly the same c before, e.g., in an A-B-A pattern as before. (Note that the messenger does not know c, he just knows that the vote is for the same c as before.) Otherwise, the messenger has to be able to compute, from value c^* the value $h_v^{\mathsf{AES}_{\mathsf{k1}}(c^*)}$, which is impossible since the messenger does not know $\mathsf{k1}$, and AES is a pseudorandom function family. In either case, the voter sees that the code is incorrect. Thus, if verification of a integrity check code is not successful, then either the voter PC was malicious, or the messenger was malicious. If the messenger is malicious, then the voter does not see the code at all, a completely random code, or a code for incorrect candidate. Assuming that AES is a pseudorandom function family and the messenger does not know $\mathsf{k1}$, the latter can only happen during revoting.

Security against Malicious Tallier: The tallier part can be secured by using standard cryptographic techniques like mixnets. Moreover, there exist relatively efficient NIZK proofs of correct decryption. (We emphasize that specifying the shuffle protocol or the operation of tallier is not the subject of the current paper.) The fact that the tallier does not know $\mathsf{k2}$, and thus only sees values $\mathsf{garble}_{\mathsf{k1}}(c)$, provides some additional protection in the case the shuffle fails or the tallier "fails" on nonpreferred tallies.

Security against Coalition of Malicious Vote Collector *And* Messenger: The coalition of a malicious vote collector and a malicious messenger sees both the values h_v and $y = \mathsf{garble}_{\mathsf{k1}}(c)^{x_v} = h_v^{\mathsf{AES}_{\mathsf{k1}}(c)}$, as computed by a voter PC. Since the coalition can also recover $\mathsf{k1}$ by collaborating with any of malicious voter PCs, they can test, for every candidate c^*, whether $y = h_v^{\mathsf{AES}_{\mathsf{k1}}(c^*)}$, and thus completely breach the privacy of voters even in the case the voter PCs are honest.

5 Second New Code-Verification Voting Protocol

As mentioned at the end of Sect. 4.3, in the code-verification voting protocol of Sect. 4, coalition of a malicious vote collector and a malicious messenger can completely breach voter privacy. To avoid this, we modify the protocol of Sect. 4 in one crucial aspect: instead of publishing the values $h_v = g^{x_v}$, we publish the Pedersen commitments $C_v \leftarrow g^{x_v} h^{r_v}$ to x_v, where $h \leftarrow \mathbb{G}$ is a new random public key, and a random coin r_v. This does not change the overall protocol much. In what follows, we will give a quick overview of the changes.

In the setup phase, all parties have additional access to the public key h (used by the commitment scheme, see Sect. 2), while no party knows the corresponding secret key. Instead of h_v, the values $C_v \leftarrow g^{x_v} h^{r_v}$ are published. The setup servers must also know the values h_v (even if in a secret-shared form). In the actual protocol, we basically only need to replace the NIZK proof of knowledge. Instead of a NIZK proof of knowledge for Eq. (1), we need here a NIZK proof of knowledge for

$$\pi' = \mathrm{PK}(\mu_1, \mu_2, \rho_1, \rho_2, \rho_3 : E_t = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu_1}; \rho_1) \wedge C_v = \mathsf{Com}_h(g^{\mu_2}; \rho_3) \wedge E_m = \mathsf{Enc}_{\mathsf{pkm}}(g^{\mu_1 \mu_2}; \rho_2)) \ , \tag{2}$$

this proof of knowledge will be constructed in Sect. 5.1. There are no further changes between this protocol, and the code-verification protocol of Sect. 4. (Thus, also this protocol follows Fig. 1.)

5.1 Second NIZK Protocol

We need a NIZK proof of knowledge for Eq. (2), in the case of the Elgamal cryptosystem and the Pedersen commitment scheme. Let $E_t = (e_{t1}, e_{t2})$, where supposedly $e_{t1} = g^{\mu_1}\mathsf{pkt}^{\rho_1}$ and $e_{t2} = g^{\rho_1}$. Let $E_m = (e_{m1}, e_{m2})$, where supposedly $e_{m1} = g^{\mu_1\mu_2}\mathsf{pkm}^{\rho_2}$ and $e_{m_2} = g^{\rho_2}$. Thus, we need that $E_m = (g^{\mu_1\mu_2}\mathsf{pkm}^{\rho_2}, g^{\rho_2}) = (e_{t1}^{\mu_2}\mathsf{pkm}^{\rho_2}\mathsf{pkt}^{-\rho_1\mu_2}, e_{t2}^{\mu_2}g^{\rho_2-\mu_2\rho_1}) = E_t^{\mu_2}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;-\rho_1\mu_2)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;\rho_2)$. To construct a truly efficient protocol, we note that for the security of the voting protocol of this section, it is not really necessary that the penultimate randomness in this expression is equal to $-\rho_1\mu_2$. So instead of the original proof of knowledge for Eq. (2), we will next construct a Σ-protocol for $\mathrm{PK}(\mu_1,\mu_2,\rho_1,\rho_2,\rho_3,\rho_4 : E_t = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu_1};\rho_1)\wedge C_v = \mathsf{Com}_h(g^{\mu_2};\rho_2)\wedge E_m = E_t^{\mu_2}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;\rho_3)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;\rho_4))$.

The corresponding Σ-protocol is as follows (assuming that g is a group generator, $k \geq 80$ is the security parameter, and q is the order of the group):

1. The prover generates $m_1, m_2, r_1, r_2, r_3, r_4 \leftarrow \mathbb{Z}_q$, and sets $\mathfrak{i}_1 \leftarrow \mathsf{Enc}_{\mathsf{pkt}}(g^{m_1}; r_1)$, $\mathfrak{i}_2 \leftarrow \mathsf{Com}_h(g^{m_2}; r_2)$, and $\mathfrak{i}_3 \leftarrow E_t^{m_2}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;r_3)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;r_4)$. He sends $(\mathfrak{i}_1,\mathfrak{i}_2,\mathfrak{i}_3)$ to the verifier.
2. The verifier sends $\mathfrak{c} \leftarrow \{0,1\}^k$ to the prover.
3. The prover sends $\mathfrak{r}_1 \leftarrow m_1 + \mathfrak{c}\cdot\mu_1$, $\mathfrak{r}_2 \leftarrow r_1 + \mathfrak{c}\cdot\rho_1$, $\mathfrak{r}_3 \leftarrow m_2 + \mathfrak{c}\cdot\mu_2$, $\mathfrak{r}_4 \leftarrow r_2 + \mathfrak{c}\cdot\rho_2$, $\mathfrak{r}_5 \leftarrow r_3 + \mathfrak{c}\cdot\rho_3$, $\mathfrak{r}_6 \leftarrow r_4 + \mathfrak{c}\cdot\rho_4$ to the verifier.
4. The verifier accepts iff $\mathfrak{i}_1\cdot E_t^{\mathfrak{c}} = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}_1};\mathfrak{r}_2)$, $\mathfrak{i}_2\cdot C_v^{\mathfrak{c}} = \mathsf{Com}_h(g^{\mathfrak{r}_3};\mathfrak{r}_4)$, and $\mathfrak{i}_3\cdot E_m^{\mathfrak{c}} = E_t^{\mathfrak{r}_3}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;\mathfrak{r}_5)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;\mathfrak{r}_6)$.

Lemma 4. *The above Σ-protocol is correct and satisfies the properties of special soundness and special honest-verifier zero-knowledge.*

Proof. CORRECTNESS. Clearly, $\mathfrak{i}_1\cdot E_t^{\mathfrak{c}} = \mathsf{Enc}_{\mathsf{pkt}}(g^{m_1};r_1)\cdot\mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{c}\cdot\mu_1};\mathfrak{c}\cdot\rho_1) = \mathsf{Enc}_{\mathsf{pkt}}(g^{m_1+\mathfrak{c}\cdot\mu_1};r_1+\mathfrak{c}\cdot\rho_1) = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}_1};\mathfrak{r}_2)$, $\mathfrak{i}_2\cdot C_v^{\mathfrak{c}} = \mathsf{Com}_h(g^{m_2};r_3)\cdot\mathsf{Com}_h(g^{\mathfrak{c}\cdot\mu_2};\mathfrak{c}\cdot\rho_3) = \mathsf{Com}_h(g^{\mathfrak{r}_3};\mathfrak{r}_4)$, and

$$\begin{aligned}\mathfrak{i}_3\cdot E_m^{\mathfrak{c}} &= E_t^{m_2}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;r_3)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;r_4)\cdot E_t^{\mathfrak{c}\cdot\mu_2}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;\mathfrak{c}\cdot\rho_3)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;\mathfrak{c}\cdot\rho_4)\\ &= E_t^{m_2+\mathfrak{c}\cdot\mu_2}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;r_3+\mathfrak{c}\cdot\rho_3)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;r_4+\mathfrak{c}\cdot\rho_4)\\ &= E_t^{\mathfrak{r}_3}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;\mathfrak{r}_5)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;\mathfrak{r}_6)\ .\end{aligned}$$

SPECIAL SOUNDNESS. First, assume that $\mathfrak{i}_1\cdot E_t^{\mathfrak{c}} = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}_1};\mathfrak{r}_2)$ and $\mathfrak{i}_1\cdot E_t^{\mathfrak{c}'} = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}'_1};\mathfrak{r}'_2)$, where $\mathfrak{c}\neq\mathfrak{c}'$. Then $E_t^{\mathfrak{c}-\mathfrak{c}'} = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mathfrak{r}_1-\mathfrak{r}'_1};\mathfrak{r}_2-\mathfrak{r}'_2)$, and thus $E_t = \mathsf{Enc}_{\mathsf{pkt}}(g^{(\mathfrak{r}_1-\mathfrak{r}'_1)/(\mathfrak{c}-\mathfrak{c}')};(\mathfrak{r}_2-\mathfrak{r}'_2)/(\mathfrak{c}-\mathfrak{c}'))$. Thus, given two accepting views, one can find $\mu_1 = (\mathfrak{r}_1-\mathfrak{r}'_1)/(\mathfrak{c}-\mathfrak{c}')$, and $\rho_1 = (\mathfrak{r}_2-\mathfrak{r}'_2)/(\mathfrak{c}-\mathfrak{c}')$, such that $E_t = \mathsf{Enc}_{\mathsf{pkt}}(g^{\mu_1};\rho_1)$.

Second, assume that $\mathfrak{i}_2\cdot C_v^{\mathfrak{c}} = \mathsf{Com}_h(g^{\mathfrak{r}_3};\mathfrak{r}_4)$ and $\mathfrak{i}_2\cdot C_v^{\mathfrak{c}'} = \mathsf{Com}_h(g^{\mathfrak{r}'_3};\mathfrak{r}'_4)$. Then $C_v^{\mathfrak{c}-\mathfrak{c}'} = \mathsf{Com}_h(g^{\mathfrak{r}_3-\mathfrak{r}'_3};\mathfrak{r}_4-\mathfrak{r}_4)$, or $C_v = \mathsf{Com}_h(g^{\mu_2};\rho_3)$ for $\mu_2 = (\mathfrak{r}_3-\mathfrak{r}'_3)/(\mathfrak{c}-\mathfrak{c}')$ and $\rho_3 = (\mathfrak{r}_4-\mathfrak{r}'_4)/(\mathfrak{c}-\mathfrak{c}')$.

Third, assume that $\mathfrak{i}_3\cdot E_m^{\mathfrak{c}} = E_t^{\mathfrak{r}_3}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;\mathfrak{r}_5)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;\mathfrak{r}_6)$ and $\mathfrak{i}_2\cdot E_m^{\mathfrak{c}'} = E_t^{\mathfrak{r}'_3}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;\mathfrak{r}'_5)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;\mathfrak{r}'_6)$. Then $E_m^{\mathfrak{c}-\mathfrak{c}'} = E_t^{\mathfrak{r}_3-\mathfrak{r}'_3}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;\mathfrak{r}_5-\mathfrak{r}'_5)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;\mathfrak{r}_6-\mathfrak{r}'_6)$, or $E_m = E_t^{\mu_2}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;(\mathfrak{r}_5-\mathfrak{r}'_5)/(\mathfrak{c}-\mathfrak{c}'))\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;(\mathfrak{r}_6-\mathfrak{r}'_6)/(\mathfrak{c}-\mathfrak{c}'))$. Define $\rho_3 = (\mathfrak{r}_5-\mathfrak{r}'_5)/(\mathfrak{c}-\mathfrak{c}')$ and $\rho_4 = (\mathfrak{r}_6-\mathfrak{r}'_6)/(\mathfrak{c}-\mathfrak{c}')$, then $E_m = E_t^{\mu_2}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;\rho_3)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;\rho_4)$ as needed.

SPECIAL HONEST-VERIFIER ZERO-KNOWLEDGE. The simulator can first choose all values $\mathfrak{c}, \mathfrak{r}_1, \ldots, \mathfrak{r}_6$ randomly, and then choose $\mathfrak{i}_1$, $\mathfrak{i}_2$ and $\mathfrak{i}_3$ such that they satisfy the verification. For example, $\mathfrak{i}_3 \leftarrow E_m^{-\mathfrak{c}}\cdot E_t^{\mathfrak{r}_3}\cdot\mathsf{Enc}_{\mathsf{pkt}}(0;\mathfrak{r}_5)\cdot\mathsf{Enc}_{\mathsf{pkm}}(0;\mathfrak{r}_6)$. Clearly, the view created by the simulator is indistinguishable from the view in an actual run, given that the verifier is honest. □

Note that in this Σ-protocol, $\mathfrak{i}_3 = E_t^{m_2}\cdot(\mathsf{pkt}^{r_3}\mathsf{pkm}^{r_4}, g^{r_3+r_4})$. Thus, the prover has to perform 10 exponentiations. The verifier has to perform 17 exponentiations.

5.2 Implementation of Second Protocol

From the perspective of implementation, the only substantial difference between the two new code-verification protocols is that in the protocol of Sect. 5, one requires a different NIZK protocol. Thus, in the protocol of the current section, the voter PC executes 16 exponentiations, and signs 1 message. The vote collector executes a signature verification, 17 exponentiations to verify π, and signs one message. For exponentiations alone, he spends $17{\cdot}3.56 \approx 60.52$ milliseconds, and thus can handle—taking into account only time, required for exponentiations—about 59 000 votes per hour. The messenger verifies two signatures, and performs 18 exponentiations, being thus able to handle about 56 000 messages per hour, when only counting the exponentiations.

5.3 Security Guarantees of Second Protocol

Most of the security analysis of Sect. 4.3 apply applies to the protocol of the current section, with one crucial difference. Namely, in the case of a coalition between malicious vote collector and malicious messenger, the coalition does not know the value $h_v = g^{x_v}$, and thus cannot verify, whether the value y is equal to $h_v^{\mathsf{AES}_{\mathsf{k1}}(c)}$ for any fixed value c or not. Thus, in this protocol, coalition of online servers cannot breach voter privacy.

6 Application to Code Voting

The code-verification voting setting is similar to that of code voting [Cha01]. The main difference is that in code-verification voting, the voters just enter c to the PCs, by using any favorite user interface, while in code voting, the voters enter long random numbers. This makes privacy against malicious voter PCs achievable in the code-voting setting but not in the code-verification setting. However, it was felt in Norway that code-voting-like approaches were inapplicable due to usability concerns. Another important difference is that in code-verification voting, at least as it will be implemented in Norway, support for revoting is mandatory. Nevertheless, the two new code-verification voting protocols are also applicable in the code-voting setting. In a nutshell, in the case of code voting, a similar composition of two pseudorandom functions can be defined to map voter's (pseudorandom) input code to the (pseudorandom) integrity check code. For example, one can assume that the input code of voter v is equal to $g^{-f_v(c)}$, where f_v is some voter-specific function, unknown to the voter PC. We omit further discussion.

Acknowledgments. The author was supported by Estonian Science Foundation, grant #8058, and European Union through the European Regional Development Fund.

References

BR93. Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In Victoria Ashby, editor, *ACM CCS 1993*, pages 62–73, Fairfax, Virginia, 3–5 November 1993. ACM Press.

Bul10. Christian Bull. Open Source e-Voting - the Norwegian approach. In *Swiss E-Voting Workshop 2010*, University of Fribourg, Switzerland, September 6, 2010. Invited talk. Slides available from `http://www.e-voting-cc.ch/index.php/en/workshop10`, as of December 2010.

CDS94. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In Yvo G. Desmedt, editor, *CRYPTO 1994*, volume 839 of *LNCS*, pages 174–187, Santa Barbara, USA, August 21–25 1994. Springer-Verlag.

CGH98. Ran Canetti, Oded Goldreich, and Shai Halevi. The Random Oracle Methodology, Revisited. In *STOC 1998*, pages 209–218, New York, May 23–26, 1998.

Cha01. David Chaum. SureVote: Technical Overview. In *WOTE 2001*, 2001. Available from `http://www.vote.caltech.edu/wote01/pdfs/surevote.pdf`, as of August, 2010.

Elg85. Taher Elgamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

FS86. Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *CRYPTO 1986*, volume 263 of *LNCS*, pages 186–194, Santa Barbara, California, USA, 11–15 August 1986. Springer-Verlag, 1987.

Gjø10. Kristian Gjøsteen. Analysis of an Internet Voting Protocol. Technical Report 2010/380, International Association for Cryptologic Research, July 5, 2010. Available at http://eprint.iacr.org/2010/380.

GL07. Jens Groth and Steve Lu. Verifiable Shuffle of Large Size Ciphertexts. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 377–392, Beijing, China, April 16–20, 2007. Springer-Verlag.

Gro03. Jens Groth. A Verifiable Secret Shuffle of Homomorphic Encryptions. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 145–160, Miami, Florida, USA, January6–8, 2003. Springer-Verlag.

HLV10. Sven Heiberg, Helger Lipmaa, and Filip Van Laenen. On E-Vote Integrity in the Case of Malicious Voter Computers. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS 2010*, volume 6345 of *LNCS*, pages 373–388, Athens, Greece, September 20–22, 2010. Springer-Verlag.

Ped91. Torben P. Pedersen. Non-Interactive And Information-Theoretic Secure Verifiable Secret Sharing. In Joan Feigenbaum, editor, *CRYPTO 1991*, volume 576 of *LNCS*, pages 129–140, Santa Barbara, California, USA, August 11–15, 1991. Springer-Verlag, 1992.

A Description of Protocol from [HLV10]

The code-verification voting protocol by Heiberg, Lipmaa and Van Laenen [HLV10] has computational complexity (by both the vote collector and the messenger) that is linear in the number Γ of the candidates. More precisely, in this protocol, the voter PC generates two ciphertexts E_m and E_t that encrypt the candidate number c^*, by using respectively the public keys of the messenger and the tallier. It sends the ciphertexts, together with a non-interactive zero-knowledge (NIZK) proof of knowledge of correctness (i.e., that E_m and E_t encrypt the same valid candidate number), to the vote collector. The vote collector verifies the NIZK proof.

Upon success, the vote collector transforms E_m to an encryption of $\mathsf{Code}_v[c^*]$ under the messenger's public key, by using proxy oblivious transfer. The vote collector sends the result to the messenger. The messenger "decrypts" the result, obtains $\mathsf{Code}_v[c^*]$, and sends it to the voter (not to the voter PC!) by using the postchannel. In parallel, the vote collector retains E_t. Finally, the voters can revote, and then the same protocol is executed. In particular, for every voter, the vote collector keeps the last version of E_t, and after the end of the Internet voting period, forwards them to the tallier.

The computational complexity of a single e-vote attempt in the Heiberg-Lipmaa-Van Laenen Internet voting protocol is dominated by that of the proxy oblivious transfer: since the codes are supposed to be random, there are no obvious ways of making the computational complexity of this step smaller than $\Theta(\Gamma)$, where Γ is the number of candidates. (See [HLV10] for a discussion.) This is true even in the case when we only aim to achieve security against semihonest voting servers (and a malicious voter PC); achieving security against malicious voting servers will be even more costly.