Return Code Schemes for Electronic Voting Systems

Shahram Khazaei[1] and Douglas Wikström[2](✉)

[1] Sharif University of Technology, Tehran, Iran
shahram.khazaei@sharif.ir

[2] KTH Royal Institute of Technology, Stockholm, Sweden
dog@kth.se

Abstract. We describe several return code schemes for secure vote submission in electronic voting systems. We consider a unified treatment where a return code is generated as a multiparty computation of a secure MAC tag applied on an encrypted message submitted by a voter. Our proposals enjoy a great level of flexibility with respect to various usability, security, and performance tradeoffs.

1 Introduction

Electronic voting systems have the potential of achieving *end-to-end verifiability*. This is obtained through different verification mechanisms throughout all the stages of the entire voting process, known as *cast-as-intended*, *recorded-as-cast* and *counted-as-recorded* [2].

Cast-as-intended verification assures each individual voter that his vote has been cast according to his intention. Mechanisms that ensures the cast votes have been correctly received and stored are called recorded-as-cast. Counted-as-recorded verification allows any third party observer, such as voters and auditors, to verify that the result of the tally corresponds to the received votes.

In some electronic voting systems, the voter casts his encrypted vote using some voting device which might either belong to the election authorities, e.g., a computer with a touch screen in a furnished voting booth, or to the voter himself, when the risk of coercion is limited. A malicious voting device (due to malware or hostile hardware) may change a voter's intended choice. Cast-as-intended verifiability detects such attacks on voting devices.

There are two basic approaches to verifying that a vote was cast as intended: (a) verify that the right choice was encrypted and that the ciphertext was recorded, and (b) verify that the ciphertext decrypts to the intended choice.

The most straightforward solution to the first problem is to simply perform the encryption independently on a different device and compare the results as is done in Estonia [17].

Another approach is continuous blackbox testing as proposed by Benaloh [5], and adopted in Helios [1], Wombat [23], and VoteBox [25]. Here the device provides a ciphertext and the voter can choose to either use it, or to challenge

R. Krimmer et al. (Eds.): E-Vote-ID 2017, LNCS 10615, pp. 198–209, 2017.
DOI: 10.1007/978-3-319-68687-5_12

the device to prove that it was formed correctly. Note that the latter choice amounts to normal blackbox testing. The key insight of Benaloh is that we can not only interlace testing with normal use (which is often done in safety critical software), we can let the *voters control when and where testing takes place* to provide maximum individual assurance. The importance of this observation lies in part in that concerned voters can run more tests, so the testing seamlessly aligns with the level of assurance needed by individual voters.

Depending on trust assumptions, i.e., who performs the verification of outputs that have been challenged (an electronic device, jointly with a human, or third parties), Benaloh's approach is more or less practical and gives different types of assurances.

The reader may object that the encryption device must commit to its output before the choice to verify it is made or not, but this is no different from many other types of testing done on software. Benaloh's approach is often confused with cut-and-choose zero knowledge proofs due to the choice given to the voter between two choices, but is better described as a *have-or-eat* protocol: you can't have your cake and eat it too (to see that the cake is not poisonous).

So called *return codes* have received a considerable amount of attention particularly due to their usability properties. We refer the reader to [3,4,12,14,18,19,22] for several proposals. This approach has been used in nation-wide elections in Norway [12,13,22] and Switzerland [10].

The idea of return codes is that each possible choice of the voter is associated with a random code that is returned upon submission of the encrypted vote as an acknowledgement that the ciphertext received encrypts the intended choice. To ensure privacy, the random codes of different voters are chosen independently. In other words, individual codes reveal nothing about the vote itself.

Note that at its core this is classical code book encryption, i.e., the parties receiving a vote in encrypted form send back the vote in encoded form to the voter. However, we only use the return codes for acknowledgement, so there is no need for the codes to uniquely identify the choices. Thus, for each voter we need a fairly regular random map from the set of choices to a set of codes, i.e., a message authentication code (MAC) with a guaranteed privacy property.

For coherent integration with electronic voting systems, the following properties must be taken into account:

1. **Secure printing.** It must be possible to generate and secretly transmit return codes for all voting options to a trusted printer.
2. **Distributed evaluation.** It must be possible to compute the return codes in a distributed way such that no knowledge is leaked about the selected voting option by the voter.

The first property can be achieved as follows. Let $\mathsf{E}_{pk}(m)$ be a ciphertext encrypted using a homomorphic cryptosystem. The secret key, unknown to printer, is verifiably secret shared among some parties. For printing m, the trusted printer, chooses a random one-time pad α and hands $\mathsf{E}_{pk}(\alpha)$ to the parties who will then execute a distributed decryption protocol for $\mathsf{E}_{pk}(\alpha)\mathsf{E}_{pk}(m) =$

$\mathsf{E}_{pk}(\alpha m)$. When αm is received back, the random pad is removed and m is printed.

Remark 1 (Code voting). One potentially serious privacy drawback with any system where votes are encrypted even in a relatively well protected environment is that it is hard to guarantee that no information about votes is leaked through malicious hardware, software, or any form of side channel.

Chaum's *code voting* idea [7] addresses this problem by letting the voters use code book encryption to submit their votes, i.e., each voter is given a list of codes along with their plaintext meanings who will then enter the code as is into a device. The Prêt à Voter [8,24] system can be viewed as a code voting scheme that uses a public key cryptosystem to prepare the code book and decode in a distributed way using a mix-net.

Motivation and contribution. We provide several proposals achieving the second property with different trust assumptions and trade-offs. Some allow a single vote to be submitted and some do not have such a restriction. Some are safe to use with write-ins and some are not. In some schemes, for each individual voter some value must be verifiably secret shared (making them less practical); whereas in other schemes, the verifiably secret shared values are not voter dependent. Some schemes demand that the tallying servers be online during the vote collecting phase, which is not desirable from a security point of view; some others allow online servers to collect the votes without any help from the tallying servers. The latter property is highly desirable since the tallying servers can decrypt the votes off-line behind an airwall.

We think it is important to provide a tool box to practitioners that allow them to choose the best trade-off between security properties, how trust is distributed, and practical and cost considerations for the given setting, since the requirements differ substantially in different election schemes and cultural contexts.

Most of our schemes work with any homomorphic public key cryptosystem, however, we concentrate on the El Gamal cryptosystem for concreteness.

2 Notation

We assume that the reader is familiar with standard definitions of public key cryptosystems, message authentication codes (MAC), zero-knowledge proofs and the random oracle model. The reader is referred to [15,16] for the required background.

El Gamal cryptosystem. Recall that the El Gamal public key cryptosystem is defined over a group G_q of prime order q with generator g over which the Decisional Diffie-Hellman assumption holds. The secret key is a random $x \in \mathbb{Z}_q$ and the corresponding public key is $y = g^x$. The encryption of a message $m \in G_q$ is $\mathsf{E}_y(m) = \mathsf{E}_y(m, r) = (g^r, my^r)$, where the randomness $r \in \mathbb{Z}_q$ is chosen randomly.

The encryption of a ciphertext $(u, v) \in G_q \times G_q$ is then defined by $\mathsf{D}_x(u, v) = vu^{-x}$. El Gamal is *homomorphic*, which means that for every two encryptions $(u_1, v_1) = \mathsf{E}_y(m_1, r_1)$ and $(u_2, v_2) = \mathsf{E}_y(m_2, r_2)$, the product ciphertext (u_1u_1, v_1v_2) is an encryption of m_1m_2 with randomness $r_1 + r_2$. Consequently, a ciphertext $(u, v) = \mathsf{E}_y(m)$ can be *re-encrypted* to produce a fresh *re-encryption* of m. This can be done without knowing the secret key, by simply multiplying the ciphertext with an encryption of identity to compute $\mathsf{RE}_y(u, v) = (ug^r, vh^r)$, for some randomness r.

Verifiable secret sharing and distributed key generation of El Gamal. Sometimes we require that a number of M parties jointly generate a public key. The corresponding secret key is *verifiably secret shared* among them such that it can be recovered by any subset of size at least λ of the parties, but it remains hidden to any coalition of size at most $\lambda - 1$. Feldman's verifiable secret sharing protocol [9] is an efficient way for distributed key generation for El Gamal. In Feldman's method, parties jointly produce a random tuple $(y_0, \ldots, y_{\lambda-1}) = (g^{x_0}, \ldots, g^{x_{\lambda-1}})$ where $x_j \in \mathbb{Z}_q$, $j \in [\lambda]$. The parties do not know x_j's; rather, each party $\ell \in [M]$ receives a share $s_\ell = f(\ell)$, where $f(z) = \sum_{i=0}^{\lambda-1} x_i z^i$. This can be viewed as sharing a secret key $x = x_0$ using the Shamir's [26] method, but parties also compute a public key $y = y_0$ and receive the Feldman commitment g^{s_ℓ} to the share of ℓth party. The same idea can be extended to Pedersen's perfectly-hiding commitment scheme [21], when verifiably sharing a secret is a preliminary goal; details are omitted.

Distributed exponentiation. Suppose an El Gamal secret key x is shared among the M parties and, given $u \in G_q$, they wish to jointly compute u^x. This can be done using the following procedure [11]. Each party ℓ, publishes $f_\ell = u^{s_\ell}$ along with a zero-knowledge proof of discrete logarithm equality. From any subset $\Delta \subseteq [M]$ of size λ of published shares, parties then compute $u^x = \prod_{\ell \in \Delta} f_\ell^{c_\ell}$, where c_ℓ's are Lagrange coefficients defined as $c_\ell = \prod_{i \in \Delta - \{\ell\}} i/(i - \ell)$. The method can be modified to work with Pedersen's verifiable secret sharing [21] as well.

Distributed decryption of El Gamal ciphertexts. When an El Gamal secret key is shared among some parties, distributed decryption of a given ciphertext (u, v) is also possible, without recovering the secret key itself. The parties first go through a distributed exponentiation protocol and compute u^x. The plaintext is then simply recovered as $m = v/u^x$.

Mix-nets. Mix-net, first introduced by Chaum [6], is an important cryptographic protocol which lies at the heart of several electronic voting systems and has other applications as well. It is executed by N voters and M mix-servers. In a re-encryption mix-net [20], mix-servers jointly generate a public key for a homomorphic cryptosystem and keep shares of the corresponding secret key. Each voter $i \in [N]$ submits a ciphertext along with a zero-knowledge proof of knowledge. When write-ins is not allowed, we assume that the voters have to choose among a set $\{m_j\}_{j \in [s]}$ of pre-defined voting options. In this case, a zero-knowledge proof must guarantee that the submitted ciphertext decrypts to one of the pre-defined choices.

When all encrypted votes have been received, the mix-net takes the list of all valid submitted ciphertexts and produces a mixed list of the decrypted plaintexts. More precisely, mix-servers take turns and re-encrypt each ciphertext. A permuted list of ciphertexts is then published along with a so called zero-knowledge *proof of shuffle*. The output list of the last mix-server is then jointly decrypted to determine the permuted list of submitted plaintexts. Any coalition of size less than λ mix-servers cannot obtain any knowledge about the correspondence between input ciphertexts and output plaintexts.

3 Online Tallying Servers

In this section we consider four return code schemes, including a few variations. All are practical but the drawback is that the tallying servers must be online during the online voting stage. The main differences between the proposed schemes come from the choice of the underlying MAC scheme Mac. Tallying servers run the mixnet and in a setup phase they jointly generate a public key y while shares of the corresponding secret key are kept private.

We assume that each voter is allowed to vote for one of a pre-defined set of choices $\{m_j\}_{j\in[s]}$. In all schemes, the ith voter submits a ciphertext $\mathsf{E}_y(m)$, where m is either one of the pre-defined choices or some random (known or unknown) representation of the designated choice. A corresponding zero-knowledge proof will also be submitted. The voter then receives a MAC tag $\mathsf{Mac}_{k_i}(m)$ as his return code, through the execution of a secure multiparty computation. Here, k_i is some (possibly) voter-dependent symmetric key shared between online vote collecting parties. Computation of such return codes are only possible by online participation of tallying servers.

In some schemes, we need to assign to each voter i a secret random value β_i and/or choice-dependent secret random values $\beta_{i,j}$ for every $j \in [s]$. This is done by assigning random encryptions $\mathsf{E}_y(\beta_i)$ and $\mathsf{E}_{pk}(\beta_{i,j})$ to the corresponding voter. In practice the ciphertexts can be defined as the output of a random oracle applied to the voter's identifier (along with that of voting alternative, if required, and other session identifiers). Thus, there is no need for the mix-servers to generate and communicate the ciphertexts to the voter.

Remark 2. We use the term "message authentication code" loosely in the sense that the schemes may not satisfy the standard definition of MACs for general purpose and the security level may also be much lower, since this suffices in our context.

3.1 Universal Hash Functions Used as MACs

Consider the ensemble of functions $F = \{f_{a,b}\}_{(a,b)\in\mathbb{Z}_q^2}$, where $f_{a,b}(x) = ax + b \bmod q$. This is the canonical example of a universal$_2$ hash function. It is well known that this is an unconditionally secure one-time MAC scheme if q is prime and large enough.

The function $f_{a,b}$ is linear, so it can be computed over homomorphic encryptions, i.e., given a ciphertext $\mathsf{E}_y(g^x)$ we can compute $\mathsf{E}_y(g^x)^a \mathsf{E}_y(g^b) = \mathsf{E}_y(g^{f_{a,b}(x)})$, which can then be decrypted in a distributed way. Any element $m \in G_q$ can be represented as g^x for a unique $x \in \mathbb{Z}_q$ since G_q is cyclic, so we can express the same relation as $\mathsf{E}_y(m)^a \mathsf{E}_y(\beta) = \mathsf{E}_y(g^{f_{a,b}(x)})$, where $m = g^x$ and $\beta = g^b$.

Thus, we can trivially compute a MAC tag for any individual party that submits a ciphertext as long as we do not do it more than once. More precisely, in a voting system we generate for the ith voter a verifiably secret shared $a_i \in \mathbb{Z}_q$ and an encryption $\mathsf{E}_y(\beta_i)$ for a randomly chosen $\beta_i \in G_q$. When the voter submits a ciphertext $\mathsf{E}_y(m_j)$ along with a zero-knowledge proof indicating that indeed one the pre-defined choices has been encrypted, he receives back the return code $m_j^{a_i}\beta_i$. Therefore, the underline MAC function is $\mathsf{Mac}_{a_i,\beta_i}(m) = m^{a_i}\beta_i$ for the ith voter. Return codes can be computed online using protocols for distributed exponentiation and decryption as explained in Sect. 2. In the setup phase, only distributed exponentiation is performed for every pre-defined voting option. The resulting ciphertexts are then communicated to a trusted third party to be securely printed, e.g., using the method described in the introduction. Furthermore, by construction the MAC tag is randomly distributed, so it can be truncated directly.

The security follows directly from the underlying MAC scheme. In addition to the danger of tallying servers being online, the drawback is that it only allows a single vote to be submitted and we need to generate a verifiably secret shared value for each voter.

3.2 One-Time Pad and Random Choice Representatives

Consider the MAC function $\mathsf{Mac}_\beta(m) = \beta m$ where key and message spaces are both G_q. The tag is a one-time pad symmetric encryption of the message and clearly not a secure MAC scheme. Indeed, an adversary can guess m and compute $\beta m'/m$ for another message m' to attempt to construct a valid MAC tag for m'. However, it *is* a one-time secure MAC for a random choice of plaintext unknown to the adversary.

A simple way to make sure that this is the case is to assign unique representatives of the choices for each voter, i.e., for the ith voter we generate random elements $\beta_{i,j} \in G_q$ for $j \in [s]$, but in encrypted form as ciphertexts $w_{i,j} = \mathsf{E}_y(\beta_{i,j})$. We can now provide the ciphertexts $w_{i,1}, \ldots, w_{i,s}$ to the ith voter. The voter then chooses the encryption of its choice, re-encrypts it, and proves in zero-knowledge that it is a re-encryption of one of its designated ciphertexts. This is a small constant factor more expensive than the corresponding proof for public choice representatives.

In the setup phase, for each voting option all representatives are shuffled, but they are published in *encrypted* form. More precisely, for each $j \in [s]$, the ciphertext list $w_{1,j}, \ldots, w_{N,j}$ is shuffled without decrypting and the re-encrypted list is made public.

The return code corresponding to the jth alternative of ith voter is then $\beta_{i,j}\beta_i$ where again β_i is a random secret value known in encrypted form $\mathsf{E}_y(\beta_i)$.

When all votes have been submitted and return codes have been received, the ciphertexts are mixed and the random elements encrypted by voters are published in permuted order. To be able to decode the actual voters' choices, the shuffled lists of representatives are also decrypted for every voting option. It is of course important that the shuffled random representatives are only decrypted *after* all votes have been collected.

The advantage of this system is that there is no need for verifiably secret shared exponents and re-voting is allowed. But zero-knowledge proofs are slightly more costly.

3.3 One-Time Pad and Standard MAC Schemes

Another way to resolve the problem encountered by solely using one-time pad is to construct a MAC scheme Mac' by combining it with a standard MAC scheme Mac [27]. More precisely, a key consists of a pair (β, k), where $\beta \in G_q$ is chosen randomly, and k is a randomly chosen key for Mac. The combined scheme Mac' is then defined by $\mathsf{Mac}'_{\beta,k}(m) = \mathsf{Mac}_k(\beta m)$.

This can be distributed in the generic way between M servers, each holding a secret key k_ℓ, by replacing the application of Mac_k by an array that is compressed with a collision resistant hash function H, i.e., we can define

$$\mathsf{Mac}_{\beta,K}(m) = \mathsf{H}\big((\mathsf{Mac}_{k_\ell}(\beta m))_{\ell\in[M]}\big),$$

where $K = (k_1, \ldots, k_M)$. It may seem that this does not suffice to satisfy our requirements for secure printing in electronic voting systems, since apparently the printer must send βm to the servers. However, the MAC keys $k_1, \ldots, k_M$ can be shared with the trusted party to print the pre-computed return codes without loss of security.

In an electronic voting system a ciphertext $\mathsf{E}_y(\beta_i)$ is generated for the ith voter and the MAC key for that voter is $(\beta_i, K) = (\beta_i, k_1, \ldots, k_M)$. The mix-servers simply take an input ciphertext $\mathsf{E}_y(m)$ submitted by the ith voter, decrypt $\mathsf{E}_y(\beta_i)\mathsf{E}_y(m) = \mathsf{E}_y(\beta_i m)$, and output $\mathsf{H}\big((\mathsf{Mac}_{k_\ell}(\beta_i m))_{\ell\in[M]}\big)$.

The advantage of this system is that there is no need for mix-servers to generate a secret shared value for each individual voter and re-voting is also allowed. The disadvantage is that it is not robust. If a server is down, the return code cannot be computed. One way to resolve this problem is to let each server verifiably secret share his symmetric key between other servers. But this guarantees security only against semi-honest adversaries and malicious servers cannot be detected.

3.4 Diffie-Hellman MAC Schemes

Recall that the Diffie-Hellman assumption states that no efficient algorithm can compute g^{ab} given g^a and g^b as input, where $a, b \in \mathbb{Z}_q$ are randomly chosen.

Furthermore, a standard hybrid argument shows that it is also hard to compute any $g^{a_i b_j}$ given g^{a_i} and g^{b_j} for $i \in [N]$ and $j \in [s]$ for some N and s, where $a_i, b_j \in \mathbb{Z}_q$ are randomly chosen. If we accept the decisional Diffie-Hellman assumption, then this is strengthened to the claim that $g^{a_i b_j}$ is indistinguishable from a randomly chosen element in G_q.

This immediately gives two MAC schemes that are compatible with mix-nets based on the El Gamal cryptosystem. Both schemes use random representations of voting options. The first variant is voter independent while the second is not. In both cases hashing the MAC tag allows truncation for any underlying group.

3.5 First Variant

We encode the jth choice by a randomly chosen element $\gamma_j \in G_q$, where in contrast to Sect. 3.2, $\gamma_1, \dots, \gamma_s$ may be public and known at the beginning. Let the mix-servers generate a verifiably secret shared MAC key a_i for the ith voter. Then, computing the MAC of the plaintext γ_j provided in encrypted form $\mathsf{E}_y(\gamma_j)$ is done by simply computing $\gamma_j^{a_i}$ by distributed exponentiation and decryption. Note that $\gamma_j = g^{b_j}$ for some $b_j \in \mathbb{Z}_q$, so the result is $g^{a_i b_j}$. To summarize, the underlying MAC scheme is defined by $\mathsf{Mac}_{a_i}(m) = m^{a_i}$ for the ith voter. This can be computed under encryption, which means that we can also provide the result in one-time pad encrypted form to a third party. This system remains secure when re-voting is allowed.

3.6 Second Variant

The first variant is somewhat impractical in that the mix-servers must generate a secret shared exponent $a_i \in \mathbb{Z}_q$ for each individual voter. We can switch the roles of randomly chosen representatives of choices and verifiably distributed secret exponents. More precisely, random elements $\beta_{i,j}$ in encrypted form as ciphertexts $\mathsf{E}_y(\beta_{i,j})$ are generated for every $i \in [N]$ and $j \in [s]$. The preparation phase and encryption procedure is exactly like that of Sect. 3.2, but now a single verifiably secret shared value a is generated and the same function $\mathsf{Mac}_a(m) = m^a$ is used for all voters.

The advantage of this scheme is that the MAC function can be evaluated in batches on submitted ciphertexts and in contrast to the construction in Sect. 3.2 the representatives may be shuffled and decrypted before all ciphertexts have been received. Re-voting is still allowed.

4 Offline Tallying Servers

In this section, we propose two schemes to resolve the online-server danger of presented schemes of Sect. 3. This is achieved without a considerable amount of performance loss or organizational overhead. The main idea is to use two independent public keys with shared secret keys. More precisely, in the setup phase, the tallying servers generate a public key y and keep shares of the corresponding secret key. Additionally, the vote collecting servers produce a public key z in the same manner.

In the online voting phase, ith voter submits a pair of ciphertexts (v_i, w_i), along with some scheme-dependent zero-knowledge proof. Here, v_i and w_i are ciphertexts encrypted under public keys y and z, respectively. The first ciphertext, v_i, is used to decode voter's choice after mixing. The second ciphertext, w_i, is an encryption of a random value, so it basically contains no information about the voter's choice. To compute the return code, w_i is simply decrypted by online servers who collect the encrypted votes. Therefore, the shares of y are never exposed during the online voting phase. Even if the secret key of z is revealed, no knowledge is leaked about the voter's choice which is encrypted under y.

When all ciphertexts have been collected, the ciphertexts list $v_1, \ldots, v_N$ is shuffled. Then, they are decrypted, and if necessary decoded, to obtain the cast votes. Since tallying can be performed behind an airwall, this approach ensures a high level of privacy for the voters.

As an alternative approach to print the pre-computed return codes, the online servers can simply share their secret key with the trusted party without loss of security.

The main requirement to be satisfied is that the two submitted ciphertexts are constructed such that they cannot be split. Below, we propose two such constructions. In addition to enhanced privacy due to airwalling, the advantage of these systems is that there is no need for voter-dependent verifiably secret shared values and re-voting is also allowed. The drawback is that the zero-knowledge proofs are more costly compared with the schemes of Sect. 3.

4.1 First Variant

In setup phase, for every $i \in [N]$ and $j \in [s]$, servers generate secret random pairs of elements $(\alpha_{i,j}, \beta_{i,j})$ in encrypted form as random ciphertext pairs $(v_{i,j}, w_{i,j}) = \big(\mathsf{E}_y(\alpha_{i,j}), \mathsf{E}_z(\beta_{i,j})\big)$ As it was explained in Sect. 3, such ciphertexts can be simply interpreted as the output of a random oracle. To vote for jth choice, the ith voter computes v_i and w_i as respective re-encryptions of $v_{i,j}$ and $w_{i,j}$. The pair (v_i, w_i) is then submitted along with a zero-knowledge proof.

In the setup phase, for each voting option j all representatives $v_{1,j}, \ldots, v_{N,j}$ are mixed and a permutation of the decrypted list $\alpha_{1,j}, \ldots, \alpha_{N,j}$ is published. When every voter i has submitted a ciphertext pair (v_i, w_i), the first elements are shuffled, decrypted and decoded.

4.2 Second Variant

In the second variant, for every $i \in [N]$ and $j \in [s]$, the pre-computed random values $w_{i,j} = \mathsf{E}_z(\beta_{i,j})$ are prepared as before. To vote for jth choice, the ith voter computes v_i as an encryption $\mathsf{E}_y(m_j)$ and w_i as a re-encryptions of $w_{i,j}$. The pair (v_i, w_i) is then submitted along with a zero-knowledge proof. Computation of return codes and tallying is straightforward. Zero-knowledge proofs are slightly less costly compared with the first variant.

5 About Write-In Candidates

Some of the systems can be adapted to allow write-ins in the sense that voters simply encrypt a representation of one of the pre-determined choices, or an arbitrary message. The zero knowledge proof of knowledge would then not impose any additional structure. Naturally, return codes can not be provided in printed form for a relatively small number of messages, so to have a chance to verify a return code for an arbitrary message the voter needs the shared MAC key.

The scheme of Sect. 3.1 is based on an unconditionally secure one-time MAC scheme, so it remains as secure for any message. The scheme of Sect. 3.2 does provide some security, but for reasons discussed in that section only if write-in votes are rare and unpredictable. Finally, the scheme of Sect. 3.5 also works for write-ins, but under a strong non-standard DDH-assumption with some care. We must assume that δ^{a_i} is indistinguishable from a random element even when the message δ strictly speaking is not randomly chosen. One way to make this a more plausible assumption is to pad a message with random bits before interpreting it as a group element, but it remains a non-standard assumption that is fragile in a complex system where slight changes may render it difficult to defend.

6 Conclusion

We present several return code systems for electronic voting applications, some of which overlaps or encompasses schemes previously proposed as separate schemes. We are unable to single out one scheme that is superior to all the other schemes in every way.

Table 1. Summary of: what is pre-computed by the tallying servers, the form of ciphertexts submitted by the ith voter to vote for the jth choice, the form of the corresponding return codes for different features of proposed schemes. Furthermore, for each scheme it is indicated if: a single global MAC key is used or if a separate key must be secret shared for each individual voter, if multiple votes can be submitted, and if the scheme matches well with write-in votes (for which the voter can not receive any pre-computed return codes in advance of course).

Section	Pre-computed	Submitted	Return code	Global MAC key	Re-voting	Write-ins
Section 3.1	$\mathsf{E}_y(\beta_i)$	$\mathsf{E}_y(m_j)$	$m_j^{a_i}\beta_i$	–	–	✓
Section 3.2	$\mathsf{E}_y(\beta_i)w_{i,j} = \mathsf{E}_y(\beta_{i,j})$	$\mathsf{RE}_y(w_{i,j})$	$\beta_{i,j}\beta_i$	✓	✓	Partly
Section 3.3	$\mathsf{E}_y(\beta_i)$	$\mathsf{E}_y(m_j)$	$\mathsf{Mac}_{\beta_i,K}(m_j)$	–	✓	–
Section 3.5	γ_j	$\mathsf{E}_y(\gamma_j)$	$\gamma_j^{a_i}$	–	✓	Partly
Section 3.6	$w_{i,j} = \mathsf{E}_y(\beta_{i,j})$	$\mathsf{RE}_y(w_{i,j})$	$\beta_{i,j}^{a}$	✓	✓	–
Section 4.1	$v_{i,j} = \mathsf{E}_z(\alpha_{i,j})$ $w_{i,j} = \mathsf{E}_y(\beta_{i,j})$	$\mathsf{RE}_y(v_{i,j})$ $\mathsf{RE}_z(w_{i,j})$	$\beta_{i,j}$	✓	✓	–
Section 4.2	$w_{i,j} = \mathsf{E}_z(\beta_{i,j})$	$\mathsf{E}_y(m_j)$ $\mathsf{RE}_z(w_{i,j})$	$\beta_{i,j}$	✓	✓	–

Instead our view is that all the schemes are simple combinations of cryptographic constructions that are well understood and that they together give a powerful toolbox to construct return codes for many types of elections. Table 1 summarizes different features of proposed schemes.

References

1. Adida, B.: Helios: web-based open-audit voting. In: Proceedings of the 17th USENIX Security Symposium, 28 July–1 August 2008, San Jose, pp. 335–348 (2008)
2. Adida, B., Neff, C.A.: Ballot casting assurance. In: 2006 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT 2006), Vancouver, 1 August 2006
3. Allepuz, J.P., Castelló, S.G.: Internet voting system with cast as intended verification. In: Kiayias, A., Lipmaa, H. (eds.) Vote-ID 2011. LNCS, vol. 7187, pp. 36–52. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32747-6_3
4. Ansper, A., Heiberg, S., Lipmaa, H., Øverland, T.A., van Laenen, F.: Security and trust for the Norwegian E-voting pilot project *E-valg 2011*. In: Jøsang, A., Maseng, T., Knapskog, S.J. (eds.) NordSec 2009. LNCS, vol. 5838, pp. 207–222. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04766-4_15
5. Benaloh, J.: Simple verifiable elections. In: 2006 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT 2006), Vancouver, 1 August 2006
6. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–88 (1981)
7. Chaum, D.: Surevote: technical overview. In: Proceedings of the Workshop on Trustworthy Elections (WOTE 2001) (2001)
8. Chaum, D., Ryan, P.Y.A., Schneider, S.: A practical voter-verifiable election scheme. In: di Vimercati, S.C., Syverson, P., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 118–139. Springer, Heidelberg (2005). doi:10.1007/11555827_8
9. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th Annual Symposium on Foundations of Computer Science, Los Angeles, 27–29 October 1987, pp. 427–437 (1987)
10. Galindo, D., Guasch, S., Puiggalí, J.: 2015 Neuchâtel's cast-as-intended verification mechanism. In: Haenni, R., Koenig, R.E., Wikström, D. (eds.) VOTELID 2015. LNCS, vol. 9269, pp. 3–18. Springer, Cham (2015). doi:10.1007/978-3-319-22270-7_1
11. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptol. **20**(1), 51–83 (2007)
12. Gjøsteen, K.: Analysis of an internet voting protocol. IACR Cryptology ePrint Archive, 2010:380 (2010)
13. Gjøsteen, K.: The Norwegian internet voting protocol. IACR Cryptology ePrint Archive, 2013:473 (2013)
14. Gjøsteen, K., Lund, A.S.: The Norwegian internet voting protocol: a new instantiation. IACR Cryptology ePrint Archive 2015:503 (2015)
15. Goldreich, O.: The Foundations of Cryptography. Basic Techniques, vol. 1. Cambridge University Press, Cambridge (2001)
16. Goldreich, O.: The Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press, Cambridge (2004)

17. Heiberg, S., Laud, P., Willemson, J.: The application of I-voting for Estonian parliamentary elections of 2011. In: Kiayias, A., Lipmaa, H. (eds.) Vote-ID 2011. LNCS, vol. 7187, pp. 208–223. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32747-6_13
18. Heiberg, S., Lipmaa, H., van Laenen, F.: On E-vote integrity in the case of malicious voter computers. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 373–388. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15497-3_23
19. Lipmaa, H.: Two simple code-verification voting protocols. IACR Cryptology ePrint Archive, 2011:317 (2011)
20. Park, C., Itoh, K., Kurosawa, K.: Efficient anonymous channel and all/nothing election scheme. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 248–259. Springer, Heidelberg (1994). doi:10.1007/3-540-48285-7_21
21. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). doi:10.1007/3-540-46766-1_9
22. Puigalli, J., Guasch, S.: Cast-as-intended verification in Norway. In: 5th International Conference on Electronic Voting 201 (eVOTE 2012), Co-organized by the Council of Europe, Gesellschaft für Informatik and E-voting.CC, 11–14 July 2012, Castle Hofen, Bregenz, Austria, pp. 49–63 (2012)
23. Rosen, A., Ta-shma, A., Riva, B.: Jonathan (Yoni) Ben-Nun. Wombat voting system (2012)
24. Ryan, P.Y.A., Schneider, S.A.: Prêt à voter with re-encryption mixes. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 313–326. Springer, Heidelberg (2006). doi:10.1007/11863908_20
25. Sandler, D., Derr, K., Wallach, D.S.: Votebox: a tamper-evident, verifiable electronic voting system. In: Proceedings of the 17th USENIX Security Symposium, 28 July–1 August 2008, San Jose, pp. 349–364 (2008)
26. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
27. Wikström, D.: Proposed during rump session of evote 2015 (2015)