Improving Helios with Everlasting Privacy Towards the Public

Denise Demirel

Department of Computer Science, Cryptography and Computer Algebra Group, Technische Universität Darmstadt, Darmstadt, Germany.
E-mail: ddemirel@cdc.informatik.tu-darmstadt.de

Jeroen van de Graaf

Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, CEP 31270-901, Brazil.
E-mail: jvdg@dcc.ufmg.br

Roberto Araújo

Faculdade de Computação, ICEN, Universidade Federal do Pará, Brazil.
E-mail: rsa@ufpa.br

Abstract

In this paper we propose improvements on the Helios voting protocol such that the audit data published by the authority provides everlasting privacy, as opposed to the computational privacy provided currently. We achieve this with minor adjustments to the current implementation. For the homomorphic Helios variant we use Pedersen commitments to encode the vote, together with homomorphic encryption over a separate, private channel between the user and Helios server to send the decommitment values. For the mix-net variant we apply a recent result which shows that mixing with everlasting privacy is possible.

Observe that we do not claim everlasting privacy towards the server, which, if dishonest, could try to break the homomorphic encryption scheme used in the private channel. Thus towards the authority the voter's level of privacy is identical to what Helios currently offers. However, our protocol is much harder to attack by an outsider: apart from having to break the computational assumption, an adversary must intercept the communication between the voter and the server to violate ballot privacy of that voter. The feasibility of such an attack depends on the way both parties choose to implement this channel. Both contributions are generic in the sense that they can be applied to other voting protocols that use homomorphic tallying or mixnets.

1 Introduction

1.1 Motivation

Voting protocols seem to come in two settings: either the protocol achieves computational privacy of the ballot and unconditional correctness of the vote count, or the reverse: everlasting privacy and computational correctness. Just as with bit commitments, achieving both everlasting privacy and unconditional correctness seems to be impossible, but this question is not fully settled yet (see [7]).

The overwhelming majority of voting protocols is based on homomorphic encryption and/or on encryption mixes, so all these protocols have in common that they provide only computational privacy. This is somewhat surprising. Already two decades ago Chaum argued, in the context of credential mechanism [28], that privacy should be everlasting, since individuals cannot be expected to assess the strength of cryptographic mechanisms.

In addition, since storage is becoming cheaper every day, we must assume that the data on the bulletin board will be stored forever. This means that the moment the cryptographic assumption on which the privacy of the ballots was based is broken, it will be possible to decrypt all published information. In other words, with computational privacy we can almost be sure that 30 or 300 years from now we can know who voted for whom. This could raise the possibility for some nasty scenarios, for instance a dictator who has come to power goes after people who have voted against him (or his father)

several decades ago. Voter privacy is also of interest for elections which are not on state level. If a university or company elects a new director it would be unpleasant for each candidate, also several years after the election, to detect which co-worker voted against him or her.

In other words, many newly proposed voting protocols suddenly have a new, potentially dangerous, property that classical protocols never had. Therefore we believe that the quest for finding suitable protocols with everlasting privacy is of utmost importance.

1.2 Helios

Helios [20] is a web application for internet voting which is online accessible, easy to use, and free of charge. The system is fully open and provides end-to-end verifiability, meaning that the voters can check all steps from the vote casting process to the computation of the election result. In addition, in 2009 the system was extended to enable election creation and voting even via a Twitter account. This makes Helios a convenient tool to support representatives elections for companies, online groups, local clubs and many more.

After its introduction in 2008 [1], Helios was used on several election levels. In 2009, for instance, the Princeton Undergraduate Student Government election [21] and the election of the President of the Université Catholique de Louvain [3] was realized with Helios. In addition, the Brazilian Society of Computing used this voting system last year to elect their Board of Directors.

The Helios voting system is end-to-end verifiable by providing individual and universal verifiability. Thus, each voter can check whether the election outcome was calculated correctly. Further, all voter options are encrypted locally on the voter's computer before they are sent to the Helios server. To ensure that only anonymized votes or the election outcome is decrypted, the private key is generated in threshold fashion and distributed among several key trustees. This provides a high level of voter privacy.

However, a serious drawback of Helios is that each voter is linked to a vote which is published in encrypted form, thus offering computational privacy only. Therefore, as soon as the underlying cryptosystem gets broken, everyone can decrypt the information written on the Helios bulletin board and reveal the votes cast by each voter.

In this paper we show how to obtain everlasting privacy towards the public, while minimizing changes to the current Helios implementation.

1.3 A brief and intuitive explanation

Pedersen commitments[1] are expressions of the form $u(t,s) = \alpha^s\beta^t \in G$ that can be used as a commitment scheme which provides everlasting privacy. Pedersen commitments are computationally binding, provided that the party who commits cannot break the discrete log in the group G. It is trivial to see that Pedersen commitments are homomorphic: $u(t_1,s_1)u(t_2,s_2) = \alpha^{s_1}\beta^{t_1}\alpha^{s_2}\beta^{t_2} = \alpha^{s_1+s_2}\beta^{t_1+t_2} = u(t_1+t_2, s_1+s_2)$. This property is what makes them so useful for voting.

A problem, however, is that, unlike conventional encryption, a commitment u can encode[2] any value, so without knowing s it is impossible to decode to t. We solve this problem by encrypting s and t separately using a conventional homomorphic encryption scheme, like Paillier: $v = P(s)$ and $w = P(t)$. This encryption is *not* published on the Helios bulletin board, but sent to the server through a private channel. In other words, we use a second channel through which we send the auxiliary information s and vote t.

We present two variations of this idea. The first variation can be thought of an extension of [12] (which is the current implementation of homomorphic tallying in Helios using exponential ElGamal) and a simplification of [11] (which uses Pedersen commitments and secret sharing among multiple authorities). A vote for candidate i is represented as a vector $\langle t_1, ..., t_l\rangle$ which is 0 everywhere, except in the ith position, where it equals 1. Each entry of this vector is encoded using Pedersen commitments, and sent to the bulletin board.

In order to count the total for candidate i, the server computes $u_i^* = \prod_j u_i(j)$, where the j in parenthesis denotes an index ranging over all the voters who submitted a valid vote to the server. Because of the homomorphic property, we have that $u_i^* = \alpha^{s_i^*}\beta^{t_i^*}$ where $s_i^* = \sum_j s_i(j)$ and $t_i^* = \sum_j t_i(j)$, and where the sum again ranges over all the j.

Now in order to publish t_i^*, that is, the total of votes for candidate i, and show its correctness, the server must know s_i^* and t_i^*. This is possible because it received from each user the value s_i and t_i homomorphically encrypted, sent through a private channel. By computing $v_i^* = \prod_j v_i(j) = \prod_j P(s_i(j)) = P(s_i^*)$ as well as $w_i^* = \prod_j w_i(j) = \prod_j P(t_i(j)) = P(t_i^*)$ and then decrypting it, the server finds s_i^* and correspondingly t_i^*. This procedure is applied to each candidate, so this is how the tally is determined.

The decommitment values s_i^* and votes t_i^* are decrypted in threshold fashion. This requires that the private key consists of several key portions which are shared

[1] Though usually attributed to Pedersen, expressions of this form were actually first presented in [8] (see page 98).

[2] Since we use bit commitments to *blind* the message, we use the term *encode* instead of the verb *encrypt*.

among several key trustees. As a result, a predefined number of authorities has to participate to decrypt the decommitment values what provides a high level of computational privacy.

Opening the computed commitment $u_i^* = \alpha^{s_i^*}\beta^{t_i^*}$ by publishing the values t_i^* and s_i^*, is sufficient to show correctness and provide universal verifiability to the public, provided that the server cannot compute the discrete log of α with respect to β before this publication takes place.

In the second variation we use te same idea, of encoding a secret (any secret, not necessarily votes represented as 0 or 1) in a Pedersen commitment and send the decommitment value over a private channel using homomorphic encryption, can be used to implement a mix-net with everlasting privacy. Though conceptually simple, the details of this get somewhat elaborate; they are described in [14]. So the second variations we present is to substitute the computational mixing that Helios incorporates by this new everlasting mixing protocol.

1.4 Contribution of this paper

Summarized the protocol we propose has the following properties

Individual Verifiability Each voter can verify that the vote was cast as intended and recorded as cast.

Universal Verifiability Any observer can verify that the votes published on the Helios bulletin board were tallied as recorded.

Correctness Unauthorized modifications to the tally will be detected with overwhelming probability even if all authorities conspire.

Everlasting Privacy towards the public All the information published on the bulletin board does not reveal any information about the vote cast.

It should be emphasized that we are NOT claiming everlasting privacy towards the Helios server. This authority gets to see the Paillier-encryptions of the $t_i(j)$, so once it can break Paillier, it can determine each vote. And if it convinces all the key trustees to conspire, the Helios server does not even have to wait for Paillier to be broken; if all these entities conspire all votes can be decrypted. This is equivalent to what Helios provides.

The proposed protocol is only an improvement with respect to the data published on the bulletin board. Under the assumption that the authority and trustees are honest, attacking this new version requires an adversary both to intercept the communication between voter and server *and* break a cryptographic assumption. Not impossible, depending on the way this private channel is implemented, but certainly requiring much more effort compared to the current Helios implementation.

1.5 Comparison to other work/ Related Work

To the best of our knowledge, the first voting protocol that provides everlasting privacy was presented by Bos [6], [5]. His voting protocol only allows Yes/No (encoded as 1 and 0, respectively) and votes are encoded as simple Pedersen commitments, i.e. $l = 1$. The votes and decommitment values are added using Dining Cryptographer nets modulo suitably chosen moduli. The Dining Cryptographer nets used assume that all voters are online simultaneously, which for a large-scale election is not realistic.

A few years later Cramer, Franklin, Schoenmakers and Yung (CFSY) presented another protocol with everlasting privacy [11]. Their basic version also uses $l = 1$ but it encodes a masked vote of two options as $\{-1, 1\}$. It uses Pedersen secret sharing [26] to split the masked vote among the authorities, who add the votes and decrypt the result in a distributed fashion.

For almost a decade, no progress was made on voting protocols with everlasting privacy. Then suddenly three different approaches appeared almost simultaneously and independently.

In [33], an off-line version of the Dining Cryptographers protocol is presented, in an attempt to resolve the main disadvantage of the Bos protocol. Unfortunately, to catch a disrupter large amounts of bit commitments with linear properties are needed. But though the protocol presented there appears sound, its application to voting is still fraught with seemingly unsurmountable problems.

Simultaneously, Chaum invented PunchScan [29, 22]. An important theoretical implication of PunchScan is that it showed how to build an election protocol based on bit commitment. However, the original papers used conventional symmetric encryption as a commitment scheme, yielding computational privacy only. Its successor, Scantegrity [9] is also based on bit commitments.

In [32], a merge between Prêt-à-Voter and PunchScan is proposed, using the former's ballot layout, and the latter's bit commitment scheme plus auditing process. If used in combination with an uncondionally hiding bit commitment scheme, an extremely simple voting protocol with everlasting privacy is obtained.

Moran and Naor have published two different protocols on voting with everlasting privacy. The protocol presented in [23] uses a voting machine, and is based on a generic commitment scheme. An optimized version uses Pedersen Commitments. The protocol presented in the second paper [24, 25] is strongly based on PunchScan, but has a very interesting extra twist. The voter must vote by splitting her choice over two ballot halves, which are sent to two different authorities. These two authorities can compute the tally, but they cannot reconstruct an in-

dividual vote without conspiring. So there is no single point of failure with respect to privacy.

In [14] a universal verifiable mix-net is introduced whose input is a set of tuples. Each pair consists of a secret, encoded using Pedersen Commitments, and the corresponding decommitment value are encrypted applying standard homomorphic public-key cryptography. Using several mix-nets in parallel secret sharing can be implemented too. This is the first approach providing a mixing process for votes encoded in an information-theoretically secure way. Further, no data published during the verification process reveals information which would enable a computationally unbounded attacker to decrypt the vote or to map an encrypted value to a decrypted vote.

1.6 Paper outline

The structure of the paper is as follows: After a description of the current Helios voting system in Section 2, we present in Section 3 the improved protocol for homomorphic tallying. In Section 4 we give formal statements of the gained properties and conclude with discussion in Section 5.

2 The Helios voting system

Helios is a publicly accessible and publicly verifiable web-based voting system. Everyone can visit the web site to register and run an election, which is publicly auditable.

Phase 1: System initialization

1. The user creates an election by setting the election name as well as the date and time when the vote casting process begins and ends. Subsequently, she becomes administrator and can set up the election parameters. Helios v4 is going to support several voting methods (e.g. plurality voting and ranking), besides offering two tallying mechanisms: homomorphic tallying or mix-nets.

2. The administrator creates a list of eligible voters with their corresponding email addresses.

3. The voting software automatically generates the election templates like the ballot paper and the private and public key needed to encrypt and decrypt the votes cast.

Phase 2: The voter's perspective

1. When the vote casting process begins, each voter receives an email containing her username, her election-specific password, the hashed election parameters, and a URL that directs to the voting booth application.

2. Following the link, a single-page web application, implemented in JavaScript, is started which downloads the election parameters and templates. The information downloaded is sufficient to update the HTML according to the voting decision and to perform the encryption and auditing process of the ballot (the voter can go offline). The voter can check whether the downloaded parameters are consistent with the hash sent by email.

3. She fills out the electronic ballot and the JavaScript application encrypts her selection and publishes the hash of the generated ciphertext.

4. The voter now has the option to to audit the encrypted information. In this case the application reveals the randomness used to encrypt her voting decision. Having this information, the voter can recalculate the published hash, either by own code or by software provided by a trustworthy third party. After that, the application deletes the voting decision and the voter has to fill out a new electronic ballot, which is encrypted using a fresh random value. Note that so far no authentication has taken place, so also people who are not eligible to vote or already cast their vote are able to challenge the system. The voter can repeat this auditing process until she is convinced that the voting decisions got correctly encrypted and decides to cast the ciphertext containing her vote.

5. Subsequently, the application first clears the plaintext and randomness from its scope and then asks the voter to authenticate with the login and password received by mail. Note that the computer has to be online for this step so the information can be sent to the Helios server. The server replies with a success message in case the received login and password are correct and publishes the encrypted voting decision on the Helios bulletin board.

6. After that, the JavaScript application clears its scope and sends a success message to the voter.

7. **(Individual verifiability)** The voter can visit the bulletin boards and check whether her vote is included in the tally. Furthermore, she can compare the hash with the information on her receipt and verify that the vote has been recorded unmodified.

Phase 3: Tallying and publishing the votes

Depending on the parameters, the encrypted votes are either made anonymous by shuffling and re-encrypting followed by decrypting and tallying, or the election outcome is generated by homomorphic tallying and decrypted afterwards. Note that, although the administrator initiates each process, the computations are performed on the Helios server, and sensitive information (like the private key or the permutations used of the mix-net) are never revealed.

Phase 4: Verification of the tally – universal verifiability

The administrator triggers the server to publish a proof for correct decryption. If mix-nets are used, it is also publicly shown that the votes have been shuffled and re-encrypted correctly.

3 Description of the new protocol

3.1 Cryptographic tools

Pedersen commitments and Paillier encryption

In order to obtain everlasting privacy in combination with homomorphic properties we will encode values using Pedersen commitments. E.g. the ciphertext $u = \alpha^s \beta^t$ encodes the value t by "blinding" it with a random number α^s. To provide decoding, the decommitment value or *key* s and the value t will be encrypted with Paillier: $v = \gamma^s r^N$, $w = \gamma^t r'^N$ and transferred over a private channel.

So the encoding of t takes three random values and has three components: $Enc(t, s, r, r') = \langle u, v, w \rangle = \langle \alpha^s \beta^t, \gamma^s r^N, \gamma^t r'^N \rangle$. The first (Pedersen) component offers everlasting privacy but is impossible to open without any help. The second (Paillier) component contains the auxiliary information (s) and the third the value (t) needed to open the first. These components offers only computational privacy and must be sent over a private channel.

For the whole process to work we need a homomorphic property, so we must carefully choose the groups in which we are working. To this end we use the construction proposed by Moran and Naor [25]. For the second component we use standard Paillier encryption using an integer $N = p_1 p_2$ as public key. The primes p_1 and p_2 are the private key; they must be safe primes. For the first component we use the order N subgroup of $\mathbb{Z}^*_{4N+1}$, where $4N + 1$ must be prime too.

We therefore obtain the homomorphic property $\mathsf{Enc}(t_1, s_1, r_1, r'_1) * \mathsf{Enc}(t_2, s_2, r_2, r'_2) = \mathsf{Enc}(t_1 + t_2, s_1 + s_2, r_1 \cdot r_2, r'_1 \cdot r'_2)$, where $*$ stands for pairwise multiplication in $\mathbb{Z}^*_{4N+1}$ and $\mathbb{Z}^*_{N^2}$ in the first resp. second component, $+$ stands for addition in $\mathbb{Z}_N$, and $\cdot$ for multiplication in $\mathbb{Z}^*_{N^2}$.

To generate the key pair consisting of the public and private key in threshold fashion (without a trusted dealer) techniques like those implemented in [13, 17] can be applied. How the shares are used to decrypt the Paillier ciphertext in threshold fashion is shown in [16].

Random Beacon

We use a random beacon for generating the generators α and β and for the coins flipped during verification. The Helios server oversees and publishes these generated random values so everyone can verify the proofs. Having a trusted random beacon is reasonable assumption, see for instance [25].

Proof of correct encoding

Note that it is crucial that the value s and t used in the first (Pedersen) component and in the second (Paillier) component are the same. Therefore the person who encodes the value t must prove this fact, besides showing knowledge of t, the random decommitment value s and the randomness r and r' used to generate u, v and w respectively. This can be accomplished by using a standard cut-and-choose verification protocol like in [25].

1. The prover privately chooses random values $s', t' \in \mathbb{Z}_N$ and $r'', r''' \in \mathbb{Z}_{N^2}$, generates a second pair $u' = \alpha^{s+s'} \beta^{t+t'}$, $v' = \gamma^{s+s'} r^N r''^N$ and $w' = \gamma^{t+t'} r'^N r'''^N$ and sends the triple $\langle u', v', w' \rangle$ to the verifier.

2. The verifier challenges 0 or 1.

3. If 0, then the prover reveals s', t', r'' and r''' . Now the verifier checks whether $u' \stackrel{?}{=} u\ \alpha^{s'} \beta^{t'}$, $v' \stackrel{?}{=} v\ \gamma^{s'} r''^N$ and $w' \stackrel{?}{=} w\ \gamma^{t'} r'''^N$.

4. If 1, then the prover reveals $s + s', t + t'$ and $r \cdot r'$. Now the verifier checks whether $u' \stackrel{?}{=} \alpha^{s+s'} \beta^{t+t'}$, $v' \stackrel{?}{=} \gamma^{s+s'} (r \cdot r'')^N$ and $w' \stackrel{?}{=} \gamma^{t+t'} (r' \cdot r''')^N$.

If two different values s or t were used in constructing u, v and w, then this will be detected with probability $\frac{1}{2}$ unless the prover can solve the discrete log problem. This process is repeated several times until the probability of not being detected is below a certain value.

Note that during the election for each vote just a commitment is published while the corresponding s and t to open it are send towards a private channel. Thus this proof also prevents attacks where a malicious voter

copies a vote from the bulletin board and tries to cast it in a modified form (like explained in [10]).

In this proof, the prover is the browser software, and the verifier is the Helios server.

Encoding of the votes

We represent the vote for candidate i as a vector $\langle t_1, ..., t_l \rangle$ which is 0 everywhere, except in the *ith* position, where it equals 1. Each entry is encoded individually using the encoding scheme presented above, i.e. $\langle u_i, v_i, w_i \rangle = \overline{u}(j)$.

To avoid cheating it must be verified that the encoded vector indeed represents a vote. For this two things need to be verified.

1. It must be shown that each t_i is either 0 or 1. This can be proven using a straightforward modification of the protocol presented in figure 1 of [11], which proves that a Pedersen commitment is either -1 or 1. For details on the modified protocol, please consult Figure 4 in the Appendix.

2. It must be shown that only one t_i equals 1. This can be done by giving a proof that $\sum_{i=1}^{l} t_i = 1$, which is achieved by showing knowledge of the discrete log of $\prod_{i=1}^{l} u_i \beta^{-1}$ with respect to α using a standard Schnorr protocol.

Both proofs are published on the bulletin board and can be verified by any third party. Note that although some of the proofs described are interactive, a non-interactive version can easily be obtained using the Fiat-Shamir technique [15].

3.2 Protocol based on homomorphic tallying

Parentheses are used to refer to values belonging to a particular voter.

Phase 1: System initialization

1. During the election setup, the Helios server computes the system's parameters G, α, β by using a trusted random beacon. Further, like in the traditional Helios approach, in threshold fashion a secret key sk is generated and the corresponding public key pk is published.

Phase 2: The voter's perspective

1. Voter j chooses a candidate, and enters her choice in the JavaScript application. Her browser generates random decommitment values $s_1, \ldots, s_l$, random strings $r_1, \ldots, r_l$ and $r'_1, \ldots, r'_l$ and computes $\langle \mathsf{Enc}(t_1, s_1, r_1, r'_1), \ldots, \mathsf{Enc}(t_l, s_l, r_l, r'_l) \rangle = \langle \overline{u}(j), \overline{v}(j), \overline{w}(j) \rangle$.

2. The voter can challenge the system by auditing the encoded vote. The browser reveals the used random decommitment values and the voter can verify the correct encoding either by recalculating or by using software provided by a trusted third party. An audited ballot cannot be cast, so she is asked to fill out a fresh ballot.

3. If the voter decides to cast the encoded vote, the browser sends the encrypted decommitment values $\overline{v}$, the login, the password and the ciphertexts $\overline{u}$ through a private channel to the Helios server. The nature of this private channel will be discussed later in Section 5.1.

   Along with it the browser sends a proof that this is a valid vote as explained in 3.1. This proof is published together with $\overline{u}$ on the bulletin board.

   Also, the browser must privately prove that the encrypted s and t it sends privately is consistent with the decommitment value used to encoded the voting decision. This can be accomplished by using the ZK Proof of correct encoding described in Section 3.1.

4. **(Individual verifiability)** After the election is over, the voter goes to the Helios bulletin board, types her name or voter id, and verifies that $\overline{u}(j)$ is included in the list of votes cast.

A schematic overview of this process is given in Figure 1.

Phase 3: Tallying and publishing the votes

Let there be l candidates and V voters, and let $i \in \{1 \ldots l\}$ range over all candidates and $j \in \{1 \ldots V\}$ range over all voters. Then for each candidate i the following steps are performed:

1. The server computes $v_i^* = \prod_j v_i(j)$ and $w_i^* = \prod_j w_i(j)$, decrypts $s_i^* = \sum_j s_i(j)$ and $t_i^* = \sum_j t_i(j)$ with the help of the key trustees and publishes the result on the Helios bulletin board.

2. The server computes $u_i^* = \prod_j u_i(j) (\mathrm{mod}\, q)$ and publishes it.

3. Paillier is semantically secure.

A schematic overview of this process is given in Figure 2.

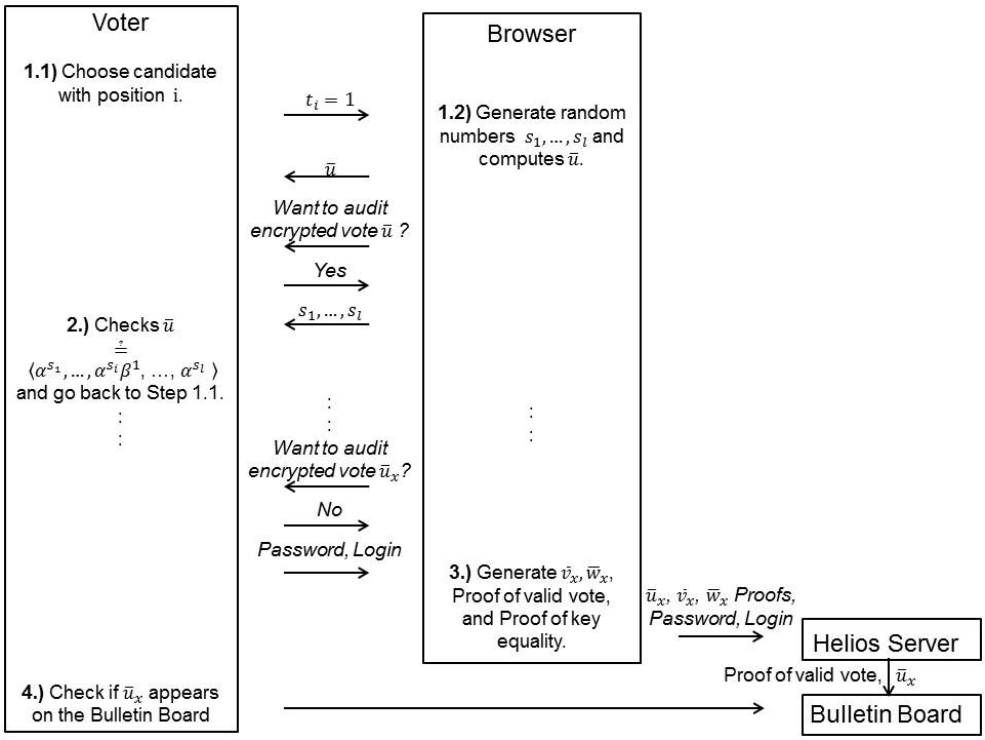

Figure 1: Voter Perspective of the Vote Casting Process

Phase 4: Verification of the tally – universal verifiability

For each vote cast, the ID of the voter and the value $\overline{u}_j = \langle u_1(j), \ldots, u_l(j) \rangle$ must appear on the Helios bulletin board. After the election, any person or entity with sufficient resources can check the correctness of the tally, as follows:

1. For each i compute $u_i^\dagger := \prod_j u_i(j)$ and check whether $u_i^\dagger \stackrel{?}{=} u_i^*$
2. Check whether $u_i^\dagger \stackrel{?}{=} \alpha^{s_i^*}\beta^{t_i^*}$.

4 Assumptions and Properties of the Protocol

Our scheme relies on the following assumptions:

1. The election officials running the Helios server cannot break the discrete log problem for the parameters chosen before the elections ends;
2. There exists a private channel between the user's browser and the server;
3. No information about the vote, other than the encoding of the vote sent to the server and the receipt to be given to the user, leaves the user's browser;

Based on these assumptions our scheme offers the same properties of Helios, but with everlasting privacy towards the public:

Correctness The correctness of the vote count is guaranteed, unless the authorities can break the discrete log problem in G before the election ends. This statement is true even if the server and key trustees conspire. This is a consequence of the fact that Pedersen commitment are used to encode votes.

Individual Verifiability In the first place, each voter can verify the ballot construction because of step 2.2, which allows her to challenge the browser application to verify the encoding. If each voter verifies one encoding, then a cheating server has probability of $1/2$ of being caught for each wrongly constructed ballot. In addition, each voter is able to verify that her vote appears on the Helios bulletin board. As presented in Phase 2.4, the voter sends a vector $\overline{u}(j)$ to the server which publishes its values on the bulletin board, together with a cryptographic hash used for comparison. Immediately after the server publishes these values, the voter can verify if her vector $\overline{u}(j)$ is the same on the bulletin board by comparing the hashes.

Universal Verifiability Any observer can verify that the tally was calculated correctly. This follows immedi-

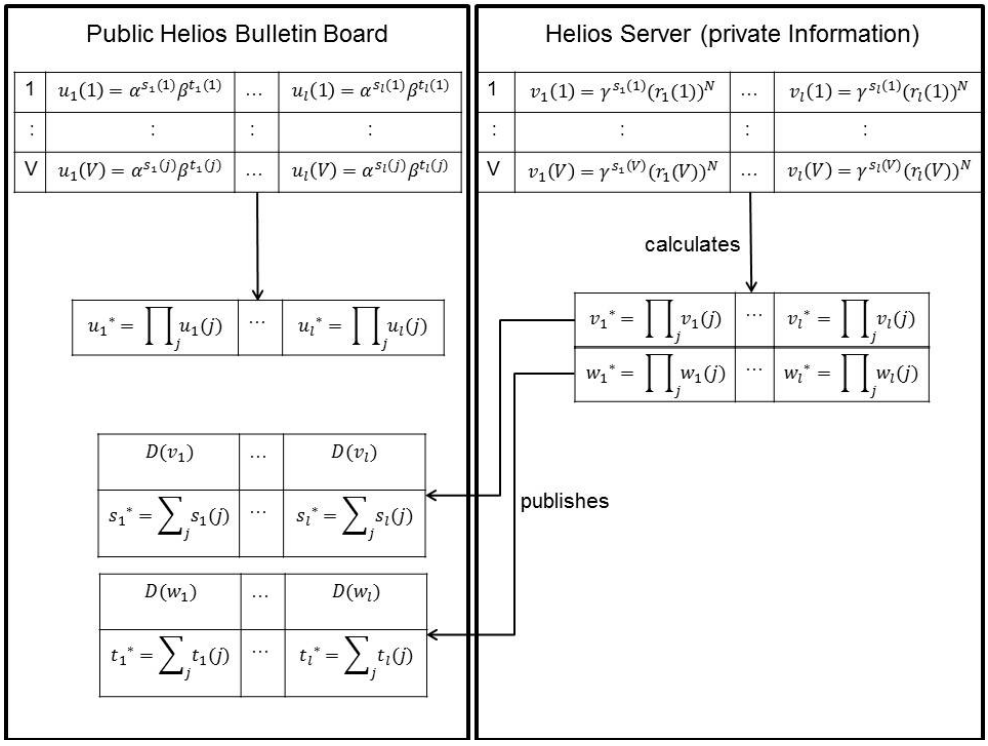

Figure 2: A schematic overview of the tallying of the results by the server. Here $D(\cdot)$ stands for Paillier decryption (with help from the trustees).

ately from the correctness and from the checks described in Steps 4.1 and 4.2. Once the correctness of u_i^* has been verified, and it has been confirmed that s_i^* is the decommitment value of the encoded value t_i^*, the election result t_i^* must be correct (under the DL assumption).

Everlasting Privacy towards the public The protocol provides everlasting privacy towards the public. Pedersen commitments are perfectly hiding. So an encoded vote published on the bulletin board can be any voting decision with equal probability. Further, all proofs presented to the voter in order to verify the correctness are witness hiding so do not reveal any information about the used decommitment values. Therefore in order to reveal the voting decision cast by a voter an attacker needs additional information besides the data published by the electronic voting system.

5 Discussion

5.1 Private channel between the voter and the authority

In the current implementation of Helios, all that a future adversary has to do is to sit and wait until the underlying homomorphic encryption scheme used has been broken, then obtain a copy of the bulletin board from some web archive and perform the computations to decrypt the votes. That is, currently attacks on Helios can be mounted retro-actively to recover all the votes.

In contrast, in the protocol proposed here, just obtaining a copy of the bulletin board is not enough to reconstruct the ballots cast. For *each* vote it wants to recover, the adversary has to intercept the communication between the voter and server *at the time this takes place*, requiring a much larger effort on behalf of the adversary. In addition, the cryptographic algorithm used to protect the communication between the voter and the server needs to be broken. But unlike in the original Helios scheme, the voter and the server are no longer restricted to using homomorphic encryption; they can superencrypt these messages using any algorithm available to them. In particular, to get everlasting privacy they can use a one-time pad, though this implies that the parties need to exchange

a key through a secure channel. In some settings this is perfectly doable; for instance, a random key can be printed as a two-dimensional barcode on a piece of paper which can be handed to the voter personally, or sent by registered mail.

Of course, if implementing a private channel is not considered necessary, not practical or too expensive, then the parties can use any public key algorithm such as McEliece. In this case the system does not offer unconditional privacy, but now this is a pragmatic choice made by the two parties, depending on the circumstances. The current Helios implementation does not offer such flexibility; its privacy is inherently limited by the homomorphic encryption algorithm used.

5.2 Solution with more than one authority

Though it is possible to generalize this scheme to multiple authorities, where votes are split and send to several authorities the advantage is not that clear. One would obtain several authorities and several key trustees (as in the single authority scheme), with subtle variations of computational and unconditional privacy. However, if this is really a concern one may be better off implementing the CFSY protocol which is based on Pedersen secret sharing [27, 18, 19].

Multiple authorities may seem desirable from a cryptographic point of view but they are not from a practical point of view. In many of the organization where Helios was applied, having one well-configured server running reliably is hard enough. Having several is difficult, and is often perceived as a serious possibility of failure.

For this reason, and because in many election where Helios is used voters are not so much worried about their vote being leaked through election authorities, we only present the single authority solution.

5.3 Protocol based on mix-nets

Encoding of the votes

Using the mix-net approach no special format of the ciphertext is needed because the original Helios ballot encoding can be adapted. In the following, t refers to voting decision and we do not address how this value is generated. Encoding the voter's choice we obtain $h = \alpha^s \beta^t$.

Phase 1: System initialization

1. During the election setup, the Helios server computes the system's parameters G, α and β by using a trusted random beacon. Further, a secret key sk is generated (in threshold fashion) and the corresponding public key pk is published.

Phase 2: The voter's perspective

1. Voter j fills in the electronic ballot paper. After that, the browser generates random number s and computes the ciphertext $u(j) = \alpha^s \beta^t$ to the voting decision t.

2. The JavaScript application has to prove that the encoded voting decision equals the voter's choice. In this case the same verification process can be performed as described in Section 3.2 for the homomorphic tallying approach.

3. If the voter decides to cast her vote, the ciphertext is sent to the Helios server along with the encrypted decommitment value (s), and vote (t), login, password, and the ZK Proof of correct encoding, as described in Section 3.1.

4. **(Individual verifiability)** After the poll closed the Helios bulletin board publishes the received votes and each voter can verifies that the ciphertext $u(j)$ is listed under the cast votes.

Phase 3: Tallying and publishing the votes

Before the votes cast can be decrypted and tallied it is necessary to break the link between the voters and their ciphertext. To accomplish this, a mix-net like proposed in [14] with n mixes $A1, A2, \ldots An$ can be used. Let there be V voters, the first mix A1 downloads all V published votes $\{u(j)\}_{j=1}^{V}$ and receives the encrypted decommitment values $\{v(j)\}_{j=1}^{V}$ and vote $\{w(j)\}_{j=1}^{V}$ by the Helios sever. A_1 will recode its input set $\{\langle u(j), v(j), w(j)\rangle\}_{j=1}^{V}$ as follows:

1. $A1$ generates random value $s_{A1}(j)$ and calculates $u_{A1}(j) = u(j)\ \alpha^{s_{A1}(j)} = \alpha^{s(j)+s_{A1}(j)}\beta^{t(j)}$, $v_{A1}(j) = v(j)E(s_{A1}(j)) = E(s(j) + s_{A1}(j))$ and $w_{A1}(j) = w(j)E(0) = E(t(j) + 0)$.

2. $A1$ puts the tuples in numerical order what defines a permutation π_{A1} between the input set $\{\langle u(j), v(j), w(j)\rangle\}_{j=1}^{V}$ and output set $\{u_{A1}(k), v_{A1}(k), w_{A1}(k)\}_{k=1}^{V}$ with $k = \pi_{A1}(j)$.

3. The recoded voting decisions $u_{A1}(k)$ are published on the Helios bulletin board while the corresponding set of encrypted decommitment values $v_{A1}(k)$ and votes $w_{A1}(k)$ is sent to mix $A2$ via a private channel.

4. The next mix $A2$ process its input the same way until the output of the final mix An gives the output of the mix-net.

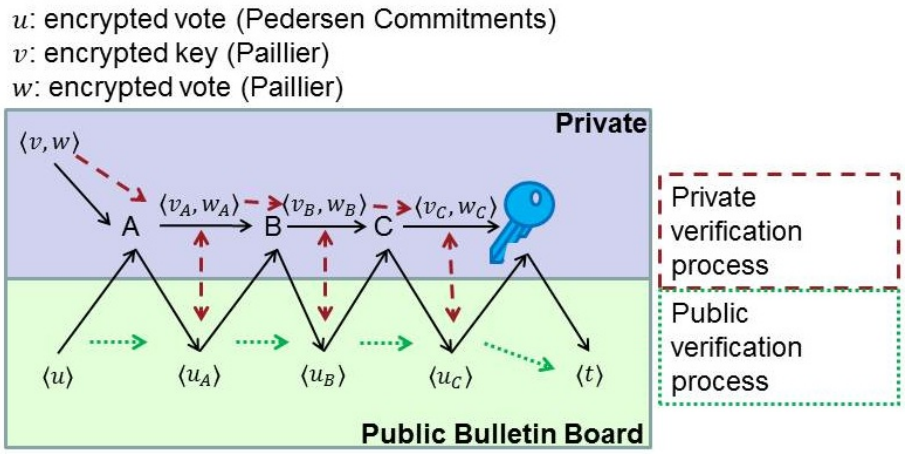

Figure 3: Overview over Mixing and Verification Process Using a Mix-net with Three Mixes

5. The final output of the mix-net consisting of the triple $\{\langle u_{An}(l), v_{An}(l), w_{An}(l)\rangle\}_{l=1}^{V}$ is published on the Helios bulletin board.

6. With the help of the key trustees first the keys $\{v(l)\}_{l=1}^{V}$ and following the votes $\{w(l)\}_{l=1}^{V}$ are decrypted and published.

7. Finally the resulting filled ballots (t) are evaluated and tallied.

An overview over this process for three mixes is given in Figure 3.

Phase 4: Verification of the shuffling process and the tally – universal verifiability

1. Each mix Am provides a proof that the published output u_{Am} is a correct recoding and permutation of the input u_{Am-1}, downloaded from the Helios bulletin board. This can easily be accomplished by a ZK proof (see for instance [31]) showing that Am has knowledge of

   (a) permutation π_{Am}

   (b) the values used to recode u_{Am-1} to u_{Am}

2. Each mix Am proves privately to the verifiers that the output v_{Am} and w_{Am} is a correct re-encryption and permutation of the input v_{Am-1} and and w_{Am-1}, sent by $Am-1$. Further Am privately proves for each triple $\langle u, v, w\rangle$ that the decommitment values are consistent, more precisely, that the randomness s and vote t in u are the same values encrypted in v and w. This can be shown by proving knowledge of

   (a) permutation π_{Am}

   (b) the value s_{Am-1} used to change the decommitment value in u_{Am-1} and v_{Am-1}

   (c) the randomness used to re-encrypt v_{Am-1} and w_{Am-1}.

   A more detailed description can be found in [14].

3. After the tallying process the Helios server provides a proof that the votes have been encrypted correctly.

   (a) Everyone can verify that the votes were decoded correctly because the "blinding" factors are revealed.

5.4 Improvements for booth voting

The implementation of Pedersen commitments for Helios showed that just minor changes on an eVoting system using homomorphic public key cryptography are necessary to provide everlasting privacy towards the public. Especially for legally binding elections this property is very important. The voters can elect their representatives free and without any influence by others if they believe that their vote remains secret. Knowing that the voting decision will be disclosed one day might affect the voter's behavior and the election outcome.

Thus, these results are also of interest for legally binding elections held in polling stations. By doing just small changes on the ballot generation, authentication, and tallying process everlasting privacy can be implemented also for electronic voting systems like Prêt à Voter [30], Scratch & Vote [4], and MarkPledge [2]. However before proposing such a solution it has to be evaluated how the encrypted decommitment values are stored and managed in the back end and which information is printed on the ballot paper. However, we believe that elaborating such

improvements are reasonable and plan to work on this in the future.

Conclusion

In the voting literature there are many protocols where the tallying is based on homomorphic encryption, leading to computational privacy. In this paper we showed that by using an unconditional homomorphic commitment combined with homomorphic encryption over a private channel, it is possible to enhance the CGS protocol used by Helios to obtain everlasting privacy. This transformation is general; it has already been applied to obtain mix networks with everlasting privacy towards the public, which, again, has applications to voting, as we showed for Helios in particular.

Though we recognize that still a lot of work needs to be done, especially with respect to efficiency, we think that by applying our ideas it might be possible to transform other voting protocols, and we strongly hope it represents the beginning of a paradigm shift towards everlasting privacy for voting protocols in general.

Acknowledgement

This work is a direct result of the second author's visit to Dagstuhl and Darmstadt in July 2011, and he would like to express his gratitude to the Dagstuhl Institute, to the Interdisciplinary Centre for Security, Reliability and Trust at the University of Luxemburg, to Centre for Advanced Security Research at the University of Darmstadt, and to his hosts, Peter Ryan and Johannes Buchmann, for making this visit possible. He also would like to thank Tal Moran and Olivier Pereira for interesting discussions. His research was partially supported by a FAPEMIG grant, number APQ-02719-10. Further this paper has been developed within the project 'VerKonWa' — Verfassungskonforme Umsetzung von elektronischen Wahlen — which is funded by the Deutsche Forschungsgemeinschaft (DFG, German Science Foundation).

References

[1] ADIDA, B. Helios: Web-based open-audit voting. In *USENIX Security Symposium* (2008), pp. 335–348.

[2] ADIDA, B., AND NEFF, C. A. Ballot casting assurance. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop* (Berkeley, CA, USA, 2006), EVT'06, USENIX Association, pp. 7–7.

[3] ADIDA, B., PEREIRA, O., MARNEFFE, O. D., AND JACQUES QUISQUATER, J. Electing a university president using open-audit voting: Analysis of real-world use of helios. In *In Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE* (2009).

[4] ADIDA, B., AND RIVEST, R. L. Scratch & vote: self-contained paper-based cryptographic voting. In *WPES* (2006), A. Juels and M. Winslett, Eds., ACM, pp. 29–40.

[5] BOS, J. *Practical Privacy*. PhD thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 1992. Available online: http://alexandria.tue.nl/extra3/proefschrift/PRF8A/9201032.pdf.

[6] BOS, J., AND PURDY, G. A voting scheme. Rump session of CRYPTO 88. Does not appear in proceedings.

[7] BROADBENT, A., AND TAPP, A. Information-theoretically secure voting without an honest majority. *CoRR abs/0806.1931* (2008).

[8] CHAUM, D., DAMGÅRD, I., AND VAN DE GRAAF, J. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO* (1987), pp. 87–119.

[9] CHAUM, D., ESSEX, A., CARBACK, R., CLARK, J., POPOVENIUC, S., SHERMAN, A. T., AND VORA, P. L. Scantegrity: End-to-end voter-verifiable optical-scan voting. *IEEE Security & Privacy 6*, 3 (2008), 40–46.

[10] CORTIER, V., AND SMYTH, B. Attacking and fixing helios: An analysis of ballot secrecy. In *CSF* (2011), pp. 297–311.

[11] CRAMER, R., FRANKLIN, M., SCHOENMAKERS, B., AND YUNG, M. Multi-authority secret-ballot elections with linear work. *Lecture Notes in Computer Science 1070* (1996), 72–81.

[12] CRAMER, R., GENNARO, R., AND SCHOENMAKERS, B. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology - EUROCRYPT 97* (1997), vol. 1233 of *LNCS*, Springer-Verlag, pp. 103–118.

[13] DAMGÅRD, I., AND KOPROWSKI, M. Practical threshold rsa signatures without a trusted dealer. In *EUROCRYPT* (2001), B. Pfitzmann, Ed., vol. 2045 of *Lecture Notes in Computer Science*, Springer, pp. 152–165.

[14] DEMIREL, D., AND VAN DE GRAAF, J. A publicly-verifiable mix-net with everlasting privacy towards observers. unpublished, April 2012.

[15] FIAT, A., AND SHAMIR, A. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO* (1986), A. M. Odlyzko, Ed., vol. 263 of *Lecture Notes in Computer Science*, Springer, pp. 186–194.

[16] FOUQUE, P.-A., POUPARD, G., AND STERN, J. Sharing decryption in the context of voting or lotteries. In *Financial Cryptography* (2000), Y. Frankel, Ed., vol. 1962 of *Lecture Notes in Computer Science*, Springer, pp. 90–104.

[17] FOUQUE, P.-A., AND STERN, J. Fully distributed threshold rsa under standard assumptions. In *ASIACRYPT* (2001), C. Boyd, Ed., vol. 2248 of *Lecture Notes in Computer Science*, Springer, pp. 310–330.

[18] GENNARO, R., JARECKI, S., KRAWCZYK, H., AND RABIN, T. Secure applications of pedersen's distributed key generation protocol. In *CT-RSA* (2003), M. Joye, Ed., vol. 2612 of *Lecture Notes in Computer Science*, Springer, pp. 373–390.

[19] GENNARO, R., JARECKI, S., KRAWCZYK, H., AND RABIN, T. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology 20*, 1 (2007), 51–83.

[20] HELIOS. Helios voting system.

[21] HELIOS. Helios deployed at princeton, 2009.

[22] HOSP, B., AND POPOVENUIC, S. Punchscan Voting Summary. Version dated Feb 13, 2006, obtained from first author, 2006.

[23] MORAN, T., AND NAOR, M. Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. CRYPTO 2006, 2006.

[24] MORAN, T., AND NAOR, M. Split-ballot voting: everlasting privacy with distributed trust. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security – CCS 2007* (2007), pp. 246–255.

[25] MORAN, T., AND NAOR, M. Split-ballot voting: Everlasting privacy with distributed trust. *ACM Trans. Inf. Syst. Secur. 13*, 2 (2010).

[26] PEDERSEN, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPT 91* (August 1991), vol. 547 of *LNCS*, Springer-Verlag.

[27] PEDERSEN, T. P. A threshold cryptosystem without a trusted party (extended abstract). In *EUROCRYPT* (1991), D. W. Davies, Ed., vol. 547 of *Lecture Notes in Computer Science*, Springer, pp. 522–526.

[28] PICHLER, F., Ed. *Advances in Cryptology - EUROCRYPT '85, Workshop on the Theory and Application of of Cryptographic Techniques, Linz, Austria, April 1985, Proceedings* (1986), vol. 219 of *Lecture Notes in Computer Science*, Springer.

[29] POPOVENUIC, S., AND HOSP, B. An Introduction to Punchscan. Version dated Oct 15, 2006. http://punchscan.org/papers/popoveniuc_hosp_punchscan_introduction.pdf, 2006.

[30] RYAN, P. Y. A., BISMARK, D., HEATHER, J., SCHNEIDER, S., AND XIA, Z. Prêt à voter: a voter-verifiable voting system. *IEEE Transactions on Information Forensics and Security 4*, 4 (2009), 662–673.

[31] SAKO, K., AND KILIAN, J. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *EUROCRYPT* (1995), L. C. Guillou and J.-J. Quisquater, Eds., vol. 921 of *Lecture Notes in Computer Science*, Springer, pp. 393–403.

[32] VAN DE GRAAF, J. Voting with unconditional privacy by merging prêt à voter and punchscan. *IEEE Transactions on Information Forensics and Security 4*, 4 (2009), 674–684.

[33] VAN DE GRAAF, J. Anonymous one-time broadcast using non-interactive dining cryptographer nets with applications to voting. In *Towards Trustworthy Elections* (2010), D. Chaum, M. Jakobsson, R. L. Rivest, P. Y. A. Ryan, J. Benaloh, M. Kutylowski, and B. Adida, Eds., vol. 6000 of *Lecture Notes in Computer Science*, Springer, pp. 231–241.

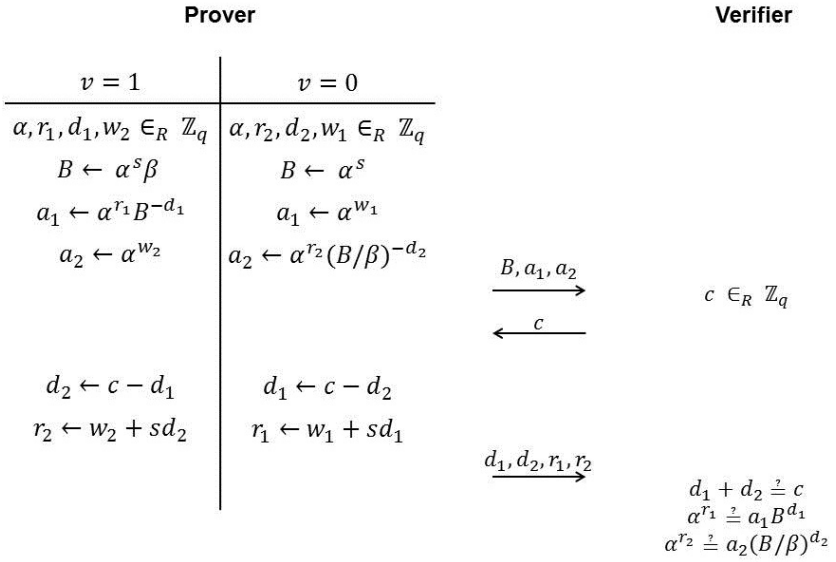

Figure 4: Proof of Validity of Ciphertext B; Compare to [11]