Countering Ballot Stuffing and Incorporating Eligibility Verifiability in Helios

Sriramkrishnan Srinivasan, Chris Culnane[1], James Heather[2], Steve Schneider[1], and Zhe Xia[3,⋆]

[1] Department of Computing, University of Surrey, Guildford GU2 7XH, U.K.
{c.culnane,s.schneider}@surrey.ac.uk
[2] Chiastic Security Ltd
james@chiastic-security.co.uk
[3] Department of Computing, Wuhan University of Technology, 430063, China
xiazhe@whut.edu.cn

Abstract. Helios is a web-based end-to-end verifiable electronic voting system which has been said to be suitable for low-coercion environments. Although many Internet voting schemes have been proposed in the literature, Helios stands out for its real world relevance. It has been used in a number of elections in university campuses around the world and it has also been used recently by the IACR to elect its board members. It was noted that a dishonest server in Helios can stuff ballots and this seems to limit the claims of end-to-end verifiability of the system. In this work, we investigate how the issue of ballot stuffing can be addressed with minimum change to the current vote casting experience in Helios and we argue formally about the security of our techniques. Our ideas are intuitive and general enough to be applied in the context of other Internet voting scheme and they also address recent attacks exploiting the malleability of ballots in Helios.

Keywords: End-to-End Verifiable Voting, Helios, Token-Controlled Encryption.

1 Introduction

A number of end-to-end (e2e) verifiable voting schemes have been proposed in the literature. It is argued that these schemes provide a number of verifiability properties.

- **Individual Verifiability:** the property whereby a voter can check that his vote has been correctly included in the tally by checking that it has been published on the election's bulletin board (BB).
- **Universal Verifiability:** the property whereby anyone can check that the outcome of the election corresponds to the list of published ballots.

⋆ Corresponding author.

Eligibility Verifiability is a less talked about property of e2e voting systems. This is the property whereby anyone can check that the published ballots have been cast by eligible voters and that only one ballot is tallied for each eligible voter. We argue this is of primary importance as eligibility verifiability (EV) ensures that ballot stuffing attacks are thwarted. Another property of end-to-end verifiable voting schemes is *ballot independence* whereby observing a voter's interaction with an election system should not allow an adversary to cast a vote that is somehow related.

To our knowledge, only a few Internet voting schemes have explicitly incorporated EV and provided defenses against ballot stuffing, e.g. JCJ/Civitas [10,5] and Cobra [7]. These schemes are also coercion-resistant i.e. a voter can appear to conform to the orders of an adversary while voting as desired. In JCJ/Civitas, these properties are achieved with the use of asymmetric key pairs for every voter, individual voter credentials and a framework for managing real and fake credentials. It can be argued, that although theoretically the gold standards, schemes like JCJ/Civitas put too much stress on intricate voter knowledge of the system and for most voters in the real world, the schemes are unusable.

Helios [1,2] takes the approach of catering to elections in low-coercion environments and aims to achieve usability. Helios stands out for its real world relevance and it has been used in a number of elections in university campuses around the world (see for example [2]) and it has also been used in the IACR presidential elections since 2010 [9]. Our work aims to address what we perceive to be some shortcomings in the Helios protocol, in particular the lack of eligibility verifiability. A dishonest Helios server can subvert the system by stuffing ballots i.e. submitting votes for voters who have not done so themselves [12]. A number of recent attacks [6,11] have also highlighted the fragility of the complex ballot structure in Helios and our work also addresses these attacks that exploit the malleability of the ballots in Helios. Our work incorporates eligibility verifiability in Helios, without the need for asymmetric keys-pairs for voters and ensures that the complexity of the scheme, in particular the voter experience, remains as close to the current experience as possible and the techniques are general enough to be applied to other internet voting schemes.

1.1 Structure of This Paper

The rest of the paper is structured as follows. In Section 2, we describe a number of possible methods to incorporate eligibility verifiability in Helios. Our main solution in Section 3 makes novel use of a primitive known as Token Controlled Encryption (TCE) [3] and in Section 4 we describe TCE and security models for TCE with the goal of arguing formally about the security of our proposed construction. We conclude with some comments in Section 5.

2 Incorporating Eligibility Verifiability in Helios

The basic functionality we would like to achieve, to ensure EV is that ballots are somehow tied uniquely to voters and we want to ensure that the various parties running the election are not able to stuff ballots. Anyone can then verify that ballots are not only tied to legitimate voters but also that they were cast by legitimate voters themselves and no one else.

To ensure this functionality, it seems necessary to assume some kind of private voter credential. However, we would like the use of this credential to be as less onerous on the voter as far as possible. In particular, we do not want voters to be able to manage and manipulate asymmetric keys. We would like to adhere to Benaloh's Ballot Casting Assurance principle [4]. In particular, we would like to ensure that although we use a private voter credential, no voter specific information is signaled to or used by the ballot creation device which can be used to influence its behavior. These requirements rule out the "obvious" solution of voters signing their existing Helios ballots before posting to the WBB. If a PKI for voter keys can be assumed to exist and if we assume that voters are able to use and manipulate these keys, it is unclear what benefits a protocol like Helios offers over a much more involved protocol like JCJ/Civitas.

We will assume for the remainder of the discussion that the party initiating the election supplies individual voters with a private one-time voter credential prior to the voting day. As an example, in the case of the IACR election, this is the IACR election management committee rather than the Helios server itself. In addition, voter authentication to the Helios server is leveraged using some existing system. We will call the one-time credential a token for reasons that will be described shortly and we will also assume that the party generating the one-time tokens is not the same party that manages user-authentication to the Helios server, or the Helios server itself. We note that such natural separation between the various entities already exists in the existing Helios use case scenarios. Thus, there are three distinct entities performing distinct functions and we assume that they do not collude.

We want to ensure that the Helios server and the service providing user authentication to the Helios server are both unable to stuff ballots. In addition, we also want security against the party generating the voter credentials and we want to ensure that this party is not able to violate the privacy of voters or stuff ballots itself.

In the remainder of this section, we will take a brief detour to describe the features of a novel primitive called Token Controlled Encryption that we will employ. Subsequently, we will describe a modified Helios work flow that will highlight the new setup process as well as the voter's view of the system.

Token-controlled public key encryption (TCE), introduced by Baek, Safavi-Naini and Susilo [3] is a cryptographic primitive in which senders encrypt messages with the public keys of receivers and a secret *token*. The token is delegated to a third-party, who is entrusted with releasing the token to the receiver when some conditions elaborated by the sender are satisfied. Receivers cannot decrypt encrypted messages until they obtain the corresponding token and the third-party who is entrusted with the token is unable to obtain any information about the original message. In addition, the communication and computational overheads for releasing the token are small.

2.1 Token Controlled Encryption

A TCE scheme is defined in terms of four probabilistic polynomial time (PPT) algorithms:

- `TCE.KeyGen`: A randomized key generation algorithm that on input a security parameter 1^k, outputs a public key pk and a matching private key sk. Associated to each public/private key pair are a message space MsgSp, a ciphertext space CtSp and a token space TokSp.
- `TCE.TokenGen`: A randomized token generation algorithm that on input a security parameter 1^k, outputs a random token $tok \in \mathsf{TokSp}$.
- `TCE.Enc`: A randomized encryption algorithm that on input pk, $tok \in \mathsf{TokSp}$ and message $m \in \mathsf{MsgSp}$ returns a ciphertext $c \in \mathsf{CtSp}$.
- `TCE.Dec`: A decryption algorithm that on input a secret key sk, a token $tok \in \mathsf{TokSp}$ and a ciphertext $c \in \mathsf{CtSp}$, returns either a message $m \in \mathsf{MsgSp}$ or a failure symbol $\perp$.

These algorithms must satisfy the consistency requirement that decryption with the appropriate token undoes encryption: i.e. $\forall (pk, sk) \leftarrow \mathtt{TCE.KeyGen}(1^k)$, $\forall tok \in \mathsf{TokSp}$, $\forall m \in \mathsf{MsgSp}$, $\forall c = \mathtt{Enc}(pk, tok, m)$, $\mathtt{Dec}(sk, tok, c) = m$.

Note in particular that the token generation algorithm does not require any inputs other than the security parameter.

To motivate the use of Token Controlled Encryption we describe briefly an application known as the Millionaire's will problem. A Millionaire wishes to write his will and entrust it to his sons, but he does not want them to be able to read the will till his demise. In the traditional setting, the will is entrusted to a lawyer, but our millionaire does not wish to reveal the contents of the will to his lawyer. A solution to this problem is proposed that uses a TCE scheme. The sons generate key pairs for a TCE scheme and publish the public key. The millionaire writes a will for his sons, encrypts it using the public key of the TCE scheme and gives the encrypted will to his sons. He entrusts the token to his lawyer and trusts that the lawyer will only reveal the token at a pre-appointed time. The lawyer does not have access to the private keys used to create the encrypted will and therefore is unable to obtain any information about the contents of the will. The sons do not have the token till the appointed time and therefore cannot decrypt the encrypted will. We will use TCE along the lines of the example above, with some tweaks, to achieve EV in Helios.

2.2 The Modified Helios Workflow to Incorporate Eligibility Verifiability

In the following section we describe a modified Helios workflow to incorporate EV in Helios. Key differences in the Setup and Voting process from the regular workflow are highlighted in **bold**. The Tallying phase now needs a few additional steps at the start.

Setup

- An election officer and a set of trustees are selected. The election officer is in charge of running the election server and bulletin board. We will refer to the election officer as the Helios server in subsequent discussions. The election officer publishes a set of common system parameters on the BB and the trustees jointly generate an asymmetric key pair i.e. the joint public key is generated by combining each trustee's public key share. The share of the private key of each trustee is kept secret.
- Each trustee's public key is published on a BB along with a proof that the respective trustees know the corresponding private key share. The joint public key is also published with a proof of correct construction.
- **The Helios server also generates an election specific key pair for a TCE scheme and publishes the public key for the TCE scheme, and a proof of knowledge of the private key on the BB. The Helios server keeps the private key for the TCE scheme secret.**
- The election officer publishes a candidate list.
- A hash that includes all the above information, called the election fingerprint is published on the BB.
- The election officer also publishes a list of eligible voters. We note that there is an implicit assumption here that the list of eligible voters is somehow made available to the election officer.
- **The party in charge of generating the list of eligible candidates also sends to the individual voters a one-time election specific token. It commits to a list of tokens against names of eligible voters; it can do so by supplying a cryptographic hash of a file in which the voter names and tokens are stored and the Helios server can publish this on the BB. The token list is kept secret.**

Voting Process

- The voter launches a browser script which downloads the global parameters published by the election officer. The script recomputes the election fingerprint and the voter should check that the fingerprint corresponds to the value published on the BB.
- The voter inputs his vote to the browser script. The script encrypts the vote with the joint public key and also generates a proof that the encrypted vote is constructed correctly i.e. only permitted values are encrypted. In the discussions that follow, we will call this the regular Helios ballot.
- **The voter inputs a token. This may either be the "real" token that is sent to the voter or a "fake" token.**
- **The regular Helios ballot is now encrypted using the public key of the TCE scheme.**
- The encrypted ballot is displayed to the voter.
- The voter may choose to either audit the ballot or cast it.

  - If the voter decides to audit the encrypted ballot then the script provides her with the random data used to create the ballot. The voter should then take this random data and re-create the ballot independently (ideally on another platform) to verify that it was indeed well formed. Audited votes cannot be cast and the browser script will prompt the voter to re-create an encrypted vote.
  - If the voter decides to cast the ballot, she submits her login credentials to the script which submits the ballot to the election officer. The election officer authenticates the voter, and posts the ballot to the election BB next to the voter's identity. The script deletes the randomness used to create the ballot.
- Voters can check the BB to be assured that their ballots appear on the bulletin board unchanged.

Tallying. Some extra steps are introduced in the tallying phase and these are outlined below.

- When the voting phase is over, the token list is revealed. The cryptographic hash can be recomputed and compared to the hash committed to on the BB and this ensures that the token list is the same committed to at the start of the election.
- The Helios server publicly reveals the election specific TCE private key.
- Every ballot against a voter's name is decrypted with the token revealed for that voter and the TCE private key. All ballots created with "fake" tokens will be automatically rejected as the decryption will fail. Where multiple ballots are created with a "real" token, a policy such as "consider last vote cast" can be used to select one voter per voter. Decryption reveals the regular Helios ballot.

All other aspects of tallying remain the same. In particular, proofs of all the ballots revealed from the process above are checked to ensure they are allowed encryptions. They are then tallied as normal.

2.3 Comments on the New Workflow

- The significant change in the Setup phase is that the Helios server now generates an election specific key-pair for a TCE scheme. This key-pair can be thresholded if greater assurance is required in its generation and perhaps to ensure that a key from an earlier election is not reused. Otherwise a regular key-pair suffices. The private key is revealed publicly during the tallying phase, so this TCE key-pair is truly election specific.
- The voting process in this new proposal still adheres to the principle of Ballot Casting Assurance. Although a voter token is used in the ballot construction phase, it is not tied to a voter's identity.
  Note that in the spirit of TCE, the token is small, so we can safely assume that the token can be represented in human understandable form and that the

voter is able to generate "fake" tokens in his head and input them easily to the ballot creation device. It is possible to envisage that the "real" token is sent as a QR code to the voter by mail or on a mobile handset, and that a separate application can generate "fake" tokens etc. It is also possible to have multiple tokens generated by more than one authority and these can be combined by the script during cote creation. This ensures that not too much trust is placed in one authority generating the tokens. We do not discuss these enhancements except to mention they are possible to keep the discussion simple.

– The voter is able to submit multiple ballots. This may include multiple ballots with "fake" tokens, but also multiple ballots with the "real" one. Ballots created with "fake" tokens will be automatically rejected, while a policy such as consider last vote submitted with "real" token" can be used to choose one vote per voter. The general spirit of this approach is in the lines of JCJ/Civitas. Our contribution lies in incorporating the ideas into the existing Helios workflow and ensuring a significantly simpler workflow for the voter without voter asymmetric keys.
– Encrypting the regular Helios ballot with the TCE public key ensures that individual ballot components of the regular Helios ballot are no longer available to those observing the BB during the election phase, rendering ballot malleability attacks infeasible. Replay attacks will also not work as ballots are bound to private user tokens. The only possibility of exploiting the individual ballot components is to somehow re-use them in a subsequent election after they are revealed in the tallying phase. This is clearly a more remote threat and one that can be easily mitigated by generating one time election specific keys as well. It is also possible to incorporating the tokens in the signature proofs of knowledge. This will enable public checkability of the proofs once the TCE decryption is completed, as tokens as publicly revealed while also ensuring that ballots are tied uniquely to election specific voter tokens.
– The audit process is unchanged and is still designed to assure the voter that the encrypted vote does in fact correspond to the choices made by the voter and cannot check attacks due to malware.
– In keeping with the spirit of Helios, we assume that some existing infrastructure is used to manage login of voters to the Helios server. Our only requirement is that this should not be same infrastructure that supplies the one-time tokens to the voters. In our construction, the Helios server itself can send out username and passwords to the voters by email and this does not enable it to stuff ballots. In the existing setup, the Helios server is able to stuff ballots whether it manages logins of voters to the system or not.
– If the infrastructure that generates the tokens also manages logins to the Helios server, then this infrastructure may be able to stuff ballots. It can be argued that this is a lesser threat than the possibility of the Helios server itself stuffing ballots and one that it is easily mitigated by leveraging login by an arbitrary existing infrastructure. On the other hand, in the existing Helios workflow, even when arbitrary infrastructure is leverages to supply login for voters, the Helios server itself may stuff ballots and it is not clear how this can be checked.

3 Security of Our Proposed Construction

We will introduce the security notions for TCE scheme more formally in the next section. Informally, we have the following types of adversaries and the following security requirements.

- Adversaries without knowledge of the private key and token. In our setting these correspond to general external adversaries i.e. other than the party generating the tokens, or the Helios server itself. In the context of TCE schemes, these are called Type-1 outside adversaries. We will require that ciphertexts are indistinguishable to these adversaries. This is captured in the security game TCE.1 in Section 4.1.
- Adversaries without knowledge of the private key, but with access to the token. In our setting this corresponds to the party generating the tokens. In the context of TCE schemes these are called Type-2 outside adversaries. We will require that ciphertexts are indistinguishable to these adversaries. This is captured in the security game TCE.2 in Section 4.2.
  As discussed earlier, this type of adversary is not able to stuff ballots, under the assumption that they do not control the infrastructure that supplies login credentials. This is because, with knowledge of the token and public key, this adversary can create valid ballots for a particular voter. As briefly discussed earlier, it is also possible to have the tokens generated by more than one party and delivered to the voter who will supply all tokens to the browser script. This prevents a single token generating entity from being able to stuff ballots. In this second scenario, a token generating party may also supply the authentication infrastructure.
- Adversaries with knowledge of the private key, but without access to the token. In our setting, this corresponds to the Helios server itself. In the context of TCE schemes, these are called inside adversaries. We require that ciphertexts are indistinguishable to these adversaries. This is captured in the security game TCE.3 in Section 4.3.
- Finally, to prevent ballot stuffing by the Helios sever itself, we require the much stronger security property that the the inside adversary against the TCE scheme is not able to make ciphertexts created under one token decrypt successfully under another token. This is captured in the security game TCE.4 in Section 4.4.

To summarize, assuming that the system is instantiated with a concrete TCE scheme meeting the appropriate security notions outlined above our construction satisfies the following properties.

- The Helios server, despite possessing the private key that corresponds to the public key used to encrypt the regular Helios ballot, is unable gain any information till the token is made available. Although the regular Helios ballot is already encrypted, the new procedure ensures that the individual ciphertext components of the regular Helios ballot are not available to anyone before the tallying phase, including the Helios server.

- The Helios server is also unable to generate valid ciphertexts without knowledge of the voter tokens and this ensures that ballot stuffing attacks are rendered infeasible as all ballots generated without the legitimate voter token are automatically rejected.
- Even when the private key for the TCE scheme and voter tokens are made public, the public decryption only reveals the regular Helios ballots, ensuring the same privacy and integrity guarantees as in regular Helios.

4 Security Notions for Token Controlled Encryption

Baek, Safavi-Naini and Susilo [3] first proposed security models for TCE schemes. Galindo and Herranz [8] identified the security models considered in [3] as being inadequate as the issue of a single ciphertext decrypting to different messages under different tokens is not addressed. They introduced an improved security model termed the *strong existential token unforgeability* model and proposed a generic construction starting from any trapdoor partial one-way function and secure in the ROM. Below we give the various security notions for TCE schemes and in Section 4.5 we talk briefly about concrete instantiations, but leave the details to the original sources.

As mentioned earlier, two types of adversaries can be considered against TCE schemes. "Outside" adversaries who do not have access to the private key and "Inside" adversaries, who possess the receiver's private key (i.e. the receiver's themselves). In addition, Outside adversaries are further categorized as Type 1 Outside adversaries, who do not possess the token and Type 2 Outside adversaries who have access to the token (the semi-trusted third party) respectively.

We will capture security against these different types of adversaries in terms of various security games that the adversary plays against a challenger $\mathcal{C}$. The adversary will be given access to one or more of the following oracles with suitable restrictions on their use.

- A token embedded encryption oracle: The adversary can query the token embedded encryption oracle with a message m and receives the ciphertext c created with the target token tok^* and the public key of the receiver i.e. $c = \mathtt{Enc}(pk, tok^*, m)$.
- A decryption oracle: The adversary can query the decryption oracle with a ciphertext/token pair and receives back the decryption of the ciphertext under the specified token and secret key of the receiver. i.e $m = \mathtt{Dec}(sk, tok, c)$. Here m is either a valid plaintext or $\perp$ to indicate failure of decryption.

4.1 TCE.1: Security Against Type-1 Outside Adversaries

We consider the following game between the adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, which captures security against an outside adversary $\mathcal{A}$ whose goal is to break the confidentiality of the TCE ciphertext created with a fixed token. The adversary $\mathcal{A}$ does not have access to the private key or to the token used to encrypt the message.

The challenger $\mathcal{C}$ takes as input the security parameter 1^k and runs algorithm $\texttt{TCE.KeyGen}$ of the TCE scheme to obtain (pk, sk) i.e. $(pk, sk) \leftarrow \texttt{TCE.KeyGen}(1^k)$. The challenger $\mathcal{C}$ also runs algorithm $\texttt{TCE.TokenGen}$ to obtain a target token tok^* i.e. $tok^* \leftarrow \texttt{TCE.TokenGen}(1^k)$.

The adversary is given the public keys pk but not the corresponding secret key sk or the target token tok^*. The game proceeds as follows.

– **Phase 1:** $\mathcal{A}$ issues a series of adaptively selected encryption queries on messages m to the token embedded encryption oracle. $\mathcal{A}$ also issues a series of adaptively selected decryption queries on ciphertext/token pairs (c, tok) to the decryption oracle.
– **Challenge:** After $\mathcal{A}$ decides to end Phase 1, it outputs two equal length messages m_0 and m_1. $\mathcal{C}$ selects $b \overset{\$}{\leftarrow} \{0, 1\}$, sets $c^* = \texttt{Enc}(pk, tok^*, m_b)$ and gives c^* to $\mathcal{A}$.
– **Phase 2:** This phase proceeds as in Phase 1.
– **Guess:** $\mathcal{A}$ outputs a bit b' and wins the game if $b' = b$.

The advantage of $\mathcal{A}$ against the TCE scheme in the above TCE.1 security game is defined to be:

$$\mathbf{Adv}_{\mathcal{A}}^{\text{TCE.1}}(k) = |\Pr[b' = b] - 1/2|$$

where the probability is measured over the random choices of coins of $\mathcal{A}$ and $\mathcal{C}$.

A scheme is said to be TCE.1 secure if the advantage of all PPT adversaries is negligible as a function of the security parameter k.

No restriction is placed on the decryption queries in Phase 1. This is because the token is not known to the adversary and it is argued that the query on (c^*, tok^*) must occur with negligible probability, implying that the token must be chosen from an exponentially large space.

4.2 TCE.2: Security Against Type-2 Outside Adversaries

This is very similar to security against Type-1 outside adversaries except that now the adversary also has access to the target token. Thus, the adversary can be thought to be the semi-trusted third party entrusted with the token, who attempts to break the confidentiality of TCE ciphertexts. Security against Type-2 adversaries is captured in terms of a game very similar to the security game stated above with the restriction that the adversary cannot query the decryption oracle with the challenge ciphertext and target token.

The challenger $\mathcal{C}$ takes as input the security parameter 1^k and runs algorithm $\texttt{TCE.KeyGen}$ of the TCE scheme to obtain (pk, sk) i.e. $(pk, sk) \leftarrow \texttt{TCE.KeyGen}(1^k)$. The challenger $\mathcal{C}$ also runs algorithm $\texttt{TCE.TokenGen}$ to obtain a target token tok^* i.e. $tok^* \leftarrow \texttt{TCE.TokenGen}(1^k)$.

The adversary is given the target token tok^* and the public key pk but not the corresponding secret key sk. The game proceeds as follows.

- **Phase 1:** $\mathcal{A}$ issues a series of adaptively selected encryption queries on messages m to the token-embedded encryption oracle . $\mathcal{A}$ also issues a series of adaptively selected decryption queries on ciphertext-token pairs (c, tok) to the decryption oracle.
- **Challenge:** After $\mathcal{A}$ decides to end Phase 1, it outputs two equal-length messages m_0 and m_1. $\mathcal{C}$ selects $b \overset{\$}{\leftarrow} \{0,1\}$, sets $c^* = \texttt{Enc}(pk, tok^*, m_b)$ and gives c^* to $\mathcal{A}$.
- **Phase 2:** This phase proceeds as in Phase 1 with the restriction that the adversary may not query the decryption oracle with (c^*, tok^*).
- **Guess:** $\mathcal{A}$ outputs a bit b' and wins the game if $b' = b$.

The advantage of $\mathcal{A}$ against the TCE scheme in the TCE.2 security game is defined to be:

$$\mathbf{Adv}_{\mathcal{A}}^{\text{TCE.2}}(k) = |\Pr[b' = b] - 1/2|$$

where the probability is measured over the random choices of coins of $\mathcal{A}$ and $\mathcal{C}$.

A scheme is said to be TCE.2 secure if the advantage of all PPT adversaries is negligible as a function of the security parameter k.

4.3 TCE.3: Security against Inside Adversaries

Insider adversaries are those who possess the appropriate private keys but not the target token and attempt to break the confidentiality of TCE ciphertexts i.e. they are receivers of TCE ciphertexts to whom the token has not yet been released by the semi-trusted third party.

The challenger $\mathcal{C}$ takes as input the security parameter 1^k and runs algorithm $\texttt{TCE.KeyGen}$ of the TCE scheme to obtain (pk, sk) i.e. $(pk, sk) \leftarrow \texttt{TCE.KeyGen}(1^k)$. The challenger $\mathcal{C}$ also runs algorithm $\texttt{TCE.TokenGen}$ to obtain a target token tok^* i.e. $tok^* \leftarrow \texttt{TCE.TokenGen}(1^k)$.

The adversary is given the public key pk and corresponding secret key sk but not the target token tok^*. The game proceeds as follows.

- **Phase 1**: $\mathcal{A}$ issues a series of adaptively selected encryption queries on messages m to the token embedded encryption oracle.
- **Challenge:** After $\mathcal{A}$ decides to end Phase 1, it outputs two equal length messages m_0 and m_1. $\mathcal{C}$ selects $b \overset{\$}{\leftarrow} \{0,1\}$, sets $c^* = \texttt{Enc}(pk, tok^*, m_b)$ and gives c^* to $\mathcal{A}$.
- **Phase 2**: This phase proceeds as in Phase 1.
- **Guess**: $\mathcal{A}$ outputs a bit b' and wins the game if $b' = b$.

The advantage of $\mathcal{A}$ against the TCE scheme in the above TCE.3 security game is defined to be:

$$\mathbf{Adv}_{\mathcal{A}}^{\text{TCE.3}}(k) = |\Pr[b' = b] - 1/2|$$

where the probability is measured over the random choices of coins of $\mathcal{A}$ and $\mathcal{C}$.

A scheme is said to be TCE.3 secure if the advantage of all PPT adversaries is negligible as a function of the security parameter k.

4.4 TCE.4: Strong Existential Token Unforgeability

In the original works by [3], the issue of a ciphertext created with one token decrypting correctly with another token was not considered. A new security notion termed the "Strong Existential Token Unforgeability" capturing this requirement was first introduced in [8].

The challenger $\mathcal{C}$ takes as input the security parameter 1^k and runs algorithm $\texttt{TCE.KeyGen}$ of the TCE scheme to obtain (pk, sk) i.e. $(pk, sk) \leftarrow \texttt{TCE.KeyGen}(1^k)$.

The adversary is given the public key pk but not the corresponding secret key sk. The game proceeds as follows.

- **1**: $\mathcal{A}$ issues a series of decryption queries on adaptively selected ciphertext/token pairs (c, tok) to the decryption oracle.
- **2**: $\mathcal{A}$ outputs two different tokens tok_1 and tok_2 and a ciphertext c. The adversary wins the game if $\texttt{Dec}(sk, tok_1, c) = m_1$ and $\texttt{Dec}(sk, tok_2, c) = m_2$ with $m_1 \neq m_2 \neq \perp$.

The advantage of $\mathcal{A}$ against the TCE scheme in the above TCE.4 security game is defined to be:

$$\mathbf{Adv}_{\mathcal{A}}^{\text{TCE.4}}(k) = |\Pr[\texttt{Dec}(sk, tok_1{}^*, c) = m_1 \wedge m_2 = \texttt{Dec}(sk, tok_2{}^*, c) \wedge m_1 \neq m_2 \neq \perp]|$$

where the probability is measured over the random choices of coins of $\mathcal{A}$ and $\mathcal{C}$.

A scheme is said to be TCE.4 secure if the advantage of all PPT adversaries is negligible as a function of the security parameter k.

This is a very strong but necessary security requirement. Here we ask that it should be infeasible for an adversary to make up a new token after the challenge ciphertext is received, and that this should be the case even if the challenge ciphertext is created by the adversary.

4.5 Constructions of TCE Schemes

As observed in [3] a somewhat intuitive construction for TCE scheme may proceed as follows. Let (pk, sk) be the keys of a PKE scheme with encryption algorithm E. Let E' be the encryption algorithm of a symmetric key scheme. Then, to encrypt a message m to a receiver with public key pk, choose a key k for the symmetric scheme at random and send ciphertext $\texttt{Enc}(pk, \texttt{Enc}'(k, m))$. The receiver is able to decrypt the ciphertext with his secret key sk but cannot obtain the message till he obtains the symmetric key k.

Intuitively, this basic approach achieves the functionality we desire to achieve EV in Helios. However, the authors in [3] observe that for this scheme to be proven secure the symmetric encryption scheme must be IND-CCA secure and it is not straightforward to analyze the security of the construction.

Two specific instantiations for TCE schemes are proposed in [3] which are TCE.1, TCE.2 and TCE.3 secure in the Random Oracle Model. However, it is shown in [8] that these schemes are not TCE.4 secure. A generic construction for a TCE scheme starting from any trapdoor partial one-way function and secure in the ROM is proposed in [8].

5 Conclusion

We have discussed the lack of Eligibility Verifiability in Helios which leads to ballot stuffing attacks and proposed a solution to incorporate EV and counter ballot stuffing without significantly affecting the existing voter experience. Furthermore, the additional cryptography does not in any way impact the existing protocol and implementation and can be seamlessly incorporated to the existing code base. The proposed solution does not introduce any onerous asymmetric key management for voters as in schemes such as JCJ/Civitas which are the only schemes that incorporate EV but with much greater usability challenges for voters. We argue formally about the security of our new ideas which make novel use of a primitive known as Token Controlled Encryption. Our ideas make minimal change to the existing voter experience in Helios and in particular, the principle of Ballot Casting Assurance is retained and in addition, attacks exploiting the knowledge and malleability of individual ciphertext components in Helios are thwarted.

Acknowledgement. Dr. Sriramkrishnan Srinivasan was at the University of Surrey when this work was carried out. This work was funded by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant EP/G025797/1, and we are grateful to the anonymous reviewers for their valuable comments on the paper.

References

1. Adida, B.: Helios Web-based open-audit voting. In: van Oorschot, P.C. (ed.) USENIX Security Symposium, pp. 335–348. USENIX Association (2008)
2. Adida, B., De Marneffe, O., Pereira, O., Quisquater, J.: Electing a university president using open-audit voting: analysis of real-world use of helios. In: Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections, p. 10. USENIX Association (2009)
3. Baek, J., Safavi-Naini, R., Susilo, W.: Token-controlled public key encryption. In: Deng, R.H., Bao, F., Pang, H., Zhou, J. (eds.) ISPEC 2005. LNCS, vol. 3439, pp. 386–397. Springer, Heidelberg (2005)
4. Benaloh, J.: Simple verifiable elections. In: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop, p. 5. USENIX Association (2006)
5. Clarkson, M.E., Chong, S., Myers, A.C.: Civitas: A secure remote voting system. In: Chaum, D., Kutylowski, M., Rivest, R.L., Ryan, P.Y.A. (eds.) Frontiers of Electronic Voting. Dagstuhl Seminar Proceedings, vol. 07311, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2007)

6. Cortier, V., Smyth, B.: Attacking and fixing helios: An analysis of ballot secrecy. In: CSF, pp. 297–311. IEEE Computer Society (2011)
7. Essex, A., Clark, J., Hengartner, U.: Cobra: toward concurrent ballot authorization for Internet voting. In: EVT 2012 (2012)
8. Galindo, D., Herranz, J.: A generic construction for token-controlled public key encryption. In: Di Crescenzo, G., Rubin, A. (eds.) FC 2006. LNCS, vol. 4107, pp. 177–190. Springer, Heidelberg (2006)
9. Haber, S., Benaloh, J., Halevi, S.: The helios e-voting demo for the iacr. International Association for Cryptologic Research (2010), `http://www.iacr.org/elections/eVoting/heliosDemo.pdf`
10. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Atluri, V., di Vimercati, S.D.C., Dingledine, R. (eds.) WPES, pp. 61–70. ACM (2005)
11. Smyth, B.: Replay attacks that violate ballot secrecy in helios. Cryptology ePrint Archive, Report 2012/185 (2012), `http://eprint.iacr.org/`
12. Smyth, B., Ryan, M., Kremer, S., Kourjieh, M.: Towards automatic analysis of election verifiability properties. In: Armando, A., Lowe, G. (eds.) ARSPA-WITS 2010. LNCS, vol. 6186, pp. 146–163. Springer, Heidelberg (2010)