To Du or not to Du: A Security Analysis of Du-Vote

Steve Kremer, Peter Rønne

To Du or not to Du: A Security Analysis of Du-Vote

Steve Kremer
INRIA Nancy - Grand'Est & Loria, France

Peter Rønne
INRIA Nancy - Grand'Est & Loria, France
SnT, University of Luxembourg

***Abstract*—Du-Vote is a recently presented remote electronic voting scheme. Its goal is to be malware tolerant, i.e., provide security even in the case where the platform used for voting has been compromised by dedicated malware. For this it uses an additional hardware token, similar to tokens distributed in the context of online banking. The token is software closed and does not have any communication means other than a numerical keyboard and a small display. Du-Vote aims at providing vote privacy as long as either the vote platform or the vote server is honest. For verifiability, the security guarantees are even higher, as even if the token's software has been changed, and the platform and the server are colluding, attempts to change the election outcome should be detected with high probability. In this paper we provide an extensive security analysis of Du-Vote and show several attacks on both privacy as well as verifiability. We also propose changes to the system that would avoid many of these attacks.**

1. Introduction

In recent years remote Internet voting has been used or trialled for legally binding elections in several countries. The first country to nationally offer the use of remote Internet voting was Estonia in 2005. In the recent March 2015 Estonian parliament elections over 30% of the votes were cast by Internet [1]. Also in March 2015, an even larger election (in terms of electronically cast votes) was the state election in New South Wales, Australia. In June 2012 French expatriates were allowed to vote via the Internet for parliament elections. The security of these systems has however been challenged. For instance, Halderman *et al.* highlighted security problems in the Estonian [2] and Australian [3] elections. In the latter a TLS bug was exploited, rather than a weakness in the actual protocol. In the 2012 French parliament elections a French expatriate, Laurent Grégoire, demonstrated that it was possible to write a malware able to change the value of a casted vote without any way for the voter to notice [4]. In the 2011 Estonian parliament election, a similar attack was reported by computer scientist P. Pihelgas who conducted a real life experiment with consenting test subjects [5].

The current situation motivated many academic works on secure e-voting. The key properties that were identified are vote privacy and end-to-end verifiability. Vote privacy ensures that the voting system should not release any information about your vote other than the information revealed by the final outcome (in an unanimous vote, of course, the outcome would leak your exact vote, independent of the system in use). This notion of privacy has been precisely defined in both symbolic [6], [7] and computational [8], [9] models for reasoning about cryptographic protocols. The second property is end-to-end verifiability which guarantees the integrity of the election and ensures its transparency by providing verifiable evidence of the accuracy of the result. End-to-end verifiability generally includes individual verifiability (each individual voter can check that his vote has been casted directly), as well as universal verifiability (any observer may verify that the election outcome corresponds to the tally of the casted votes). This also leads to the notion of software independence [10]: there is no need to trust the software used to conduct the election, as the election produces proofs that can be independently verified. However, when proving properties of protocols, the (sometimes implicit) assumption is that the client platform indeed executes correctly the expected protocol. This assumption is however not realistic since standard PCs are generally prone to malware attacks, which are becoming both more frequent and more sophisticated. In the context of remote e-voting one of the main problems is that the client software used for voting cannot be trusted: malware may divulge the vote or other supposedly secret values such as passwords, encryption randomisations, etc. or even simply alter it. The threat of dedicated malware for, e.g., changing votes is a realistic threat as it was demonstrated in the above discussed French parliament elections.

Ensuring privacy and verifiability when the voter's computing device is malicious has become one of the most challenging problems in remote e-voting. A first solution to ensure privacy was proposed by Chaum [11] who introduced the notion of *code voting*. Instead of entering the vote itself, the voter inputs a code, representing the vote, but which cannot be linked to the actual vote by the client platform. The scheme, however, does not offer verifiability. An improvement was proposed by Ryan and Teague in the Pretty Good Democracy [12] scheme. This scheme does offer verifiability, but the integrity of the election could be compromised in the case where code sheets are leaked. To ensure the integrity of the election, i.e., avoid that a malware would change the vote, Benaloh [13] proposed to

use a cut-and-choose technique to audit ballots: after having constructed the ballot, a voter can either cast it or decide to audit it in order to check that the vote has not been changed. This technique has for instance been implemented in the Helios system [14]. However, usability studies, see e.g. [15], suggest that most voters do not audit their ballots. Moreover, as most systems display the result of the audit on the same platform as used to compute the ballot, a sophisticated malware could avoid being detected.

Another promising direction to circumvent malware attacks is to rely on an external hardware device with a proper display which may be used to perform some computation. Such *hardware tokens* are already used by some banks to enhance the security of online transactions. Recently there have been some proposals using such tokens in the context of e-voting, e.g. [16]. The hardware token should take a particularly simple form such that it is hard to tamper with (compared to e.g. a smartphone). These tokens can easily be audited and, being offline, are less vulnerable to malware. In order to be effective, the token must not be connected to the computer and requires short strings to be copied by the human from the token to the computer or vice-versa. The recent Du-Vote [17] scheme, on which we will concentrate in this paper, is another attempt to create an e-voting protocol in which the devices used to vote are not considered trustworthy – hence the name (Devices that are Untrusted used to Vote).

Roughly, Du-Vote works as follows. The vote platform generates encryptions and corresponding short voting codes (e.g. 4 digits) for each candidate. The voter enters the code for the chosen candidate (and some additional information) into the token which computes a vote code to be submitted to the vote server. The additional information will be used to verify that the platform correctly generated the encryptions. From the vote code, the vote server, which shares a key with each of the tokens, is able to extract the encryption of voter's chosen candidate. The encrypted ballots can then be tallied using either mixnets or a homomorphic tally.

Our contributions. In this paper we will provide an intensive security analysis of Du-Vote. As Du-Vote aims to provide very strong security guarantees under low trust assumptions, the protocol takes a somewhat complicated form which opens up for attacks.

We present several attacks on privacy. We first show that a *vote replay attack* by a dishonest voter, similar to the attack presented by Cortier and Smyth [18] on Helios, is possible. At a first glance it is not obvious that the attack is feasible, but we note that it is sufficient to re-randomise the encryption to obtain a collision between the last digits of the encryption and one of the vote codes, which can be done efficiently. Privacy can also be broken as soon as the token is used twice. This has already been discussed in the original Du-Vote paper when they emphasise that, for this reason, a device must not be used for several elections. We here discuss how phishing might be used to trick a voter into using the device twice, and how the factory producing the devices (or the delivery service) may actually abuse its access to the token to get enough information to link a vote and a token a posteriori to voter IDs. This information may then be combined with a malware infecting the vote platforms to perform a large scale privacy breach. Finally, we also discuss how error messages, or their absence, can be used to, at least mildly, break privacy by excluding one or more possible candidates.

We also present several attacks against verifiability, and the integrity of the election. We first note that in its current form, *ballot stuffing* is possible. This problem might be considered orthogonal and a generic technique [19] to avoid this attack could be deployed. We however propose a slight modification to the token that would avoid deploying this additional technique. Next we present a severe attack in which the server may compute a *fake key* that allows the server to change the vote in an undetectable way. We propose a change to the zero-knowledge proof to avoid this attack. We also propose a *collision attack* where the platform and the server can collude to trigger frequent collisions among the output computed by the token for different candidate choices. This allows for a randomisation attack where the voter's choice can be modified to another random candidate. Next we propose attacks based on the fact that the platform may display fake vote codes to the voter. A first computationally very heavy attack allows the platform to compute and display a set of vote codes such that for any possible choice of the voter, the vote goes to the attacker's candidate of choice. While this attack is not feasible for a large number of candidates, we show that for 2 candidates the attack could be performed without much effort. We also show that when the server and the platform collude and the token is software modified, the voter's choice may be modified, according to the attacker's choice. Moreover, the software modification of the token would not be noticed on honest platforms, meaning that the attack is undetectable.

We summarise the attacks and necessary trust assumptions in Tables 4 and 5. Some of these attacks require to take very relaxed trust assumptions, e.g., the last attack which supposed that the platform, server and token were malicious. This nevertheless contradicts the strong security claims of Du-Vote [17] which we summarise in Tables 2 and 3.

Outline of the paper. We first give an overview of the Du-Vote scheme (Section 2) and then recall the trust assumptions and claimed security goals in Section 3. Then we present our attacks on privacy (Section 4) and verifiability (Section 5). We finally conclude in Section 6 and discuss what vulnerabilities are avoided after applying our proposed fixes and which ones are still present.

2. Short description of the protocol

We here present a brief description of the Du-vote protocol. For details we refer the reader to [17]. The parties involved in the protocol are the voting authority (A), the vote server (S), the voter (V), the computing platform for the voter (P), the hardware token for the voter (H), a bulletin board (BB) and a set of decryption tellers T. The tallying can be done using mixnets or via homomorphic encryption, but this is not important for the protocol. We also explicitly

add a delivery service for the hardware tokens (D) and a hardware token producer (HP) to the description of the protocol, as they are also taken into account in our analysis.

At a very high level, and from the voter's perspective, the protocol works as follows. The voter's platform, e.g. a laptop computer, displays a ballot that consists of 3 columns, as depicted in Table 1. The first column displays the candidates $a_1, \ldots, a_n$. The columns A and B associate to each candidate a_i two possible vote codes A_i^* and B_i^*. These vote codes are typically rather short, e.g. 4 decimal digits. First, V chooses randomly one column to be the vote column, and the other to be the audit column. In order to vote for candidate a_i, the voter types into the hardware device all the codes of the audit column, in the displayed order, followed by the code in the vote column that is associated to a_i. For example, if we assume 4 candidates and that the voter chose column B to be the audit column, she enters the sequence $B_1^*|B_2^*|B_3^*|B_4^*|A_2^*$ into H to vote for candidate a_2. The device outputs a short vote code that the voter enters into the platform. The platform transmits the vote code and encryptions of the candidates to the vote server, and the voter checks that the vote code, together with her voting ID and the ballot ID, are correctly displayed on the bulletin board. The bulletin board additionally displays evidence to be checked that will ensure the voter that the protocol has been followed correctly and that her vote has been casted correctly. The tallying can then be done using any standard verifiable method using, e.g., homomorphic tallying or a verifiable mixnet.

In the remainder of this section we will give a more detailed description of the protocol. An overview of the whole protocol in form of a message sequence chart is also depicted in Figure 1.

Preliminaries. The voting protocol uses a number of cryptographic tools which need to be settled. We assume a cyclic group, e.g. a Schnorr group of order q with multiplication modulo p and two generating elements g, h, for which nobody knows the log relation. The encryption used by Du-Vote is exponential El-Gamal and we denote by

$$\mathsf{enc\text{-}exp}(m; y; r) := (g^r, h^m y^r)$$

the encryption of message m with public key y and randomness r. We denote by $\mathsf{hash}(\cdot)$ the application of a cryptographic hash function and assume the existence of a pseudo-random generator. There is further a parameter κ and a corresponding mapping $\cdot^*$ which takes numbers, bitstrings or pairs of bitstrings into κ-digit numbers (simply taking the last κ digits of the corresponding integer maybe after concatenating).

2.1. Registration and preparation

Well ahead of the election, the decryption tellers T publicize a shared public key y. The voting authority determines all eligible voters. The eligible voters can then register and will get a password for the vote server and a voter ID. The voter IDs can be published to the bulletin board BB. The voting authority chooses randomly secret keys $K = y^k$ for each voter and an associated serial number and sends an order to the hardware producer HP for hardware tokens with these keys with the serial numbers printed in clear on them.[1] Then A informs the vote server S secretly about the correspondence between vote ID and secret keys $k = \log_y K$. A sends the correspondence between serial numbers and postal addresses for the voters to the delivery service D, and D then delivers the hardware tokens H to the voters. Finally, just before the beginning of the election, A announces the election nonce I which is computed pseudo-randomly in some predetermined, but unpredictable, way. Also the set of names of the n candidates

$$Cand = \{a_1, \ldots, a_n\}$$

is published on BB.

2.2. Ballot preparation by P

The voting can now proceed as follows. The voter logs into the voter server S via her computational platform P using the password and voter ID that were issued at the registration. P then produces encryptions of the candidate names *Cand* and displays a ballot that is computed as follows.

- First P computes a set of $2n$ *distinct* pseudo-random voting codes
$$\{c_1, \ldots, c_{2n}\}$$
in a way that is reproducible knowing the voter ID and the election nonce I. This is done using the pseudo-random generator seeded with $\mathsf{hash}(I, \text{voter ID})$ using possibly retakes to ensure that all c_i are distinct numbers. P now chooses the ordered set $\{\mathsf{code}_A(1), \ldots, \mathsf{code}_A(n)\}$ as a random cyclic shift of $\{c_1, \ldots, c_n\}$ and likewise $\{\mathsf{code}_B(1), \ldots, \mathsf{code}_B(n)\}$ as a random cyclic shift of $\{c_{n+1}, \ldots, c_{2n}\}$. These shifts are performed to ensure privacy.
- P computes encryptions
$$A_i = \mathsf{enc\text{-}exp}(a_i; y; r_i^A)$$
for $r_i^A = r_{\mathsf{start}}, r_{\mathsf{start}} + 1, \ldots$ until $A_i^* = \mathsf{code}_A(i)$. In the same way encryptions
$$B_i = \mathsf{enc\text{-}exp}(a_i; y; r_i^B)$$
are determined such that $B_i^* = \mathsf{code}_B(i)$. Recall that the *-operation is the truncation to the last κ digits after concatenating the two strings in the encryption and converting this into an integer.

1. Note that it is probably more realistic that the hardware producer HP simply produces these and sends the list of serial numbers and corresponding $k = \log_y K$ to A. However, if it is done the other way round there is a small advantage that HP does not know $\log_y K$. Even though knowing K is enough to launch serious attacks on the voting protocol, knowing the exponent k is even stronger, e.g. the zero-knowledge proofs of the vote server cannot be constructed without this knowledge.

- Next, P displays to the voter V the ballot shown in Table 1. The ballot ID is computed to be $\mathsf{hash}(A, B)$ where A and B are the lexicographically ordered $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$ respectively.

Candidate	Column A	Column B
a_1	A_1^*	B_1^*
a_2	A_2^*	B_2^*
a_3	A_3^*	B_3^*
a_4	A_4^*	B_4^*
Enter your vote code here		

Ballot ID: hash(A,B)

TABLE 1. THE CRYPTOGRAPHIC CONSTRUCTION OF THE BALLOT SHOWN BY P TO THE VOTER V.

2.3. Vote casting

Now the voter can cast her vote as follows.

- The voter V chooses randomly either column A or B as an audit column. We call the other column the vote column. As we will see, this choice will be used as a cut- and choose audit of the correctness of the encryptions that P has calculated.
- Next, V enters the entire audit column and the code for the chosen candidate from the vote column into the hardware token H. That is, if the audit column is B then V enters

$$d := B_1^* | \cdots | B_n^* | x^*$$

into H where $x^* \in \{A_1^*, \ldots, A_n^*\}$.
- The token H computes $C = Kh^d$ and displays the vote code C^* which V enters into the device P. V also makes a note of the vote code and the ballot ID in order to verify their presence on the bulletin board later.
- P sends the vote code C^* to S together with A, B, the lexicographically ordered ciphertexts.

2.4. Ballot processing by S

The server now needs to process the vote and perform checks and proofs. The aim is to be able to post on the bulletin board an encryption of the candidate chosen by the voter. First the server will determine which ciphertext in $A \cup B$ is the encryption of the voter's choice. For this S proceeds as follows.

- S computes $c_1, \ldots, c_{2n}$ like P and checks that these values are consistent with A, B, i.e., the last κ digits of each element in A equal a code in $c_1 \ldots, c_n$ and similarly for B. S also posts the voter ID, C^*, A, B and the ballot ID (which it calculates from A, B) to BB.
- S then computes the plaintext d entered into H by brute force. This is possible since S knows the secret hardware token key K. There are $2n^2$ possibilities for d, namely n choices for the cyclic shift of the audit column, n choices for the code chosen in the vote column and a factor of 2 from the choice of the audit column.
- Knowing d, S can determine the audit column α (α is A or B). S can also determine x^* from d which is the last κ digits of the encryption of the chosen candidate. This determines the encryption of the chosen candidate $A_* \in \{A_1, \ldots A_n\}$ (if the audit column was B). Note that the server cannot determine the shift in the vote column and hence cannot associate the vote ciphertext to a given candidate.

Next, using the audit column, S will check that P correctly encrypted the candidates. This is needed to avoid that, for instance, all ciphertexts displayed by P are encryptions of the same candidate. For this, S will request P to open the audit column.

- S publishes the audit column α on BB and requests the random coins used for the encryptions in the audit column from P. The publishing of the audit column on BB is to prevent a privacy attack that we found on an early draft of the Du-Vote protocol, where S would simply ask for the vote column instead of the audit column.
- P checks if S has posted the audit column to BB, and if true sends the random coins to S, i.e. $r_1^\alpha, \ldots, r_n^\alpha$ to S in that order.
- S posts the random coins in the correct order to BB, and checks that the encryptions are of the correct candidates. It knows the order of the vote codes in the audit column from d.
- If at this point any of the checks have failed, i.e. that S, H or V did not follow the protocol or made an error, then S refuses to do any further processing. This in particular also happens if d is not unique due to a collision. In this case, it is suggested that the voter should vote offline.

Finally S publishes (a re-encryption of) the encrypted vote on BB together with zero knowledge proofs ensuring the voter that S followed the protocol.

- S now re-encrypts A_* (the ciphertext of the voter's chosen candidate) to the ciphertext E and posts this to BB as the vote of V. This re-encryption is necessary as otherwise P would be able to identify the chosen candidate.
- S produces non-interactive zero knowledge proofs that it behaved honestly and publishes these on BB. The details of the proof can be found in [17]. An important part for us is that these proofs include publishing $C = Kh^d$ which is the untruncated value calculated internally in H.
- Finally, V checks that the ballot ID and vote code appear correctly on the bulletin board BB. This can be done by a third party without violating privacy, and should preferably be done from another device than P.

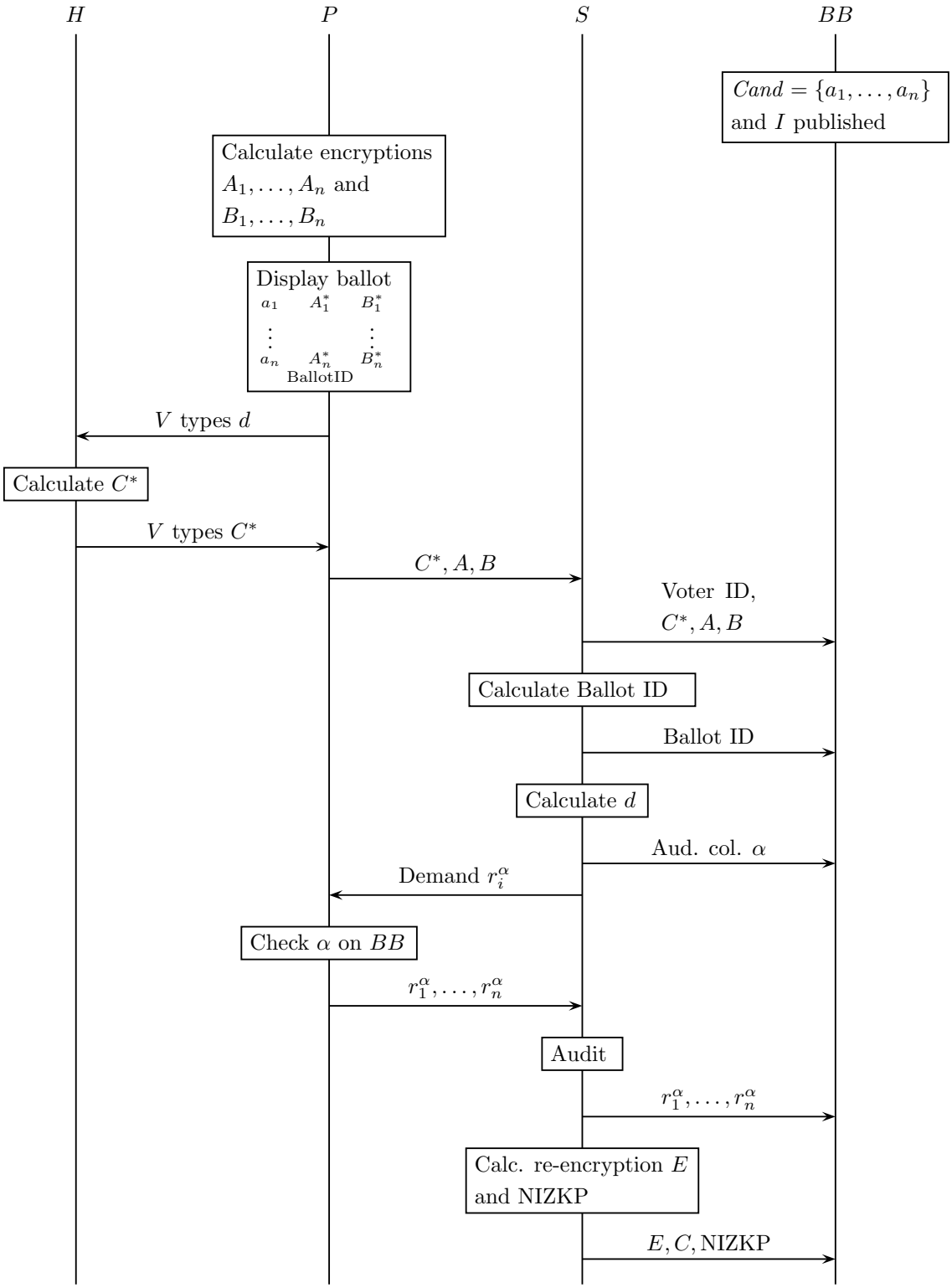

Figure 1. Diagram illustrating the important parts of the vote protocol.

A less explicit part of the protocol is the following: if just a simple disturbance is detected by the voter, e.g. that the computer needs to restart the voting session after entering the vote code, she has to abort the protocol and vote offline or by other means.

3. Assumptions and claimed security of Du-Vote

3.1. Assumptions

We will now recall the security assumptions made about the Du-Vote system. Firstly, we assume that the voting authority is trustworthy. If the voting authority is not trustworthy it can in many attacks below replace a colluding vote server. Further, we assume that the bulletin board is ideal, i.e. it is append-only and cannot be tampered with.

For the hardware token H, we assume that it has not been hardware modified for extra communication channels. One could e.g. imagine incorporating a microphone to allow P to send extra messages to H via the speakers of P. This could even happen in an inaudible frequency. We however allow the hardware to be software modified, i.e. the software could have been installed by the attacker during manufacturing. As we disallow any communication channel, this means that the adversary does not control the token in real time.

There should also be a time lapse between the ending of e-voting and the voting by other means. Otherwise it would be easy to kill last minute electronic votes by a denial-of-service attack if the voter does not have enough time to cast a paper vote instead.

We also keep the realistic assumption from [17] that not all computing platforms P can be infected with malware. This e.g. means that attacks using software-modified hardware tokens could potentially be detected on the honest computing devices.

3.2. Security claims

We now give an overview of the security claims made in [17]. The claimed level of verifiability of Du-Vote is rather high. If H is trustworthy, verifiability is secured. In this case, if the voter confirms the ballot ID and vote code on BB, then with a probability of almost $1/2$ the vote will be counted as cast: basically the only chance to change the vote is to guess which audit column will been chosen by the voter. The idea is then that there is roughly a probability $1/2$ for an adversary to provoke a halt in the protocol and hence large scale attacks will be detected.

Even in the case where H is untrustworthy (the software on all devices has been changed), verifiability is assured with high probability. This is under the assumption made above that not all computing platforms P can be corrupted, but a fraction will remain trustworthy. The malware infected P can be controlled live by the adversary and can be colluding with S, but the attacks proposed in [17] will be detected with high probability via the honest computing platforms P when they interact with the corrupted tokens H. If H is reprogrammed to output randomized codes, this will clearly be detected with large probability on trustworthy computing platforms. A more subtle attack was suggested where H substitutes codes, but without knowing I which is first known after the hardware production. However, also this attack will often be detected by an honest platform P. To be more precise, if h is the fraction of P that are honest and the adversary changes N votes, then

$$\frac{hN(10^\kappa - n)}{n((1-h)n+h)}$$

votes are rejected by the trustworthy platforms P. The results are summarized in table 2.

The Du-Vote protocol is designed with emphasis on verifiability. The privacy security is correspondingly not quite as strong. If P and S are colluding, privacy is clearly violated, but if H is honest and if either P or S is also trustworthy (or at least P and S are not colluding), then privacy is guaranteed. If H is corrupted, e.g. programmed to return the chosen candidate code as vote code, privacy is also violated, but this would easily be detected. The claimed privacy results are presented in table 3.

Verifiability		
	H honest	H corrupt
P & S colluding	If the voter confirms the ballot ID and vote code on BB, the probability that the vote is decrypted as cast is $0.5(1-10^{-\kappa})^{2n^2-1}$	If a fraction h of P are honest and the adversary changes N votes, then $\frac{hN(10^\kappa-n)}{n((1-h)n+h)}$ votes are rejected.

TABLE 2. VERIFIABILITY OF DU-VOTE AS PRESENTED IN [17]

Privacy		
	H honest	H corrupt
P corrupt, S honest	✓	No privacy, but attack is detectable
P honest, S corrupt	✓	No privacy, but attack is detectable
P & S colluding	No privacy	No privacy

TABLE 3. PRIVACY OF DU-VOTE AS PRESENTED IN [17]

4. Privacy attacks

We now present several attacks on the privacy of the votes in the Du-Vote protocol.

4.1. Replay attacks

Note that as the protocol stands, it does not ensure privacy in the strict formal sense, at least not with respect to an adversary who is a voter or controls voters and for these can control the communication with S.

The reason is that the re-encrypted vote is posted on the bulletin board. This means that a vote copy attack, similar

to the one by Cortier and Smyth [18] on Helios, is possible. In the Du-Vote scheme, simply copying a re-encrypted vote would however not work. An adversary has to re-encrypt it until the last κ digits correspond to say A_1^*. The remaining encryptions are chosen honestly. The adversary now chooses column B as audit column and votes for the first candidate which in the end will be a vote cast for the candidate encrypted in the copied vote. The same can be done for controlled voters. This will change the distribution of the votes and thus violates privacy. To better appreciate the effect of this attack, consider a situation where 2 honest voters and 1 dishonest voter are eligible. If the dishonest voter copies Alice's vote, the election result will entirely leak Alice's choice.

A simple solution is to delay the publishing of the last part of the zero-knowledge proofs and the re-encrypted vote until after the e-voting has stopped. The vote code and ballot ID should however be published immediately such that the voter can check this part right away, keeping the protocol *vote & go*, as the remaining verifications can be performed by any person who wishes to verify the results.

4.2. Phishing (P corrupt, H passively corrupt)

Via, e.g., phishing mails an adversary can try to make the voter enter all zeroes (or some other chosen value) into the device H and reveal this value online. If the computer is then simultaneously infected with malware, this will reveal the vote to the adversary, as we will explain below. An infected platform could also simply request the voter to enter a first code for initialisation purposes. We denote this by H being passively corrupt since no hard- or software modifications are necessary.

The reason that an access to a second output from the device breaks privacy is that on the bulletin board BB both the internal output from H i.e. C and the column which was used for auditing is published. Given C and the audit column there are only n possibilities for the secret key K. The second output from H can then be used to select the correct key among the n possibilities. To be explicit, assume that the audit column is B, then K is one of the n values

$$K_i = Ch^{q-B_1^*|\cdots|B_n^*|A_i^*}$$

for $i = 1, \ldots, n$. The adversary can check which of these keys gives the same result as in the phishing attack, and thus with high probability uniquely determine A_i^*. If the adversary has also infected P with malware this directly determines the vote.

This attack could also be carried out by any person which has access to the hardware token before or after the voting procedure, e.g. a person in the same house or on large scale the delivery service, and who at the same time knows the ballot shown by P. It is thus important that nobody sees the screen on P during the voting and it is not enough to hide the input into H. Note however that it is also necessary to know the voter ID. Therefore it is important not to write the voter ID on H, but rather some serial number. Also note that seeing the ballot, displayed on P, makes it possible to determine the voter ID afterwards, searching the voter with the correct audit column vote codes. In the next attack, we will see, that H could also reveal the voter ID directly.

The counter measure is of course to make sure that the voter is well-informed of the importance of using H only once and to keep H, and the voting session private. This attack is based on the note in the original paper [17] where they explain that H must not be used with the same key for two different elections since P then knows the key K with probability $1/n$. Another reason to explicitly mention this attack is that it should be kept in mind as a difficulty for a countermeasure for the verifiability attacks from sections 5.2 and 5.3 below.

4.3. Key uniqueness (P corrupt, H passively corrupt)

The previous attack assumed that we got extra information from H while knowing the identity of the voter. However, since the secret keys in the hardware tokens are with high probability unique, partial knowledge of the key might reveal the voter ID. This can in turn be used for privacy attacks on voters with corrupt P along the lines of the last section.

Consider a worker on the hardware production factory. This worker could simply enter some values say 0 and 1 into the hardware token and write down the outputs. No notion of serial number or relation to the voter is needed. The adversary can now for each voter on BB calculate n possible secret keys (see section 4.2) and check which one complies with the extra inputs to H.

This attack can also be carried out by anyone else who has access to the hardware token e.g. the delivery service if the hardware tokens are not properly secured, or on a large scale the hardware producer which directly have the secret token keys. It could also be someone finding the token in the voters home or it could even happen after voting, e.g. after the token has been disposed in the garbage.

Let us now calculate the probability that the extra information can uniquely determine the voter ID of the voter that uses the passively corrupted H. We assume that v other voters have voted and that the hardware token has revealed the output of m extra predetermined input values. We also assume for simplicity that the hardware token keys are unique since they should have high entropy. For a given vote on BB and a choice of one of the n possible keys, the chance that it reproduces the revealed information is $10^{-m\kappa}$ where we have assumed independence of the outputs. The probability that the revealed information does not comply with any of the n possible keys of each of the v voters is thus $(1 - 10^{-m\kappa})^{nv}$. As an example with $\kappa = 4$, $n = 4$ candidates, $v = 100.000$ voters and making only $m = 2$ inputs to h, the probability that the voter ID can be uniquely determined is 0.996.

The person in possession of the extra readings can thus with high probability determine the voter ID and sell the extra output information and the voter ID bundled. An

adversary controlling the computing device of the voter with the corresponding voter ID can directly use the extra information to break privacy. Luckily, this attack only violates privacy since it can only be performed after vote casting.

The attack illustrates the importance that the hardware tokens are delivered in tamper resistent envelopes to the delivery service D. Also the device should be made unusable before disposing of it. Another choice would be to make the hardware token limited to only work one time, but that would certainly decrease usability. Further this would also prevent checks of the honesty of the vote server S, see sections 5.2 and 5.3 below.

4.4. Information leak from no error (P corrupt)

Since the vote processing by S with large probability halts in the case of an error, e.g. if the voter did a typo when entering the codes on H, this actually gives knowledge to an adversary. Imagine an adversary controlling P and making one or more small changes in the vote codes in say column A of the displayed ballot. The encryptions of the candidates are however calculated honestly. With probability $1/2$ the voter chooses B as the audit column, and if the vote is processed by S, it means with large probability that V did not vote on the candidates with corrupted vote codes. If S does not process the vote no knowledge is gained – either the voter used column A as audit column or used B as audit column, but voted on one of the candidates with corrupted codes.

It could happen with low probability that a corrupted vote code is processed if it happens to give a C^* matching one of the $2n^2$ real vote choices. This probability is thus $1-(1-10^{\kappa})^{2n^2}$ i.e. lower than $2n^2 10^{-\kappa}$. However, in this case the audit column that S determines is most likely not consistent with the honestly computed, and it will be revealed to the adversary. Thus again the adversary gets no information, but not the false information that the voter did not choose the altered codes.

Note that a similar attack could be used to actually alter the vote (e.g. exchange vote codes of the candidates), but the point is that this attack is not scalable since it will be detected with probability around $1/2$ due to the audit column procedure. However, a privacy attack does not need to be scalable since the knowledge of certain votes are more interesting than others. Also note, that even though S cannot process the vote, the failure could just as well come from a typo from the voter, and is thus almost undetectable if it is not carried out on a big scale. According to [20], entering 10 digits into a calculator has an error rate of 0.05. We would expect the typo rate to be on the same scale since the voter here has to enter more digits (e.g. 20 for $k=n=4$), but is probably quite careful in the voting process.

A possible countermeasure is that the voter actually copies the audit column and checks this against the published values on BB on a different computing platform. The voter can also safely let a third party perform this check. However, if the voters are instructed to make such checks the usability of the voting protocol suffers considerably. Another option is to incorporate this check of the audit column into the token H, see section 5.1 below.

4.5. Information leak from no collisions (P corrupt)

An even more serious attack comes from the fact that if S processes the vote then we know there are no collisions in C^* between the actually chosen vote and any of the $2n^2-1$ other possible entries to H (see section 2.4).

From the output to BB we saw in section 4.2 that anyone can calculate n possibilities for the secret key $K_i = Ch^{q-B_1^*|\cdots|B_n^*|A_i^*}$ for $i=1,\ldots,n$ where we have assumed that the audit column was B. If K_i is really the true key, we know that

$$\begin{aligned} C^* \neq &(K_i h^{B_j^*|\cdots|B_{j+n-1}^*|A_k^*})^* && \text{for } j,k=1,\ldots,n \\ & && \text{except } (j,k)=(1,i) \\ C^* \neq &(K_i h^{A_j^*|\cdots|A_{j+n-1}^*|B_k^*})^* && \text{for } j,k=1,\ldots,n \end{aligned}$$

where the addition in the indices of A, B are modulo n. Note that these constraints are all different, also across the different values of i. For a given i the probability that there is a collision, i.e. the probability that we can rule out that V voted for the candidate encrypted in A_i, is $1-(1-10^{-\kappa})^{2n^2-1}$. The probability that one of the keys K_i have a collision is thus

$$1-(1-10^{-\kappa})^{(2n^2-1)(n-1)} \tag{4.1}$$

An adversary controlling P, or anyone seeing the displayed ballot, can now with this likelihood break privacy in the mild form of excluding at least one candidate. For $n=\kappa=4$ we have a low probability of 0.00926 i.e. only about one in hundred, but with more candidates say $n=6$ and setting $\kappa=3$ to reduce the number of number the voter has to type, we have a rather high probability of 0.299 i.e. a bit less than one in three.

A way to limit the possibility of these attacks would be to increase the length of the output from H to say 2κ. The downside would be that more typing is needed from the voter, and it could be faster to determine the secret token key using inputs to H.

All the privacy attacks are summarised in table 4 according to trust assumptions.

Privacy attacks		
	H honest	H passively corrupt
P & S honest	Replay attack, sec. 4.1	
P corrupt, S honest	Information leak from no error, sec. 4.4 Information leak from no collisions, sec. 4.5	Phishing, sec. 4.2 Key uniqueness, sec. 4.3

TABLE 4. PRIVACY ATTACKS ON THE DU-VOTE PROTOCOL ACCORDING TO TRUST ASSUMPTIONS. H IS PASSIVELY CORRUPT WHEN IT IS BEING MISUSED, BUT ITS SOFTWARE IS NOT BEING CHANGED.

5. Verifiability attacks

We now consider six attacks where votes can actually be changed.

5.1. Ballot stuffing (S corrupt)

We first note that the hardware token does not prevent S from voting on behalf of the voter. The secret token key is known to S and it can thus fully simulate the voting process with a chosen vote. Of course, this will be disputed by a voter actually trying to vote or – somewhat more unlikely – checking the BB.

A general solution to such ballot stuffing problems was proposed in [19]. In this case it would be employed by providing the voters with credentials and letting them sign the vote code when P sends it to S.

One could have hoped that such extra infrastructure could have been circumvented using the hardware token. Indeed, we suggest the following modification. Each hardware token H now also stores a second secret key, denoted K_2, besides the secret key K. As before, during the voting process, the voter enters $d = B_1^*|\cdots|B_n^*|x^*$ into H if V chose column B as audit column and wants to vote for the candidate with code x^* in column A. In the modified scheme, the hardware token H now displays $\kappa + 2$ digits, i.e. two extra digits. The first κ digits are again $[Kh^d]^*$, whereas the extra output is the last two digits of $K_2 h^{d-d^*} = K_2 h^{10^\kappa B_1^*|\cdots|B_n^*}$. Note that the extra output only depends on on the audit column and the extra secret key. The voting process is as before, but now S publishes the $\kappa + 2$-digit vote code on BB and V checks this.

The voting authority A commits to K_2 on BB before I is announced, but the point is that it first reveals it after the voting has ended. Thus S, even if it is colluding with P, cannot guess the extra output with a better probability than $1/100$. Everybody can now check that with probability $99/100$ the vote came from the selected voter. This would make ballot stuffing from S easily detectable, unless S colludes with A, which we assumed trustworthy. Note that this is contrary to the solution with credentials which is not safe when S and P are colluding. Further, the extra output only depends on the audit column, which is public knowledge, and the second secret key, so it will not reveal anything about the vote. Also, the extra secret key is not easily revealed by brute force typing attacks on H since only two digits are displayed. The downside of this construction is that the extra digits make it easier to link a passively corrupted token as in section 4.3 to a cast vote displayed on the bulletin board, thus making this privacy attack easier. It would thus be of advantage if A proves in zero knowledge that the key it committed to indeed generated the extra output. However, we have not found an efficient proof for this.

5.2. Fake key attack (S corrupt)

The zero knowledge proofs that the vote server S presents on BB, does not reveal which secret key $K = y^k$ is used in its computation, but it is important for the construction of the proofs that $k = \log_y K$ is known to S.

This gives rise to the following attack which allows S to change the vote in an undetectable way. As in the normal protocol execution, S computes the code $d := B_1^*|\cdots|B_n^*|x^*$ that the voter gave as input to H using the vote code C^* (here assuming B is the audit column). This means that the voter chose the candidate with code x^* in column A. S can now alter this to a vote for the candidate with code A_i^* as follows: S computes $\left(y^{k+j} h^{B_1^*|\cdots|B_n^*|A_i^*}\right)^*$ for $j = 1, 2, \ldots$ until this value is equal to C^*. This is equivalent to the computation that P performs for the encryption of a candidate, and according to [17] this should only take 0.24 seconds on average using a Macbook air. Now S simply uses the fake key $K_{\text{fake}} = y^{k+j}$ to compute all the NIZK proofs and records the vote A_i for V.

Note that S cannot choose A_i to correspond to a particular candidate c, rather this is a *cyclic* attack. Namely, the server's choice is a function of the voter's choice as it can choose that the vote goes to the candidate that is, e.g., 3 positions further in the candidate list. One can therefore exploit an a priori known vote preference of a given class of voters. If S, P are colluding, privacy is broken and S can directly choose the candidate.

A possible way to mitigate this attack is that voters use their hardware token H to test whether S uses the correct key. This is done like in section 4.2 by first calculating the n different possible keys that can be derived from the published C. Then the voter types an extra value into H, e.g. zero, and checks if this output would also have been produced by one of these n keys.

This can also be done using a third party. V can reveal the extra information from H and also the candidate vote code used, and let this third party check if the results comply with the published C. It will only reveal the true vote to the chosen third party if this party controls S or has been shoulder surfing during the voting, cf the privacy attack in section 4.2. However, the counter measure of that attack was that the voter is instructed to use H only during the vote process. This extra check hence might be confusing to the voters and could be exploited.

A more useful and elegant way to prevent the attack is to add the following NIZKP to the protocol. Before the vote nonce I is determined, S commits to $K = y^k$ using El Gamal encryption by publishing $(g^r, h^r y^k) := (H_1, H_2)$ along with the voter ID on BB. When S publishes C on BB, it also adds the following OR-proof of knowledge

$$\text{SPK}\{r|\exists d : d \in \Omega \wedge g^r = H_1 \wedge h^r = H_2 h^d / C\} \quad (5.1)$$

We here use the notation of Camenisch and Stadler [21] since this is the one employed in the original paper [17]. Ω is the set of the $2n^2$ possible inputs to H. The notation suggests that this is a double ring signature proof of knowledge where the public keys are the pairs $\{(H_1, H_2 h^d / C)|d \in \Omega\}$. The construction of the remaining NIZK proofs in [17] also use this type of OR-proofs. The proof shows that $C = Kh^d$ for one of the possible inputs d, but of course not that the correct input is chosen. However, since we assume that the

voter checks that C^* is indeed the vote code displayed on H, S can only cheat if there are collisions. This will be the subject of the next attack.

The NIZKP above does not show that the correct key is used, just that the key which was committed to before the voting was used, and this e.g. allows for denial-of-service attacks. One could mend this if S sends the random coins used in the encryption to the voting authority A, and then A signs the encryption and publishes this after checking that the correct key is used. Again this assumes that A is trustworthy.

5.3. Collision attack (S corrupt or S, P colluding)

The vote server is supposed to stop processing the vote if S cannot uniquely determine the input d to H due to collisions, i.e. there are more than one of the $2n^2$ possible outputs that match the actual vote code. However, the present NIZK proofs does not to give a direct proof that S indeed does this. This opens for possible attacks.

The probability of collisions can luckily be rather low and decreases with κ, however, increasing κ has to be weighted against usability. To be precise, the probability that there is a collision in C^* between the voter's choice and the $2n^2 - 1$ other possibilities is, as in section 4.5,

$$p_{\kappa,n} = 1 - (1 - 10^{-\kappa})^{2n^2-1}$$

These probabilities are low, but non-negligible, e.g. for $\kappa = n = 4$ we have $p_{4,4} = 0.0031$, but this still means that one in 300 voters will have problems with their vote, and will not have the vote processed by an honest S. For $\kappa = 3$ and $n = 5$ we have $p_{3,5} = 0.048$ i.e. one in twenty have problems.

If an adversary controlling S wants to use these collisions to change votes without the possibility of getting caught, the probabilities fortunately decreases further. The point is that S does not know which ballot P constructed. However, it can happen that there is a collision in C^* between the voter's choice and another choice using the same audit column, and at the same time there is no collision with the remaining $2n^2 - n$ possibilities. In this case, S could simply process the vote using one of the colliding choices and this would not be detected when P reveals the encryptions of the audit column. Thus with probability a bit more than $1/2$, S will have changed the vote. The probability of such an inner-column collision (and no other collisions) is

$$\left(1 - (1 - 10^{-\kappa})^{n-1}\right)\left(1 - 10^{-\kappa}\right)^{2n^2-n}$$

For $\kappa = n = 4$ we have a probability of 0.00030 i.e. only three in ten-thousand votes could be attacked, and for e.g. $\kappa = 3$ and $n = 5$ we have the probability 0.0038 i.e. a bit less than four in thousand votes could be attacked. The probabilities are thus low, but in large scale elections there is a good probability to actually change votes, and we know that in large elections it can happen that a disproportionally small number of votes can change the outcome.

However, if S and P are colluding the probability of collisions become much higher. Since the adversary then knows the secret token key, the positions of possible collisions are known. P can then make sure to use a ballot choice which contains a collision. The probability of a collision is then (the standard birthday attack probability)

$$p'_{\kappa,n} = 1 - \frac{10^{\kappa} \cdot (10^{\kappa} - 1) \cdots (10^{\kappa} - 2n^2 + 1)}{10^{2\kappa n^2}} \tag{5.2}$$

For $\kappa = n = 4$ we have $p'_{4,4} = 0.048$ i.e. one in twenty votes could be attacked. For $\kappa = 4$ and 5 candidates we have $p'_{4,5} = 0.115$, and if we lowered κ to 3 in order to be user friendly (i.e. the voter only has to enter 18 digits to H), we would have $p'_{3,5} = 0.712$.

The kind of attack that can be carried out depends on the nature of the collision. First, consider the case where there is a collision within the same audit column, say the input $B_1^*|\cdots|B_n^*|A_i^*$ gives the same output from H as $B_1^*|\cdots|B_n^*|A_j^*$ where $i \neq j$. Then the adversary chooses $B_1^*, \cdots, B_n^*$ as column B. Further, if the adversary knows whom the voter is most probably going to elect, P can even put the collision on this candidate. This could e.g. happen if the malware is directed at a specific candidate's vote base, or for a general voter if some candidate is known to be the most popular. Say that the voter is likely to choose candidate one, then column A is chosen as $A_i^*, A_{i+1}^*, \ldots, A_{i-1}^*$ with addition modulo n in the index, and the encryptions are done correctly corresponding to this choice. With probability $1/2$ the voter will choose audit column B, and if V votes for the expected candidate the attack can be launched, and the vote can be changed into candidate $j - i + 1$ in this case. The chance that there is a choice of audit column with such an inner-column collision is

$$1 - \left(\frac{10^{\kappa} \cdot (10^{\kappa} - 1) \cdots (10^{\kappa} - n + 1)}{10^{\kappa n}}\right)^{2n}$$

For $\kappa = n = 4$ the probability is 0.0048, i.e. one in two hundred are vulnerable to this attack, but for $\kappa = 3$ and 5 candidates we have a probability of 0.095 i.e. one in ten are vulnerable.

The second possibility is that the collision is within the same choice of audit column (i.e. A or B), but that the two columns are permuted compared to each other, e.g. the inputs $B_1^*|\cdots|B_n^*|A_i^*$ and $B_2^*|\cdots B_n^*|B_1^*|A_j^*$ result in the same output of H. In this case an attack can be carried out if $i \neq j$. However, it requires that one column, in this case B, is diplayed dishonestly to the voter. Say that the honest ballot uses $B_1^*, \cdots, B_n^*$ as column B, but P presents $B_2^*, \cdots B_n^*, B_1^*$ as column B to the voter with A_j^* placed at the candidate, the voter is most likely to use, like above. If the voter uses column A as audit column, the vote will go through and the postings on BB are verifiable, however, if V voted for candidate m it will really be a vote for candidate $m+1$ calculated modulo n. If V chooses column B as audit column and votes for the expected candidate, the collision attack can be carried out and the vote is changed to a vote for candidate $i-j+k$, where k is the voter's favored candidate. However, if V chooses A as audit column and does not vote

as expected, S can most probably not fill in the BB in a way that would be verified due to the fake audit column. Instead a denial-of-service would have to be made. For this attack it is thus essential that the adversary is very confident about what V is going to vote. As mentioned above, according to [20] entering 10 digits into a calculator has an error rate of 0.05. Let us assume that entering the 20 digits into H (for $\kappa = n = 4$) has roughly the same rate. For the denial-of-service rate due to the fake column to be in the same order, the adversary thus has to be 90% sure about what the voter is going to choose. Thus the attack is limited to very dedicated voters. The probability that a collision like this happens is

$$1 - (1 - 10^{-\kappa})^{n^2(n-1)^2/2}$$

For $\kappa = n = 4$ the probability is 0.0072, i.e. less than one in hundred are vulnerable to this attack, but for $\kappa = 3$ and 5 candidates we have a probability of 0.18. We note that this attack could be revealed with probability $1/2$ if the voter also check the audit column codes on BB (at least the first one). Note that if we modify H to contain an extra key and output two extra digits as suggest in section 5.1, the audit column is also automatically checked.

The last possibility is that the collision is for different choices of audit columns. This is the most likely case and happens with a probability of

$$1 - (1 - 10^{-\kappa})^{n^3(n-1)}$$

when we exclude the cases where the collisions point to the same candidate. For $\kappa = n = 4$ the probability is 0.019, i.e. a bit less than two in hundred are vulnerable to this attack, but for $\kappa = 3$ and 5 candidates we have a probability of 0.39. For this attack the ballot chosen by P is of course the one with a collision and the encryptions are honestly constructed. In this case the collision cannot be put at a chosen candidate so on average there is a probability of $1/n$ that the attack can be triggered. If the adversary has no idea what the voter actually has chosen to vote and if the collision is triggered, there is only a probability $1/2$ that the vote will be changed against the voters choice. Note that the vote will then be processed with the audit column opposite to the voter's choice. Thus an easy repair is to demand that the voters check that the audit column selection (A or B) on BB corresponds to their actual choice. The modified token H with an extra key and two extra output digits from section 5.1 again automatically checks this.

A general solution to these collision attacks would be to find a NIZKP that shows that no collision appears for the vote chosen by V. However, it is not clear to us how to do this efficiently. Another possibility is that the voter or a third party, like in the last section, use H on extra input to determine the correct secret key from C displayed on the bulletin board. Then it can be checked if there is a collision for the given C^* or not. Again this is not ideal since using H several times opens up for privacy attacks. However, since S probably is risk avoiding, and if a small fraction of the voters do this check, the possibility of getting caught might prevent S from using this attack. Alternatively, the voting authority A, which knows K, could also check if S honestly stops, when there is a collision.

Let us also mention that there actually is a very small probability that the information on the BB could prove the existence of a collision. This happens if all of the possible keys K_i, $i = 1, \ldots, n$ that are calculated from C leads to collisions. However, S could calculate this before carrying out the attack and in case refrain from the attack.

Finally note, that for this attack an increase in the number of digits in the vote code output by H and entered into P by V would greatly help. Indeed, if we have $\kappa = n = 4$, but H outputs the last eight digits of C, instead of only four, we do not have any collisions with a probability of 0.9999950.

5.4. Computational attack (S, P corrupt)

In this attack P needs to know the secret key K e.g. from S. The attack is computationally heavy in the number of candidates, so it is most easy to carry out for $n = 2$ candidates. Assume the adversary wants the voter to vote for candidate 1. P first constructs honestly candidate encryptions and an honest ballot with columns A_1^*, A_2^* and B_1^*, B_2^*, but it does not show the honest ballot to V. It also computes the vote codes $C_A^* = (Kh^{A_1^*|A_2^*|B_1^*})^*$ and $C_B^* = (Kh^{B_1^*|B_2^*|A_1^*})^*$ corresponding to votes on candidate 1.

The idea is then to compute and display a fake vote sheet $\{\{A_1', B_1'\}, \{A_2', B_2'\}\}$ such that the output of H, no matter what the voter chooses, will always be C_A^* or C_B^*. This is done as follows. Let $M = \{0, \ldots, 10^\kappa - 1\}$.

- For $A_1' \in M$ and $A_2' \in M \setminus \{A_1'\}$ do the following
- search for the $B' \in M \setminus \{A_1', A_2'\}$ such that $(Kh^{A_1'|A_2'|B'})^* \in \{C_A^*, C_B^*\}$. On average, there will be two such B'. The calculation should only take around 0.24 seconds using a Macbook air since it is similar to the calculation of the encryption done in [17].
- If two or more B' are found, then check if there are two of these $B_1' \neq B_2'$ such that $(Kh^{B_1'|B_2'|A_i'})^* \in \{C_A^*, C_B^*\}$ for $i = 1, 2$. If this is true, we are done, otherwise proceed with the next A_i'.

The extra checks in step 3 should maximally take the same time as step 2. On average there should be just below 16 fake ballots of the wanted form.[2] Thus, on average, a fake ballot should be found after $\frac{1}{32}10^{2\kappa} \cdot 2 \cdot 0.24$ seconds on a Macbook air. Actually, we do not need to choose the permutation in column A and B before calculating the fake sheets. There are thus four sets of codes (C_A^*, C_B^*) that are usable, and we should only need around $\frac{1}{128}10^{2\kappa} \cdot 2 \cdot 0.24$ seconds for the calculation. For $\kappa = 4$ this is 4.3 Macbook

2. This is actually hard to estimate. If we find m B' satisfying the constraints, we can organise these into a column in $m(m-1)$ different ways. A calculation shows that this gives just below 4 different columns on average. For the choice of A_1', A_2' should come another factor of $2 \cdot 2 = 4$.

air processor days, but the calculation is completely parallelisable and can be computed as soon as the election nonce I is known.

Again, this attack can be repaired if the voter also checks that the audit column on BB is correct. Alternatively, the construction from section 5.1 with an extra secret key in H and two extra output digits could also be used.

5.5. Fake extra candidate attack (P, H corrupt)

Let us now consider the case where the adversary has the power to change the software on H. We can then imagine the following attack: When P is infected with malware it displays a valid ballot, but augmented with an extra candidate in a specific position. This extra candidate could be custom tailored to look less suspicious, e.g. a known politician not on the candidate list. The idea is that the extra code from this extra candidate row is used by H to give the adversary's desired output.

Let us now describe the attack in detail. We assume that the adversary wants the vote to go to the first candidate. P then calculates encryptions in the standard way with column A as $A_1^*, \ldots, A_n^*$ and column B given by $B_1^*| \ldots |B_n^*$. However, P displays a ballot to the voter with an extra fake row inserted at position k. Column A is then $A_1^*, \ldots, A_{k-1}^*, B_1^* + r, A_{k+1}^*, \ldots, A_n^*$ and the column B is $B_1^*, \ldots, B_{k-1}^*, A_1^* + r, B_{k+1}^*, \ldots, B_n^*$. Here r is a random number programmed into H and addition is modulo 10^κ. When H gets a vote code with κ extra digits of the form $a_1| \cdots |a_n|a_{n+1}|a_{n+2}$, it is programmed to change this input to $a_1| \cdots |a_{k-1}|a_{k+1}| \cdots |a_{n+1}|(a_k - r)$, and use this to calculate the output in the standard way. Effectively, this changes the vote from a vote on the candidate with code a_{n+2} to a vote for candidate 1. On honest P, the hardware token behaves normally, and these will thus not reveal the attack.

The display might not sustain the extra κ digits, but could simply scroll to show these, but hide the first entered digits.

The attack is of course easily detectable for the alert voter unless there are many candidates. As in several previous attacks it will also be revealed if the voter checks that the audit column displayed on P is equal to that on BB. However, the repair where H has an extra secret key from section 5.1 is easy to circumvent in this case.

5.6. Special codes (H, P, S corrupt)

If both P and S are corrupt and can change the software on H, there is a very simple attack. P gets the secret key from S (this is the only involvement of S). P calculates encryptions and determines column A and B without showing it to the voter. It calculates the desired outputs by H, say if the adversary's choice is candidate 1 it calculates $C_A^* = (Kh^{A_1^*|\cdots|A_n^*|B_1^*})^*$ and $C_B^* = (Kh^{B_1^*|\cdots|B_n^*|A_1^*})^*$. The idea is now that P changes the codes in the first $n-1$ rows of the ballot to alert H and gives the chosen vote code in the last row.

This could happen in the following way: P shows a ballot to the voter where it has made changes in up to four digits of the vote codes of the first $n-1$ rows such that the sum modulo 10^κ of these rows is x, and it writes $C_A^* + r$ modulo 10^κ in the last row of column A. Here x and r are two κ-digit numbers. It does the same in column B where the last row now shows $C_B^* + r$ modulo 10^κ. H is programmed such that if it gets an input of the form $a_1| \cdots |a_n|a_{n+1}$ and the sum of $\sum_{i=1}^{n-1} a_i$ modulo 10^κ is x, it displays $a_n - r$ modulo 10^κ. After V has entered this code into P, it proceeds with the protocol as if it had shown the unaltered ballot to V. In the end, a vote for candidate 1 will be registered with high probability.

If P is not infected with the specific malware, then H will only with a probability $10^{-\kappa}$ accidentally change the vote. This is indistinguishable from voters making a typo when entering the codes into H or P (remember that the error rate for this is around 0.05).

This attack will be caught if the voter carefully checks the audit column which should be displayed explicitly on BB. Note that in this case it does not help to check the audit column number. The attack is also prevented by the modification from section 5.1 where H has an extra secret key and displays two extra digits based on the audit column. The point is that even if the hardware producer is colluding, the link between the second secret token key K_2 and the voter is not known to the colluding parties, but only to the voting authority, at the time of the election.

All the verifiability attacks are displayed according to trust assumptions in table 5.

Verifiability attacks		
	H honest	H corrupt
P corrupt, S honest		Fake candidate, sec. 5.5
P honest, S corrupt	Ballot stuffing, sec. 5.1 Fake key, sec. 5.2 Collision attack, sec. 5.3	
P & S colluding	Collision attack, sec. 5.3 Computational attack, sec. 5.4	Special codes, sec. 5.6

TABLE 5. VERIFIABILITY ATTACKS ON THE DU-VOTE PROTOCOL ACCORDING TO TRUST ASSUMPTIONS.

6. Conclusion

In summary, we have found a number of attacks on privacy and verifiability for the Du-Vote protocol, and we have suggested some repairs.

An easy change was to delay the announcement of the re-encrypted vote to prevent the replay attack from section 4.1. To stop the phishing attacks of section 4.2 it was important that the voters are instructed to use the hardware token H only once. This is problematic for the countermeasures suggested in the fake key attack, section 5.2, and the collision attacks in section 5.3. Fortunately, we found alternative countermeasures for both of these

attacks. In the former case a new zero-knowledge proof was constructed such that S cannot cheat. In the latter case, the voting authority A will have to make computations to check whether S behaves honestly. This requires A to do this in a trustworthy way.

Several attacks were based on the idea that P shows a manipulated ballot to the voter. This happened in the privacy attack using information leak from no errors in section 4.4, and in the verifiability attacks using collisions (section 5.3), using heavy computations (section 5.4), using fake extra candidates (section 5.5) and using special codes (section 5.6). A repair which prevents these attacks is to instruct the voter to copy the audit column and check that the BB displays the correct audit column number and some, or better all, of the vote codes in the audit column. However, this means a lot of work for the voter and certainly decreases the usability of the protocol. Note that printing the whole ballot and showing this to a third party opens up for privacy attacks e.g. in the form presented in section 4.5.

To prevent ballot stuffing (section 5.1) we noted that the standard solution of giving the voter credentials will also work in this case. However, we also suggested another solution, where an extra secret key is stored in the hardware token H and is used to compute two extra digits (or more if wished) in the output. This assured with probability $99/100$ that the vote is indeed cast by the voter self. Again, this assumes an independent trustworthy voting authority A. The extra digits also automatically checks that the displayed audit column is correct, and it can thus replace the above cumbersome repair where the voter needs to check the audit column on BB. Only the fake extra candidates attack (section 5.5) cannot be prevented by this method, but should be noticed by alert voters.

Two privacy attacks are hard to prevent. For the information leak from no collisions (section 4.5) there is no certain patch. However, the probability can be greatly lowered if H shows a longer output, but again this means more typing for the voter. For the key uniqueness attack in section 4.3 we also found no direct repair except to note that the tokens have to be kept very private.

In conclusion, we believe that the Du-Vote system is not ready to be deployed, even with the fixes we suggest, for several reasons. First, the usability of the protocol is, as also noted in the original paper, rather bad and the protocol is not robust against a voter that does not follow the instructions correctly (and moreover can be instructed by a malware to follow different instructions). Second, some of the attacks we discovered are only partially mitigated by our fixes. Finally, the protocol misses a rigorous security proof which is complicated as it requires a model that must take into account that vote codes are short and hence collisions may be found, as well as tokens that may be software modified but are not completely under the control of the attacker.

Acknowledgements. We would like to thank Gurchetan Grewal and Mark Ryan for interesting discussions on the Du-Vote protocol, as well as the anonymous reviewers for their comments. This work has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC) and the ANR project SEQUOIA ANR-14-CE28-0030-01.

References

[1] E. I. V. Committee, "Statistics about Internet voting in Estonia," http://www.vvk.ee/voting-methods-in-estonia/ engindex/statistics.

[2] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman, "Security Analysis of the Estonian Internet Voting System," in *Proc. of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*, 2014, pp. 703–715.

[3] J. A. Halderman and V. Teague, "The new south wales ivote system: Security failures and verification flaws in a live online election," *CoRR*, vol. abs/1504.05646, 2015.

[4] L. Grégoire, "Comment mon ordinateur a voté à ma place (et à mon insu)," May 2012, http://tnerual.eriogerg.free.fr/autovote.pdf.

[5] Supreme Court of Estonia, "The supreme court dismissed an electoral complaint regarding e-voting security," Mar. 2011, http://www.nc.ee/?id=1235.

[6] S. Delaune, S. Kremer, and M. Ryan, "Verifying privacy-type properties of electronic voting protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 435–487, 2009.

[7] M. Backes, C. Hriţcu, and M. Maffei, "Automated verification of remote electronic voting protocols in the applied pi-calculus," in *Proc. IEEE Computer Security Foundations Symposium (CSF'08)*. IEEE Computer Society Press, 2008, pp. 195–209.

[8] R. Küsters, T. Truderung, and A. Vogt, "Verifiability, privacy, and coercion-resistance: New insights from a case study," in *Porc. 32nd IEEE Symposium on Security and Privacy (S&P'11)*. IEEE Computer Society Press, 2011, pp. 538–553.

[9] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi, "A comprehensive analysis of game-based ballot privacy definitions," in *Proc. 36th IEEE Symposium on Security and Privacy (S&P'15)*. IEEE Computer Society Press, May 2015.

[10] R. L. Rivest and J. P. Wack, "On the notion of "software independence" in voting systems," 2006, prepared for the TGDC, and available at http://people.csail.mit.edu/rivest/RivestWack-OnTheNotionOfSoftwareIndependenceInVotingSystems.pdf.

[11] D. Chaum, "Surevote: Technical overview," in *Proc. Workshop on Trustworthy Elections (WOTE'01)*, 2001.

[12] P. Y. A. Ryan and V. Teague, "Pretty good democracy," in *Proc. 17th International Workshop on Security Protocols. Revised Selected Papers*, 2009, pp. 111–130.

[13] J. Benaloh, "Simple verifiable elections," in *Proc. USENIX/ACCURATE Electronic Voting Technology Workshop (EVT'06)*, 2006.

[14] B. Adida, "Helios: Web-based open-audit voting," in *Proc. 17th USENIX Security Symposium*, 2008, pp. 335–348.

[15] F. Karayumak, M. M. Olembo, M. Kauer, and M. Volkamer, "Usability analysis of helios - an open source verifiable remote electronic voting system," in *Proc. Electronic Voting Technology Workshop / Workshop on Trustworthy Elections (EVT/WOTE'11)*, 2011.

[16] R. Haenni and R. Koenig, *Design, Development, and Use of Secure Electronic Voting Systems*. IGI Global, Mar. 2014, ch. Voting over the Internet on an Insecure Platform.

[17] G. S. Grewal, M. Ryan, L. Chen, and M. Clarkson, "Du-vote: Remote electronic voting with untrusted computers," in *Proc. 28th Computer Security Foundations Symposium Conference (CSF'15)*. IEEE Computer Society, 2015, pp. 155–169.

[18] V. Cortier and B. Smyth, "Attacking and fixing helios: An analysis of ballot secrecy," in *Proc. 24th IEEE Computer Security Foundations Symposium (CSF'11)*. IEEE Computer Society Press, 2011, pp. 297–311.

[19] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène, "Election verifiability for helios under weaker trust assumptions," in *Proc. 19th European Symposium on Research in Computer Security (ESORICS'14), Part II*, 2014, pp. 327–344.

[20] D. J. Smith, *Reliability, Maintainability and Risk, 7th Edition*. Elsevier, 2005.

[21] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," Institute for Theoretical Computer Science, ETH Zürich, Tech. Rep. TR260, 1997.