Du-Vote: Remote Electronic Voting with Untrusted Computers

Gurchetan S. Grewal
School of Computer Science,
University of Birmingham, UK
research@gurchetan.com

Mark D. Ryan
School of Computer Science,
University of Birmingham, UK
m.d.ryan@cs.bham.ac.uk

Liqun Chen
HP Laboratories,
Bristol, UK
liqun.chen@hp.com

Michael R. Clarkson
Department of Computer Science,
Cornell University, US
clarkson@cs.cornell.edu

***Abstract*—Du-Vote is a new remote electronic voting protocol that eliminates the often-required assumption that voters trust general-purpose computers. Trust is distributed in Du-Vote between a simple hardware token issued to the voter, the voter's computer, and a server run by election authorities. Verifiability is guaranteed with high probability even if all these machines are controlled by the adversary, and privacy is guaranteed as long as at least either the voter's computer, or the server and the hardware token, are not controlled by the adversary. The design of the Du-Vote protocol is presented in this paper. A new non-interactive zero-knowledge proof is employed to verify the server's computations.**

Du-Vote is a step towards tackling the problem of internet voting on user machines that are likely to have malware. We anticipate that the methods of Du-Vote can be used in other applications to find ways of achieving *malware tolerance*, that is, ways of securely using platforms that are known or suspected to have malware.

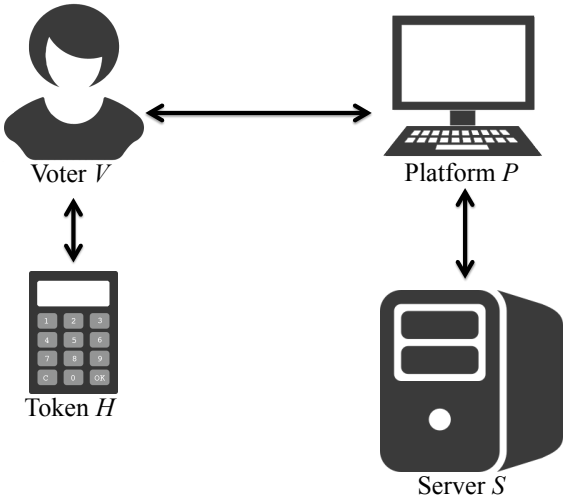

Fig. 1. Du-Vote architecture.

1 Introduction

Electronic voting—especially remote voting over the internet—is very challenging to secure [39], [13], [10], [24], [51], [41], [52]. Nonetheless, internet voting systems are being deployed for national elections [35], [30] and organizational elections [4], [50], [8], [33]. This deployment has been enabled, in part, by improvements in cryptographic protocols that ensure ballot *privacy* as well as *verifiability* of the election outcome, even when the authorities and servers running the election cannot be trusted. Yet these protocols typically assume that the computer used by the voter to submit her vote is trusted for privacy [3], [35], [30], or for both privacy and verifiability [38], [23]. This assumption might seem necessary, because humans cannot compute the encryptions and other cryptographic values necessary to run such protocols. But the assumption is problematic, because a voter's computer might well be controlled by an attacker [27].

Our aim in this paper is to show how attacks by malware on the voter's computer can be avoided, but this is achieved at some cost of usability. More research to improve the user interface of Du-Vote is needed before it could be deployed in real elections.

1.1 Du-Vote

Du-Vote (Devices that are Untrusted used to Vote), introduced in this work, is a protocol (described in Section 4) that eliminates the need for voters to trust their computers. As depicted in Figure 1, a voter V interacts with her computing platform P and a simple hardware token H to cast a vote on server S. The encryption of candidates takes place on P, and the voter's choice of encrypted candidate is made with H, such that neither P nor H learns the voter's plaintext choice. The server S verifies that P is behaving honestly and anyone (e.g. observers and voters) can verify the computations done by S, to check its honesty.

Moreover, Du-Vote requires very little functionality from H. It needs only

- to accept short inputs (e.g., twenty decimal digits) and produce very short outputs (e.g., four decimal digits),
- to store a secret value, similar to a cryptographic key, that can be pre-installed before delivery to the voter, and
- to compute modular exponentiations and multiplications.

H does not need any connection to another computer, so there are no requirements for a general-purpose operating system, drivers, etc. Indeed, Du-Vote requires H to be unable

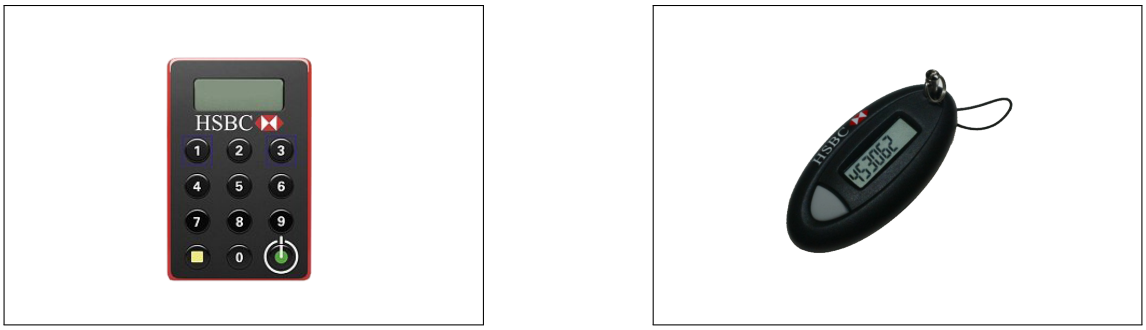

Fig. 2. Hardware tokens used in banking. These devices store secret information into a tamper-resistant hardware. One on the left, has device with screen and keyboard; on the right, a device with screen and just one key.

to communicate with anyone other than V. The only means for H to communicate with the outside world should be by using a built-in keyboard and screen. H can even be *software closed*, such that its software cannot be modified. These requirements can be satisfied by hardware tokens similar to those shown in Figure 2, which are used by banks for two-factor authentication. Such tokens have a decimal keypad, an LCD screen that can display a few digits, and a *command* button that causes the device to compute an output.

Du-Vote is the *front end* of a voting system: it is designed to collect encrypted votes and post them on a bulletin board. The *back end* used to tally the bulletin board is a separate module; it could be instantiated with verifiable reencryption mixnets [16], [36] or verifiable homomorphic tallying [25], [4].

1.2 Trustworthiness of H

Our initial verifiability and privacy analysis is based on the assumption that H is trustworthy. Later, we consider the possibility that the manufacture of H might be controlled by an adversary: for example, the H devices might be made in a foreign nation state. Therefore, we also carry out verifiability and privacy analysis based on the assumption that H is not trustworthy. We briefly discuss these two here:

1.2.1 If token H is trustworthy

Given a trustworthy H, Du-Vote does not need to make any assumptions about P, either for privacy or for integrity. Our analysis (in Section 5) shows that, when H is trustworthy, Du-Vote guarantees verifiability of the election outcome and of individual votes even if both P and S are controlled by the adversary. Du-Vote guarantees privacy of votes if at least one of P and S is not controlled by the adversary. So one contribution of Du-Vote is the relocation of trust from a large computational device (the voter's general-purpose computer) into a small computational device (a token). Although other systems (including SureVote [15], Pretty Good Democracy [47], and Helios [3]—all discussed in Section 7) have similarly worked to relocate trust, Du-Vote is the first to ensure both privacy and verifiability with a trusted computing base that consists only of a hardware token like that used in real-world banking applications. In fact, because of its simple design, the trustworthiness of H could be established through verification and/or testing.

1.2.2 If token H is untrustworthy

Surprisingly, verifiability of the vote is assured with high probability even if the manufacture of H is controlled by an adversary, and P and S are controlled by the adversary in real time. However, to achieve this result, we assume that a certain fraction of the platforms P are trustworthy. Privacy is guaranteed if either P, or H and S, are not controlled by the adversary. Another contribution of Du-Vote is, therefore, a voting system that achieves verifiability and privacy without requiring trust in the computers used by voters. The assumptions we make to achieve privacy are justifiable because voters who are technically competent can protect their privacy by making sure that their own computer (P) is free from any election related malware. The voters who don't trust their computers can rely on trustworthiness of devices (H and S) provided by the election authorities.

1.3 Malware tolerance

Methods to use an untrustworthy general-purpose computer for security-sensitive purposes would be useful in many contexts besides voting. The aim of *malware tolerance* is to prevent some kinds of attack, and to make other kinds detectable, recognising that it is not possible to have perfect platform security. This paper provides malware tolerance for internet voting.

Our contributions: (1) We introduce Du-Vote, a new voting system that addresses the long standing problem of voting using untrusted voting machines and untrusted servers. Du-Vote is the first voting system to ensure both privacy and verifiability with a TCB that consists only of a hardware token like that used in real-world banking applications. (2) We show that even if this TCB is in fact untrusted (Section 5.1.2), it is practically impossible for it to change some voters' votes

without detection. (3) We develop a novel non-interactive zero-knowledge proof (Section 4.6), that enables a server to prove that it correctly selects and reencrypts the ciphertext chosen by the voter, without revealing it.

2 Security Goals

Du-Vote is designed to satisfy the following security properties:

- **Universal and individual verifiability**. Anyone can check that all votes cast are counted, that only authorized votes are counted, and that no votes are changed during counting. Each voter can check that their own vote is included in the tally.
- **Privacy**. The association between voters and votes is not revealed.

Universal and individual verifiability are *integrity* properties. Together, they assure all voters that the election outcome is computed correctly. Any attempt to corrupt the integrity of the election must be detected and correctly attributed. Privacy is a *confidentiality* property. It assures voters that no one can learn how they voted. As shown in section 5, Du-Vote assures these properties with high probability, depending on which machines (H, P, or S) are controlled by the adversary. Verifiability holds even if all machines are controlled. But for privacy either P, or H and S, must remain uncontrolled.

Some election schemes satisfy stronger confidentiality properties, such as *receipt-freeness* [9] or *coercion resistance* [38]. But systems that provide these properties, such as Helios [3] and Civitas [23], require voters' machines to be trusted for confidentiality. It is currently an open problem how to satisfy these confidentiality properties with a completely untrusted voter platform under realisable trust assumptions [32].

In Du-Vote, there is some protection against vote selling and coercion: an adversary who cannot observe V's interaction with H will be unable to determine V's vote, as long as either P or S is not controlled by the adversary. To prevent that observation, we must assume that voters do not give or sell their tokens to the adversary. Unfortunately, any credential used in remote voting—in any scheme, not just Du-Vote—is inherently salable. So to realise that assumption, voters would have to be disincentivised to sell tokens. Associating the token with the voter's legally binding digital signature, as is done in Estonia,[1] might suffice.

3 Voter Experience

We begin by describing Du-Vote from the voter's point of view. No cryptography is involved in this description, in part to illustrate that the voter experience is straightforward. Section 4 details all the cryptographic protocols.

Registration. Voter V registers to vote with the authority running the election. This registration may occur online. V establishes a username and password for server S. The authority issues to V a hardware token H, either in person or by postal mail (perhaps in tamper-evident envelopes).

Voting. Using her computing platform P, voter V authenticates to server S using her voter ID and password. [2] P displays a *code page* to V, as shown in Figure 3.[3]

The instructions in Figure 4 are conveyed to the voter by P. These instructions are also widely publicised—for example, in newspapers, TV channels, and online media—so that they are

Candidate	Column A	Column B
Alice	3837	7970
Bob	6877	2230
Charlie	5525	1415
Dave	9144	2356
Enter your vote code here		

Ballot ID: JSAhVEVYIFRTLXByb2dyYW

Fig. 3. An example code page as displayed by P.

Please follow these instructions to cast your vote:

- Check that your computer has displayed the candidates in alphabetical order and has displayed two codes (in column A and B) for each candidate.
- Flip a coin.
- If the coin landed **heads**:
  - Enter all the codes, from top to bottom, from column A into your token.
  - Find the code for the candidate for whom you wish to vote in column B. Enter that code into your token.
- If the coin landed **tails**:
  - Enter all the codes, from top to bottom, from column B into your token.
  - Find the code for the candidate for whom you wish to vote in column A. Enter that code into your token.
- You should now have entered a total of 20 digits into your token. Press the command button on your token. Your token will display a new, four-digit vote code. Enter that vote code into your computer.
- Record your ballot id and vote code to later double-check that your vote was received.

Fig. 4. Voting instructions, assuming $n = \kappa = 4$. The voter is in possession of a hardware token similar to that of Figure 2(left), and a computer P displaying a code page similar to Figure 3 .

[1] Use of the Estonian National ID card for internet voting is described on Estonia's website at http://vvk.ee/voting-methods-in-estonia/engindex/.

[2] Neither privacy not integrity of the election depend on the security of the password. It is present only to prevent data disruption that could anyway be detected in the verification stage.

[3] Du-Vote's code pages appear superficially similar to Chaum's code sheets [15]. But Du-Vote constructs and uses code pages in a different way, as we describe in Section 4.

common knowledge among voters. Following them, V enters five codes into H, obtains a new *vote code* from H, and enters that vote code into P. Then V records her ballot ID and vote code.

Auditing. A log of all the votes and other data received by S is made available for review by all voters. To double check that her vote was correctly received and recorded, V confirms that her ballot ID and vote code are in that log.

The cryptography behind the curtain. Although voters don't need to know this, the codes on the code page are *truncated* probabilistic encryptions of candidate names—that is, the last four digits of a decimal representation of a ciphertext. P computes those ciphertexts. That leads to three problems, that are solved with the help of H:

- P might try to cheat in constructing the ciphertexts. So V is required to enter an entire, randomly chosen column to H. The vote code output by H is based on that column. S uses the vote code to verify that the column was encrypted correctly.
- P might try to change the order of codes displayed to V. Therefore, the vote code output by H is also based on the order in which V has seen codes in the audit column. This later informs S in what order V saw the codes.
- P might try to learn the voter's plaintext vote. Clearly, since P knows the plaintexts, V cannot reveal its chosen candidate code directly to P. So V makes her selection on H. The vote code output by H is based on that selection. S uses the vote code to recover the voter's encrypted candidate.

Moreover, S is not able to see the candidate inside the voter's vote, because that candidate is encrypted. (As is standard, the encrypted votes can be homomorphically combined or shuffled with a mix network, such that S cannot link decrypted votes with ciphertexts.) How S uses the vote code, and all the other cryptographic details, we explain, next.

4 Voting Scheme

In addition to the principals already introduced (hardware token H, voter computing platform P, and server S), Du-Vote uses a set T of *decryption tellers*, who are principals entrusted with shares of the decryption key under which votes are encrypted. Du-Vote also uses a bulletin board BB and a cryptographic hash function $\mathsf{hash}(\cdot)$.

4.1 Setup

Du-Vote employs a distributed El Gamal encryption scheme [11]. Each decryption teller has a private key z_j and a corresponding share of the public key y_j, where $y_j = g^{z_j}$. The product $\prod_j y_j$ of all the public keys can itself be used as a public key, denoted y. Decryption of a ciphertext encrypted under y requires participation of all the tellers, each using their own private key.

The scheme works over a cyclic group $\mathcal{G}$ with generator g. Encryption of a message m with a public key y using randomness r is denoted $\mathsf{enc}(m; y; r)$, where $\mathsf{enc}(m; y; r) = (g^r, my^r)$. We omit y when it is clear from context. And when the particular random value r is not relevant, we omit it. Decryption of a ciphertext c with a private key z is denoted $\mathsf{dec}(c; z)$. We omit z when it is clear from context.

Du-Vote also employs *exponential* El Gamal encryption [2], in which during encryption another group generator h is raised to the power of the plaintext message. Exponential encryption of message m with public key y using randomness r is denoted $\mathsf{enc\text{-}exp}(m; y; r)$, where $\mathsf{enc\text{-}exp}(m; y; r) = (g^r, h^m y^r)$. Sometimes we omit parameters to $\mathsf{enc\text{-}exp}$, as with enc. Exponential El Gamal [29] satisfies the homomorphic property $\mathsf{enc\text{-}exp}(m_1; y; r_1) \cdot \mathsf{enc\text{-}exp}(m_2; y; r_2) = \mathsf{enc\text{-}exp}(m_1 + m_2; y; r_1 + r_2)$. The security of the scheme relies on the standard DDH assumption, and also that $\log_g h$ is unknown to anyone [46].[4]

4.2 Registration

The electoral roll of authorized voters is determined by the election authority in advance. Each authorized voter V may register to establish a voter ID and password for server S. Du-Vote does not require any particular protocol for this registration.

As part of registration, the election authority issues to V a hardware token H. Each H is configured with its own, unique value K, where $K = y^k$ for a randomly chosen k.[5] At the time H is issued to V, the authority causes server S to record an association between k and V. Note that S can, therefore, derive the K stored on V's token. We assume that H is issued over a private channel.

Given a sequence of decimal digits as input, H interprets that input as an integer d. When the command button is pushed, H outputs $(Kh^d)^*$, where * denotes a *truncation* operator that produces short representations of large values, as follows. Let κ be a security parameter of Du-Vote. Given a large value L, let L^* denote the final κ digits of some canonical representation of L as a decimal integer. Therefore $(Kh^d)^*$ denotes the final κ digits of the decimal representation of group element Kh^d. Here are two other examples of how to form L^*:

- If L is a bit string, then convert that bit string to an integer and yield its final κ decimal digits.
- If L is an El Gamal ciphertext (γ_1, γ_2), first represent (γ_1, γ_2) as a bit string $\gamma_1 || \gamma_2$, where $||$ denotes concatenation, then proceed as in the previous example.

Denote the output $(Kh^d)^*$ of H on input d as $H(d)$.

[4] One way to realize this assumption is based on a *Schnorr group*. Let p and q be large prime numbers, such that $p = \ell q + 1$ for some ℓ. Define Schnorr group $\mathcal{G}$ to be an order-q subgroup of $\mathbb{Z}_p^*$. To construct g and h from $\mathcal{G}$, first choose any two bit strings a and b, such that $a \neq b$. Then let $g = (\mathsf{hash}(a) \bmod p)^\ell$ and $h = (\mathsf{hash}(b) \bmod p)^\ell$. Check that $g \neq 1$ and $g \neq -1$, and likewise for h. Also check that $g \neq h$. If any checks fail (which happens with low probability), then try again with some other values for a and b.

[5] Our notation is meant to suggest keys, and indeed k will be used in Section 4.6 to compute a value that resembles an El Gamal encryption.

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: $\mathsf{hash}(A, B)$

Fig. 5. The cryptographic construction of a code page. Each entry A_i^* denotes the truncation of an encryption $A_i = \mathsf{enc\text{-}exp}(a_i)$ of candidate a_i. Likewise, each B_i^* is also the truncation of an encryption $B_i = \mathsf{enc\text{-}exp}(a_i)$ of a_i. In the ballot ID, value A denotes the lexicographically sorted sequence of ciphertexts $A_1, \ldots, A_4$ in column A, and likewise for B in column B.

Typically, we assume $\kappa = 4$. More digits would require more typing into H, affecting the usability of Du-Vote. Fewer digits would affect the security of Du-Vote by reducing resistance to randomization attacks (discussed in Section 5.1.2, Remark 4) in which H causes V's vote to change to an unpredictable candidate.

4.3 Opening of election

In advance of an election, the set $Cand$ of n candidate names $\{a_1, \ldots, a_n\}$ is published on BB. An algorithm for generating a publicly verifiable but unpredictable *election nonce* I is also declared and published. This election nonce I could be generated by using stock market data [14], [20], [21]. The nonce I must be generated after the hardware tokens H have been manufactured. It is critical for integrity that tokens H are unable to learn I (that is why they have no input interface except the short value keypad).

4.4 Voting: preparation of vote by P

To vote, V first authenticates to S through P using her voter ID and her password. P retrieves candidate names $Cand$ and nonce I from BB.

Second, P prepares a code page for the voter. The essential idea in preparing that code page is that P creates two different encryptions of each candidate name. The two encryptions will be used later to verify that P computed the candidate's encryptions honestly. The construction of a code page with four candidates is shown in Figure 5. Each candidate's name is encrypted once in column A, and again in column B. To defend against an attack (cf. Section 5.1.2, Remark 5) in which a malicious H and P conspire to violate integrity, those encryptions must be chosen by P in a specific way, detailed below. This method is designed to prevent a dishonest P from having a covert channel with which to communicate to a dishonest H.

- P computes a set $\{c_1, \ldots, c_{2n}\}$ of distinct decimal codes determined by I and voter ID, each code having κ digits (recall that n is the number of candidates). To achieve this, P generates a bit stream seeded by $\mathsf{hash}(I, \text{voter ID})$ and repeatedly consumes an appropriate number of bits from it to form each of the codes in turn. If any code would be identical to a previous one, P discards it and produces a new one, until it has the required $2n$ distinct codes.
- P chooses a random α such that $1 \leq \alpha \leq n$ and cyclically assigns the subset $\{c_1, \ldots, c_n\}$ of codes to candidates: c_α to a_1, $c_{\alpha+1}$ to a_2, etc., wrapping around and assigning $c_{\alpha-1}$ to a_n. Let $\mathsf{code}_A(i)$ denote the code assigned to a_i according to α.
- Likewise, P chooses a random β such that $n+1 \leq \beta \leq 2n$ and assigns the subset $\{c_{n+1}, \ldots, c_{2n}\}$ of codes to candidates: c_β to a_1, $c_{\beta+1}$ to a_2, etc. Let $\mathsf{code}_B(i)$ denote the code assigned to a_i according to β.
- For each candidate a_i, platform P chooses a random r and computes $\mathsf{enc\text{-}exp}(a_i; y; r + j)$ for each $j \in \{0, 1, 2, \ldots\}$ until the ciphertext $A_i = \mathsf{enc\text{-}exp}(a_i; y; r + j)$, satisfies $A_i^* = \mathsf{code}_A(i)$. [6]
- Likewise, P finds ciphertexts B_i such that $B_i^* = \mathsf{code}_B(i)$.
- P generates a code page as depicted in Figure 5. This code page also contains a ballot ID where P commits to

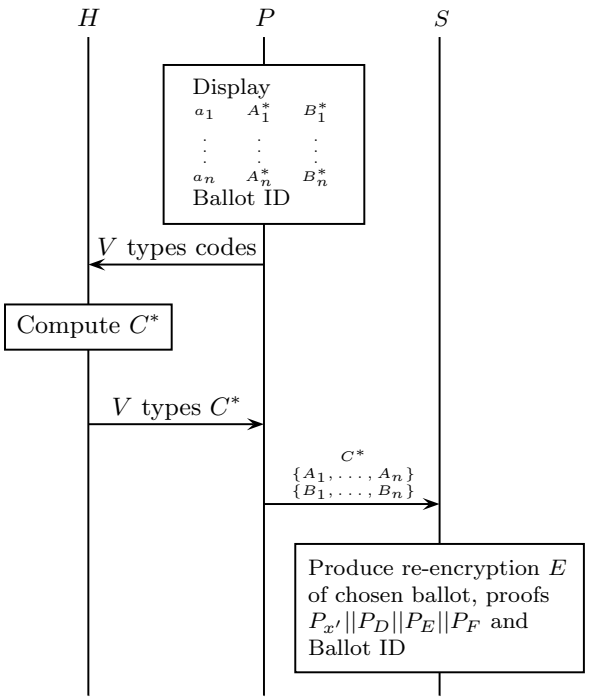

Fig. 6. Du-Vote voting protocol. Platform P displays the candidates and the codes. Token H computes vote code C^* from the codes entered by the voter. The server S later produces re-encryption E of the chosen candidate's encryption and NIZKPs to prove its honesty.

[6] This is efficient, because if $\mathsf{enc\text{-}exp}(a_i; y; r + j) = (\alpha, \beta)$ then $\mathsf{enc\text{-}exp}(a_i; y; r + j + 1)$ can be computed as $(\alpha g, \beta y)$. See section 6. Moreover, it is secure; we prove the security of this variant of Elgamal in Appendix A.

the ciphertexts. The ballot ID is defined as $\mathsf{hash}(A, B)$ where A denotes the lexicographically sorted sequence of ciphertexts $A_1, \ldots, A_n$ in column A, and likewise for B in column B. Due to this ballot ID P cannot later choose different ciphertexts (with different plaintexts) that match the codes.

Third, P invites V to vote with the code page. Figure 6 depicts this part of the protocol. To vote, V follows similar instructions as in Figure 4. For example, if V's coin flip yields tails and V wants to vote for a candidate whose ciphertext is x, then V enters $B_1^*||\ldots||B_n^*||x^*$ into H, where $x^* \in \{A_1^*, \ldots, A_n^*\}$, and presses the command button. Let d be the decimal representation of what the voter enters. In response to the command button, H outputs vote code $H(d)$, which is the final four digits of Kh^d. Let $C^* = H(d)$. The voter enters C^* into P. Finally, P sends C^*, along with $\{A_1, \ldots, A_n\}$ and $\{B_1, \ldots, B_n\}$, to S in lexicographic order of ciphertext.

4.5 Voting: processing of vote by S

First, to verify P's honesty, S checks that P computed the codes in A and B correctly. To do so, S computes bit string $\mathsf{hash}(I, \text{voter ID})$ and—just as P should have—produces a set $\{c_1, \ldots, c_{2n}\}$ of codes from it. Then S checks that A contains the first half of that set, and B the second half (both in some cyclic order). S posts C^*, A, and B on BB. S also generates ballot ID (just as P should have) and posts that on BB as well. V matches this ballot ID with the one shown by P at the time of voting.

Second, S does a brute-force search to determine what code V entered as a candidate choice. The input space for this search is the set of inputs V could have entered into H, given that V's code page was constructed from A and B. Anyone who knows I and voter ID can compute those inputs. The size of the input space is $2n^2$: the full column entered by the voter is 1 of 2 columns; the cyclic permutation of that column starts at 1 of n candidates; and the candidate chosen by the voter is also 1 of n.

Computing the output of H, however, requires knowledge of value K stored in H. Recall (from Section 4.2) that S can derive that value. Therefore, S simulates the voter's token to find the code x^* that produces C^*. Next, S recovers the full ciphertext for that code from the ciphertexts received from A or B; and S determines which column was fully entered by the voter (call that the *audit column*) and post it on BB.

Third, S requests from P the random coins used to encrypt each candidate in the audit column. P checks if S has published the audit column on BB before sending the random coins. S posts those on BB, then checks whether the encryptions for each candidate in that column were computed correctly by P (note that S knows the order in which V has seen the audited codes as voter entered them into H in the order codes were seen). If that check fails (or if the brute force search previously failed), then one or more of P, H, and V is faulty or dishonest. In that case, S refuses to do any further processing of V's vote.

Fourth, S reencrypts x to a new ciphertext E and posts E on BB as V's vote. The reencryption is necessary, because P (having generated the encryptions) knows the plaintext corresponding to x. So if S directly posted x to BB, then P would easily be able to violate V's privacy.

Fifth, S produces a non-interactive zero knowledge proof that it followed all the above protocol honestly by posting a proof on BB. We present that proof, next, in Section 4.6.

Finally, voter V finishes by recording the vote code and the ballot ID, and by logging out of S. V must check that this recorded information is present on BB using an honest platform, or V must convey this information to a trusted third party who does the check on behalf of V. Note that conveying it does not harm V's privacy.

4.6 Voting: S proves its honesty

As mentioned above, S must create universally-verifiable proofs that it correctly selected and reencrypted the ciphertext x chosen by the voter. The proofs will be posted on BB. Here we give the technical details of those proofs. We use standard cryptographic techniques for constructing a non-interactive zero-knowledge proof, based on ring signatures [1], [19] and the Fiat-Shamir heuristic [28].

4.6.1 Ring signatures and OR-proofs

Consider a set $\{P_1, \ldots, P_n\}$ of participants, and suppose that each participant P_i has a private key sk_i and a corresponding public key pk_i. We suppose P_i possesses his own private key sk_i and all of the public keys $\{pk_j\}_{1 \le j \le n}$. A *ring signature* is a cryptographic signature that such a participant P_i can make, and has the property that any verifier in possession of the public keys $\{pk_j\}_{1 \le j \le n}$ can be assured that one of the participants $\{P_1, \ldots, P_n\}$ made the signature, without being able to tell which one it was [1], [19]. In a ring signature, there is one "actual" signer, in our example P_i; the signing process uses his secret key sk_i and the public keys $\{pk_j\}_{1 \le j \le n,\, j \ne i}$ of the other ring members. The actual signer does not need the permission or involvement of the other ring members.

A zero-knowledge proof (ZKP) is a proof between a prover and a verifier that demonstrates to the verifier that a certain statement is true, without revealing any information except the validity of the statement [31]. Non-interactive zero knowledge proofs (NIZKP) are a variant of ZKP that do not require interaction between the prover and the verifier. If the statement being proved asserts that the prover knows a secret signing key, this type of NIZKP can be a signature on some arbitrary value using the signing key, and is called a signature-based proof of knowledge (SPK). The message being signed is not important, it can be a constant value or even blank. A Schnorr signature [49] is commonly used for this purpose.

If the statement being proved asserts that the prover knows a secret value satisfying a disjunction of properties, then a ring signature can be used. For example, given a finite set $\Omega = \{\omega_1, \ldots, \omega_n\}$, a group generator g, and a function f, we

may prove in zero knowledge that we know an x such that $g^x = f(\omega_1)$ or ... or $g^x = f(\omega_n)$. Such a proof is a ring signature on an arbitrary value, where the actual signing key is x and the public keys are $\{f(\omega) \mid \omega \in \Omega\}$. We write such a signature proof of knowledge as

$$SPK\{x \mid \exists\omega : \omega \in \Omega \wedge g^x = f(\omega)\}.$$

It is useful to generalise this to a *double signature*. Let Ω be a set of pairs, and g_1, g_2 be two group generators, and f_1, f_2 be two functions. We write

$$SPK\{x \mid \exists\omega_1, \omega_2 : (\omega_1, \omega_2) \in \Omega \wedge g_1^x = f_1(\omega_1) \wedge g_2^x = f_2(\omega_2)\}$$

to mean a proof of knowledge of x satisfying $g_1^x = f_1(\omega_1)$ and $g_2^x = f_2(\omega_2)$ for some $(\omega_1, \omega_2) \in \Omega$. The signer's private key is x, and the ring of public key pairs is $\{(f_1(\omega_1), f_2(\omega_2) \mid (\omega_1, \omega_2) \in \Omega\}$.

4.6.2 Proof of selection and re-encryption of the voter's chosen ciphertext

Using these ideas, we show how the server S produces a zero-knowledge proof that it correctly computed the voter's chosen ciphertext $x = (x_1, x_2)$ from the input C^* that S receives from P; and that it outputs a correct re-encryption of that x, all without revealing what x is. The proof is divided into three parts.

Proof of selection.

S computes a pair $D = (D_1, D_2) = (g^d, y^d h^{x_2})$, for some value d, where x_2 is the second component of the voter's ciphertext; and proves that D is of this form.

To see how this is achieved, we assume that V's coin flip yielded tails, and V enters $B_1^* \dots B_n^* x^*$ into H, where $x^* \in \{A_1^* \dots A_n^*\}$. S starts as follows:

- retrieve $C = y^k h^{(B_1^* || \dots || B_n^* || x^*)}$ and publish it. The last κ digits of C is C^*, which can be checked by everybody.
- compute $C_{x^*} = C/h^{(10^\kappa B_1^* || \dots || B_n^*)} = y^k h^{x^*}$. This value is publicly known as anybody can compute it from C and $(B_1, \dots, B_n)$.
- compute x' by truncating the last κ digits of x_2. Thus, $x_2 = x' || x^*$.
- compute $C_{x'} = y^{r'} h^{10^\kappa x'}$, and publish it, where r' is chosen randomly.
- prove knowledge of x' and r' in $C_{x'}$. We call this proof $P_{x'}$.
- compute $D = (D_1, D_2) = (g^d, C_{x^*} C_{x'}) = (g^d, y^d h^{x_2})$, where $d = k + r'$; anyone can compute D_2 from C_{x^*} and $C_{x'}$.

These computations demonstrate the relationship between D and C: they show that $D_2 = y^d h^{x_2}$ and $C = y^k h^{B_1^* || \dots || B_n^* || x^*}$ share the same x^*.

To prove the structure of D, S must prove knowledge of d such that $D = (D_1, D_2) = (g^d, y^d h^{x_2})$, for some $x_2 \in \{A_{1,2}, \dots, A_{n,2}\}$ (recall that each El Gamal ciphertext A_i is a pair $(A_{i,1}, A_{i,2})$). This proof can be written as

$$P_D = SPK\{d \mid \exists x_2 \in \Omega : g^d = D_1 \wedge y^d h^{x_2} = D_2\},$$

where $\Omega = \{A_{1,2}, \dots, A_{n,2}\}$. We show that such a proof can be achieved, by writing it equivalently in the following form to match the shape of ring signature proofs above:

$$P_D = SPK\{d \mid \exists x_2 \in \Omega : (gy)^d = D_1 D_2 / h^{x_2}\}.$$

Proof of re-encryption of a ciphertext.

As previously mentioned, the ballot x can't directly go to the bulletin board BB, because P knows the plaintext and could later use BB to see how the voter voted. To address this, S now computes a re-encryption $E = (E_1, E_2) = (g^e x_1, y^e x_2)$ of x (where e is chosen at random).

The proof of the structure of E can be shown using double ring signature:

$$P_E = SPK\{e \mid \exists x_1, x_2 : (x_1, x_2) \in \Omega \wedge g^e = E_1/x_1 \wedge y^e = E_2/x_2\},$$

where $\Omega = \{A_1, \dots, A_n\}$.

Proof that the re-encryption is of the selected ciphertext.

The proof P_E shows that E is the re-encryption of one of the values in $\{A_1, \dots, A_n\}$, but not that it is the one chosen by the voter. To prove that the (x_1, x_2) used in the computation of E matches the code chosen by the voter, S has to show that the x_2 in E is the same as the x_2 in D. To demonstrate this, S computes $F = (F_1, F_2) = (E_1 D_1, E_2 D_2) = (g^f x_1, y^f x_2 h^{x_2})$ where $f = d + e$, and proves that F indeed has this form. The proof that F has the correct form is another double ring signature:

$$P_F = SPK\{f \mid \exists x_1, x_2 : (x_1, x_2) \in \Omega \wedge g^f = F_1/x_1 \wedge y^f = F_2/x_2 h^{x_2}\}$$

where $\Omega = \{A_1, \dots, A_n\}$.

Note that by putting $P_{x'}, P_D, P_E$ and P_F together, we have actually proved that E is a re-encryption of $A_i = (A_{i,1}, A_{i,2})$ where $A_{i,2} = x_2$ and x_2^* is involved in C^*. The proofs $P_{x'}, P_D, P_E$ and P_F are bound together as a single Schnorr signature proof P_{total}. This proof P_{total} is also posted on the bulletin board BB.

Our constructions are novel, but they make use of the standard signature-based proof of knowledge (SPK) technique. By using the Fiat-Shamir [28] heuristic, we are assured that the proof is sound and complete, and has the computational zero-knowledge property.

4.7 Tabulation

The tellers T receive from S the ciphertext E for each voter. The voter's names or identities can be publicly associated with their encrypted vote. This allows any voter to check that their E is included.

The set of encrypted votes is tabulated by a back end. Du-Vote does not require any particular back end. One of these two standard methods from the literature could be used:

- Homomorphic combination, followed by decryption of the combination [4]. Since Du-Vote ballots are exponential El Gamal ciphertexts, multiplication of the ciphertexts corresponds to addition of the plaintext.

- A reencryption mixnet, followed by decryption [38], [23].

4.8 Verification

Voters and observers take the following steps to verify the integrity of the material on BB. As usual, we envisage that a variety of programs produced by different agencies would be available for performing the verification.

During vote processing, S publishes the following information for each vote:

- the voter ID
- the ciphertexts $\{A_1, \ldots, A_n\}$ from column A and $\{B_1, \ldots, B_n\}$ from column B
- the identity (A or B) of the audited column
- the random values $r_1 \ldots r_n$ used for encryption in the audited column
- the ballot ID
- the outputs $C, C_{x'}, D_1, E, P_{\text{total}}$ described in Section 4.6, where P_{total} includes $P_{x'}, P_D, P_E$ and P_F.

Any voter can ensure that her vote is present, by checking that BB contains an entry with her ballot ID and the value C^* that she obtained from the code page displayed by P during voting. Any observer (including any voter) can verify the integrity of the other information, by performing the following BB **verification steps**:

1) Identify the randoms used in the audited column, and check the audited ciphertexts by reconstructing them. Without loss of generality, assume column B is audited and column A is used to cast vote
2) Verify the ballot ID by computing $\mathsf{hash}(A, B)$, where value A denotes the lexicographically sorted sequence of ciphertexts $A_1, \ldots, A_n$ in column A, and likewise B in column B
3) Identify C and check that the last κ digits of C are C^*
4) Compute $C_{x^*} = C/h^{\{10^\kappa(B_1^*||\ldots||B_n^*)\}}$
5) Verify proof $P_{x'}$
6) Identify $C_{x'}$ and compute $D_2 = C_{x^*}C_{x'}$ and $D = (D_1, D_2)$, and verify P_D
7) Identify E and verify P_E
8) Compute F from D and E and verify P_F
9) Check E has been sent to T

5 Verifiability and privacy

Full security analysis of Du-Vote (similar to [26] or [40]) is beyond the scope of this paper. Our contribution is to introduce the novel ideas that allow secure voting to take place even if the backend server and the devices the voter interacts with are all malicious. In this section, we present some results that add detail to this claim.

Du-Vote's primary objective is to ensure integrity and verifiability of the vote, even if the voter's platform is untrusted. Section 5.1.1 is devoted to explaining why any manipulation of a vote by the untrusted P or untrusted S will be detected with high probability, even if S and P are corrupt and controlled by the same attacker, while assuming that H is trustworthy. Additionally in Section 5.1.2 , we consider the possibility that the hardware token H is corrupt too. There, we show that even if all three of S, P and H are corrupt and controlled by the same attacker, we still obtain detection of large scale manipulation under certain reasonable assumptions.

A secondary objective of Du-Vote is to ensure privacy of votes. In Section 5.2.1 and 5.2.2, we assume H is trustworthy, and we argue that privacy of a vote holds even if one of S or P is corrupt and controlled by an attacker, provided the other one is honest. In Section 5.2.3, we consider the case that H is not trustworthy. We show that although a privacy attack is possible, it cannot be conducted on a large scale without being detected.

We take the view that integrity and verifiability of the election is more important than privacy, and that is why our assumptions to guarantee integrity are weaker than those required to guarantee privacy. Our view about the importance of integrity over privacy is justified by considering what happens if one of them fails, while the other is preserved. An election with privacy but no integrity is useless: the outcome is meaningless. But an election with integrity and no privacy, while not ideal, is still useful (indeed, show-of-hands elections in committee rooms are used all the time).

5.1 Verifiability

We divide our analysis into two parts. In the first part, we assume the hardware tokens H are trustworthy. In the second one, we consider the case that their manufacture is controlled by an adversary.

5.1.1 Verifiability analysis assuming honest H & corrupt P, S

We explain why any manipulation of a vote by the untrusted P or S will be detected with high probability, even if P and S are corrupt and controlled by the same attacker.

We make two assumptions. First, we assume that the tokens H are honest. Second, we suppose that the BB *verification steps* detailed in Section 4.8 have been carried out by some voters or observers, and have passed. Our first remark uses the well-known *cut-and-choose* argument:

Remark 1. *Suppose a voter V votes and checks that her ballot ID and vote code is present on BB. With probability at least 50%, P correctly computes the ciphertexts $\{A_1, \ldots, A_n\}$ and $\{B_1, \ldots, B_n\}$ for V.*

Proof Sketch: Suppose P incorrectly computes one or more of $\{B_1, \ldots, B_n\}$. With 50% probability voter's coin toss is tails and she enters $(B_1^*|| \ldots || B_n^*|| x^*)$ into H and sends C^* to S. The server S identifies the audited column and requests the randoms for the ciphertexts corresponding to the audited column $\{B_1, \ldots, B_n\}$, and publishes the randoms, ciphertexts, voter ID and ballot ID on BB. An observer is now able to check that the received ciphertext corresponds to the randoms

and plaintexts in the correct order, and that the ciphertexts are the original ones computed by P corresponding to the commitment in the ballot ID.

■

As a reminder, n is the number of candidates and κ is the length of codes.

Remark 2. *Suppose a voter V votes and checks that her ballot ID and vote code is present on BB. If P correctly computes the ciphertext x corresponding to the code x^* chosen by V, then with probability $\frac{(10^\kappa-1)^{2n^2-1}}{10^{\kappa(2n^2-1)}}$, we have that:*
(a) S correctly computed the chosen code x^*,
(b) S correctly computed the chosen ciphertext x, and
(c) S correctly computed the re-encryption of ciphetext E.
For example, if $\kappa = 4$ and $n = 4$, this probability is 0.9969. Thus, with high probability, a correct encryption of the voter's chosen candidate is submitted to the tellers.

Proof Sketch: Suppose the ciphertexts $\{A_1, \ldots, A_n\}$ and $\{B_1, \ldots, B_n\}$ correctly encrypt the candidates $\{a_1, \ldots, a_n\}$. We prove each of the statements (a), (b), (c) in turn.

(a) Suppose without loss of generality that the voter chose to audit $\{B_1, \ldots, B_n\}$ and vote with $\{A_1, \ldots, A_n\}$; thus she entered $B_1^*||\ldots||B_n^*||x^*$ into H, generating C^*. By the assumption that the BB verification steps pass, the server receives C^* correctly. Next, the server computes possible inputs to H that could have generated C^*. Since the vote codes are pseudorandom, the probability that the voter has chosen inputs to H which result in a unique C^* can be calculated as $\frac{10^\kappa-1}{10^\kappa}\ \frac{10^\kappa-1}{10^\kappa}\ \ldots \frac{10^\kappa-1}{10^\kappa}$ (there are $2n^2-1$ factors), which equals $\frac{(10^\kappa-1)^{2n^2-1}}{10^{\kappa(2n^2-1)}}$. In this case, since there no collisions on C^*, S has no choice but to correctly identify x^*.
(b) Any observer that checks proof P_{total} is assured of the computation of x from x^*.
(c) Any observer that checks proof P_{total} is assured of the computation of re-encryption E from x.

■

Putting these remarks together, we obtain:

Remark 3. *Suppose a voter participates in the election, and later verifies that the bulletin board contains her ballot ID and vote code. Then the probability that her vote has not been altered and will be decrypted as cast is $0.5 \times \frac{(10^\kappa-1)^{2n^2-1}}{10^{\kappa(2n^2-1)}}$.*

In the case $\kappa = 4$ and $n = 4$, this probability is 0.4985. Since the attacker's chance of manipulating a vote is independent of all the others, the probability of him undetectably altering more than a few votes quickly becomes negligible.

5.1.2 Verifiability analysis assuming corrupt H, P, S

Even if all three of H, P and S are corrupt and controlled by the same attacker, we still obtain detection of large-scale vote manipulation under some reasonable assumptions.

The first assumption is that H has no wireless or wired interfaces; the only way for it to communicate with the outside world is via its keypad and screen. The attacker controls S and P in real time; he can communicate directly with them. But his ability to control H is limited to manufacture time. The attacker can choose the firmware and other data that is installed on H, but once H has been manufactured he can no longer communicate with it because of the lack of wired or wireless interfaces. The idea is that H is prevented from learning the election nonce I, and this prevents it from being useful to the attacker. H's keyboard does provide a very low-bandwidth channel by which the attacker can try to communicate with it, but that channel has far too low bandwidth to communicate the high-entropy I.

The second assumption is that, although the attacker can spread malware to voters' machines and thus control the P that they execute, it cannot spread malware to all the machines. Some PCs are well-looked after and some users are cautious enough to avoid downloading malware.

Some of the H's may be programmed correctly and others may be programmed maliciously, in a proportion chosen by the attacker. But we assume he cannot effectively match dishonest P's with dishonest H's, because the malware distribution channels are completely different from the hardware distribution channels. This means that some honest P's will be paired with dishonest H's; and we assume those ones represent a fraction h of all the P's ($0 \leq h \leq 1$).

As before, we also assume that the voters have checked that their vote code and the ballot ID are present on BB by using a trusted platform and that BB *verification steps* detailed in Section 4.8 have been carried out by some voters or observers, and have passed.

To obtain our result, we note that there are precisely two ways in which a dishonest H could contribute to an integrity attack:

- H could produce a random output instead of the correct one.
- H could produce an output calculated according to some attack strategy that has been built into it at manufacture time. (We call such attacks substitution attacks and discuss them later.)

Intuitively, a random output will be very likely to be detected, because S will fail to find a value x^* that matches one of the available ciphertexts. We formalise this intuition in the following remark.

Remark 4. *If a dishonest H modifies a vote by replacing the code x^* for a voter's chosen candidate with a random x', it gets detected with probability $1-\{(n-1)/(10^\kappa-(n+1))\}$ by S. If $n = 4$ and $\kappa = 4$, this probability is $1-3/9995 \approx 0.9996$.*

Proof Sketch: Suppose without loss of generality that the voter chose to audit $\{B_1, \ldots, B_n\}$ and vote using $\{A_1, \ldots, A_n\}$; thus she entered $(B_1^*||\ldots||B_n^*||x^*)$ into H, where $x^* \in \{A_1^*, \ldots, A_n^*\}$. The dishonest H chooses x'^* randomly instead of x^* and generates C'^*. To succeed in substituting codes, H has to guess x'^* s.t. $x'^* \in \{A_1^*, \ldots, A_n^*\} - \{x^*\}$. Device H already knows $n+1$ codes of κ digits that can't be repeated. So, the probability of getting a correct code is

$(n-1)/(10^\kappa-(n+1))$. If $x'^* \notin \{A_1^*,\ldots,A_n^*\}-\{x^*\}$ then C'^* will not match as expected by S and the attack will be detected with probability $1-\{(n-1)/(10^\kappa-(n+1))\}$. (Recall that S publishes the set of values $C, C_{x'}, D_1, E, P_{\text{total}}$, and by our trust assumption, they are verified by some voters and observers.)

■

Substitution attacks: Instead of relying on H to replace the code x^* with a random value (as done in previous remark), an attacker can have strategically placed some data onto H. Such data allows dishonest P to choose code sequences that covertly trigger malicious behaviour within H. The attacker can arrange it so that H produces an output corresponding to a choice which is valid but different from the one made by the voter. We call such attacks as *substitution attacks*. For instance, P and H can agree to assign a code, say c_1, to one of the candidates in one column and, say, c_2 to attacker's chosen candidate in the other column. While voting, if code c_1 appears then H substitutes V's chosen candidate's code with the code c_2.

The risk for the attacker mounting substitution attacks arises when an honest P inadvertently triggers the malicious behaviour, resulting in H substituting the voter's choice for an invalid one. This will be manifested by some voters having their votes rejected. Considering the above example, if an honest P assigns code c_1 to one of the candidates in a column but code c_2 is not assigned to any candidate in the other column, then if H replaces V's code then the resulting vote will be rejected. We formalise this intuition in the following remark.

Remark 5. *Suppose a fraction h of the platforms P are honest. The server S and dishonest P's are all controlled by the same attacker, who also manufactured the devices H. In order to modify on average N votes, the attacker will on average cause $R\times N$ voters to have their votes rejected, where*

$$R = \frac{h\ (1-n/10^\kappa)\ 10^\kappa}{((1-h)n+h)\ n}$$

For example, if $h = 0.5$ and $n = \kappa = 4$, then $R \approx 500$.

Proof Sketch: Suppose the attacker has configured dishonest H with data that allows P to trigger a substitution attack. Such data can be represented as a series of pairs of the form (z, Q) where z is the substitute code, and Q is a predicate on the input received by H which determines whether the substitution is triggered. Thus, if H receives input satisfying the predicate Q, then H should substitute z for the voter's chosen code and compute its response accordingly. The predicate Q can express a set of codes to use and/or a particular order in which to use them. We write codes(Q) to mean the set of codes in inputs that satisfy Q. We suppose the attacker has chosen ℓ lines, $(z_i, Q_i)_{1\le i\le \ell}$. Without loss of generality, we assume that the sets codes(Q_i) are all disjoint (if they are not, it introduces ambiguity about how P should behave if the codes it should use match several sets Q_i). We also suppose that the codes of Q_i satisfy the predicate Q_i only in one particular order (Allowing more than one order of those codes would increase the risk of an honest P inadvertently triggering the attack.).

Suppose P has calculated the allowed codes based on I, and the voter has chosen whether to vote using the left or right column. A dishonest P had the possibility to choose to trigger the attack if code z_i appears in the "vote" column and Q_i is true of the "audit" column. The first of these events occurs with probability $n/10^\kappa$. The second one occurs with probability, say, p. Because the two probabilities are independent, their conjunction occurs with probability $np/10^\kappa$. Because these joint events are exclusive for different values of i, we have that P had with probability $np\ell/10^\kappa$ the possibility to trigger the attack.

As mentioned (and unfortunately for the attacker), an honest P may also trigger the attack. This will happen if it inadvertently chooses codes that match the codes of some Q_i (this occurs with probability p), and puts in the order of Q_i (since there is only one order, this occurs with probability $1/n$). If this inadvertent triggering happens, it will cause the voter's vote to be rejected if the code z_i is not a valid code; this occurs with probability $1-n/10^\kappa$. Thus, the proportion of votes that the attacker causes to be rejected because of honest P's is $(p\ell/n)(1-n/10^\kappa)$.

If an honest P inadvertently triggers the attack, the attacker may be lucky because the substituted code turns out to be a a valid code. The probability of this combination of events is $(p\ell/n)(n/10^\kappa)$.

Recall that the proportion of honest P's is h. Thus, the proportion of votes the attacker inadvertently causes to be rejected is R times as many as he successfully changes, where

$$R = \frac{h(p\ell/n)(1-n/10^\kappa)}{(1-h)np\ell/10^\kappa + h(p\ell/n)(n/10^\kappa)} = \frac{h\ (1-n/10^\kappa)\ 10^\kappa}{((1-h)n+h)\ n}$$

■

Even if the proportion h of honest P's is much lower, say 10%, still about $R = 70$ times more votes will inadvertently be rejected than those successfully modified.

5.2 Ballot privacy

If an attacker controls P and S together, it can easily find how a voter voted, even if H is honest. P can leak the link between candidates and codes to S. Nonetheless, we argue that an attacker that controls only one of P or S cannot discover how a voter voted.

5.2.1 Privacy analysis assuming honest S, H & corrupt P

We claim that if S is honest, an attacker who controls P cannot find how a voter voted. An attacker who controls P has all the ciphertexts and their randoms and plaintext, together with the value C^* entered by the voter, and the set of values C, $C_{x'}$, D_1, E, P_{total} published by S. The values C^*, C, $C_{x'}$,

D_1, E are determined pseudorandomly from the high entropy value K and randoms chosen by S that are not available to the attacker. Therefore these values do not reveal the vote.

The proof P_{total} is a non-interactive zero-knowledge proof (NIZKP). Such a NIZKP does not reveal the secret value that is the knowledge to be proved to its verifier; therefore, P is not able to learn how the voter voted. Our use of a signature-based proof of knowledge (SPK) technique, by using the Fiat-Shamir heuristic, to implement the NIZKP, this ensures universal verifiability while preserving user privacy.

Pass [45] argued that in the common reference string model or the random oracle model non-interactive zero-knowledge proofs do not preserve all of the properties of zero-knowledge proofs, e.g. they do not preserve deniability from the prover, although the proof does not reveal the secret value to the verifier. However deniability from a prover is not required in our case.

5.2.2 Privacy analysis assuming honest P, H & corrupt S

We claim that if P is honest, then an attacker who controls S can't find how a voter voted. The candidates $a_1, \ldots, a_n$ are encrypted by P with the teller's public key y and the server S does not know the corresponding private key z. (We show that our new variant of Elgamal is secure, in Appendix A.) Moreover, P being honest does not reveal the link between candidates and codes for the column that voter has used for voting. Therefore, S has a ciphertext representing V's choice, but no other information about it to find how V voted.

5.2.3 Privacy analysis assuming corrupt H

Assuming H has no wireless or wired interfaces, the situation for privacy is same in the case when P is honest and dishonest H and S are controlled by the attacker as the case that H is honest. However, if H and P are controlled by the same attacker then H could leak some information to P via the κ digit code C^* that V enters into P. For example, instead of performing the correct computation, H could just output the last entered code (which belongs to the chosen candidate). This violates the voter's privacy as the attacker will learn how the voter voted. However, if the attacker performs this attack on a large scale then this misbehaviour would be detected, as S will reject the vote and a lot of rejections will lead to an enquiry to find what went wrong.

6 Discussion

Performance: One might think that the search that platform P has to perform to find ciphertexts matching a given code is inefficient. On average, the platform P has to compute 10^κ El Gamal ciphetexts in order to find one that matches the code. If $\kappa = 4$, this is 10,000 ciphertexts. However, as explained, P does not have to compute each of these from scratch. It computes the first one from scratch, and then from a ciphertext (α, β) it computes the next one as $(\alpha g, \beta y)$. Hence, to find a ciphertext matching the code, P computes one ciphertext (i.e. two exponentiations and a multiplication) followed by (on average) 10^κ multiplications.

We conducted a simple experiment to see how long this would take in practice. We programmed the relevant part of the platform P in Python, and we used a Macbook Air computer. Once again, we assume $\kappa = 4$. For a given plaintext and code, finding a ciphertext that matches the code took on average 0.24 seconds. (The worst case in our sample of 1000 trials was 1.72 seconds.) If there are $n = 4$ candidates, then 8 codes are required; thus, the platform P computes the entire code page in $8 * 0.24 < 2$ seconds on average.

Re-usability of token H: In order for H to be reused for multiple elections, a different key K must be used each time. To see the necessity of this requirement, we show that an adversary who controls P could manage to learn K over the course of two elections using the same K. Since P knows $2n$ candidates for x^*, and since $C_{x^*} = Kh^{x*}$ is publicly known, adversary can compute $2n$ candidates keys for K, as C_{x^*} $(h^{x^*})^{-1}$. If K is re-used in two elections, he can intersect the set of $2n$ candidates from the first election with the $2n$ candidates from the second election, and will likely find only one value in common, namely real key K.

Therefore, a different key K must be used for each election. A naive way to achieve this is to store multiple values K_1,K_2, ..., K_n, each one dedicated to a particular election. However, this would require fixing the number of elections in advance. To overcome this restriction, we suppose that the server S has a fixed set of keys $k_1, ..., k_\ell$ for each voter, and H has the public counterparts $K_i = y^{k_i}$. For each election, a set of ℓ short random values $e_1, \ldots, e_n$, are publicly announced representing a kind of election ID. Each user programs their H by typing these values into H. H computes a new election key $K = \prod_{1 \le i \le \ell} K_i^{e_i}$, and the server computes the voter's key as $k = \sum_{1 \le i \le \ell} e_i \cdot k_i$.

Recovering key from H: An attacker might try to recover the key K stored in H, e.g. by solving Hidden Number Problem [5]. However, such attack would require very high number of calls which would be impractical (and H could be rate-limited). Even if an attacker manages to recover the key, it affects privacy (not integrity), and it requires the co-operation of the voter.

Error-detecting codes: To detect typing mistakes and to provide friendly feedback to the user, error-detecting codes could be employed. For example, the Luhn algorithm [42] could be used by P to add a single digit checksum after each candidate's code in both columns A and B. Instead of entering κ digits for each candidate, V would enter $\kappa+1$ digits. Before computing C^*, token H would check whether all checksum were correct; if not, H would ask V to retype them.

Error recovery: Several kinds of errors can arise, which could result in a voter's vote being rejected. Some errors arise because of the voter's mistakes (errors in inputs, for example); others could arise because of malicious behaviour by P, H or S. Another possible error could occur if during the search for the voter's input to H, the server S finds that

there are more than one possible input (a clash due to the small number of digits in H's output). Unfortunately, it is difficult to put error recovery procedures in place without compromising the security properties. In particular, a situation in which the platform P starts again could allow it to cheat undetectably. Therefore, if any of these kinds of errors arise, the voter is forced to abandon the attempt to vote and use an alternative mean (such as polling station voting). We hope to improve this situation in further work.

Malicious S: One might think that a malicious S could ignore any malicious behaviour it detects from P. But S is expected to create a public audit trail on BB. If many voters or observers detect cheating behavior in that audit trail, the collusion between S and P will be detected. If S wants to maintain its reputation as an honest voting service, it cannot engage in large-scale, detectable attacks.

Limitation on the number of candidates: Du-Vote does not place an upper bound on the number of candidates in an election. But, from a usability perspective, it would be tedious for voters to enter a long string of digits into H. This problem can be addressed by changing the set of codes that are typed into H. Currently, the user types all n codes of the audit column, plus one code for her vote. But alternative arrangements could require the user only to type some of the audit codes. We leave this issue, and other investigations into usability, as future work.

7 Related Work

Haenni and Koenig [34] propose a trusted device for voting over the internet on an untrusted platform. In their system, the platform generates encryptions in the form of barcodes, the voter makes her selection on the trusted device by scanning chosen candidate's barcode and later transfers her vote to a trusted computer. That device requires a camera, matrix barcode reader, and uplink to a computer. Compared to Du-Vote's hardware token, that device is complicated and difficult to make trustworthy. Their hardware device learns voter's vote. Moreover, a camera built-in into untrusted platform may record a voter's behaviour while she scans her chosen candidate's barcode. In Du-Vote voters do not need to trust hardware token for privacy.

Civitas [23], based on a scheme by Juels et al. [38], provides coercion resistance but requires voters to trust their computers for both integrity and privacy. Neumann et al. [43], [44] extend Civitas with a smart card to manage voter credentials [23]; that scheme does not protect the voter's privacy from an untrusted machine. Several other variants of Civitas improve the usability of the aspects related to verifiability [12], [48] and coercion-resistance [22], but they still rely on trusted machines.

Chaum et al. [18] introduce a *computational assistant* to verify the vote cast on a voting machine inside a polling station. This system is designed for polling-place voting, whereas Du-Vote is suitable for remote voting.

SureVote [15] uses the voter's computer as an untrusted communication channel. Voters receive a *code sheet* through a trusted channel, such as physical mail. The computer is used only to send and receive codes from the sheet. SureVote ensures security from the untrusted computer, but voters cannot verify the construction of code sheets. Integrity turns out to depend on secrecy of the code sheets in SureVote, but Du-Vote does not require code pages to be secret. Pretty Good Democracy (PGD) [47] improves SureVote by ensuring stronger integrity properties. But in PGD, leakage of codes could undetectably undermine integrity.

Helios [3], [4] uses a *cast-xor-audit* technique to assure integrity of ballot construction. A voter can either cast her ballot or audit it. Auditing challenges the computer to prove it acted honestly in encrypting her candidate choice. But to verify that proof, the voter must seek out a trustworthy computer. Du-Vote does not require a trustworthy computer. In Helios, the voter's computer automatically learns how the voter voted, thus violating privacy. Du-Vote guarantees privacy as long as either P is honest, or H and S are honest. This is better because voters who are technically competent can protect their privacy by making sure that their own computer is free from any election-specific malware. Voters who don't trust their computers can rely on trustworthiness of the devices (H and S) provided by the election authorities.

Remotegrity [53] extends the Scantegrity II [17] paper-ballot voting system to use for remote voting. In Remotegrity, a voter receives a paper ballot and an *authorization card* by mail. Part of the ballot is printed with invisible ink, and parts of the authorization card are covered with scratch-off coating. A voter marks the ballot with a special pen, revealing a previously invisible code for a candidate. She then scratches off a field on the card, revealing a previously hidden authentication code. As part of the voting protocol, she submits both the candidate and authentication code to a bulletin board. Remotegrity assumes a secure means of distributing the ballots and authorization cards. Since Du-Vote needs to securely distribute a hardware token that is used to authenticate votes, Remotegrity could be seen as a paper-based analogue of Du-Vote. Both systems solve the problem of untrusted computing platforms, but Du-Vote does so without relying on paper, invisible ink and scratch-offs, or auditing of printed ballots (which is required by Scantegrity II and necessitates use of a random beacon and a distinguished group of auditors).

8 Concluding Remarks

Decades of research have concentrated on securing the back end of voting systems, while assuming that the front end—the voter's computer—is trusted. Unfortunately, those computers are usually untrustworthy. So we set out to design a voting system that would distribute trust between a voter's computer and a minimal token that could be made trustworthy. We therefore eschewed designs that would have required tokens to have cameras, GUIs, or general-purpose operating systems.

The main challenge we consequently encountered was how to minimize the amount of information a human would have to convey between the token and computer. Typing of short codes presented an attractive tradeoff between security and usability. We discovered that the token need not be trusted for integrity, provided that the adversary cannot communicate with it after the election opens.

Could we replace Du-Vote's special-purpose tokens with smartphones? Users might download a simple app that simulates the Du-Vote token. Preventing untrustworthy smartphones from conspiring with other computers to violate verifiability and privacy—especially preventing communication with the adversary—would be challenging, as would be distributing the secret values stored by tokens. Du-Vote's security degrades gracefully when the token is malicious, so it might be possible to address these concerns. If so, smartphones could be profitably employed to improve usability.

Conventional wisdom used to hold that remote voting cannot be secure, because voters' computers themselves are not secure [37]. Research in the last decade, though, indicates that voters do not need fully trustworthy computers. With voter-initiated auditing [6], [7], computers prove to voters that votes are properly encrypted—but voters are instructed to seek out many computers, so that at least one is likely to be honest, and computers must still be trusted for privacy. Du-Vote doesn't require voters to trust general-purpose computer for integrity, nor does it require voters to trust computers for privacy. We are now optimistic, based on the techniques developed in this work, that secure remote voting systems can be deployed without requiring any trust in the computers used by voters.

Further work One of the more pressing topics for future work is a systematic security analysis of Du-Vote. This is likely to be rather complex, because of the non-standard cryptography we use, as well as the probabilistic nature of the security guarantees. It is beyond the scope of this paper.

Acknowledgements

We thank Veronique Cortier, Flavio Garcia, Rui Joaquim, Steve Kremer, Mihai Ordean, Olivier Pereira, Ron Rivest, Peter Roenne, Peter Ryan, and Poorvi Vora for discussions about this work; and the anonymous reviewers for their comments. We are thankful to EPSRC for supporting this work through the projects "TVS: Trustworthy Voting Systems" (EP/G02684X/1), and "Analysing Security and Privacy Properties" (EP/H005501/1). Clarkson was supported in part by AFOSR grants FA9550-12-1-0334 and FA9550-14-1-0334, NSF grant 1421373, and by the National Security Agency. This work was performed in part at George Washington University.

References

[1] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n Signatures from a Variety of Keys. In *ASIACRYPT '02*, pages 415–432, London, UK, UK, 2002. Springer-Verlag.

[2] Ben Adida. *Advances in Cryptographic Voting Systems*. PhD thesis, Massachusetts Institute of Technology, August 2006.

[3] Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.

[4] Ben Adida, Olivier Pereira, Olivier De Marneffe, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In *EVT/WOTE*, 2009.

[5] Adi Akavia. Solving hidden number problem with one bit oracle and advice. In *Advances in Cryptology - CRYPTO09*, pages 337–354, 2009.

[6] Josh Benaloh. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, 2006. USENIX Association.

[7] Josh Benaloh. Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, Berkeley, CA, USA, 2007. USENIX Association.

[8] Josh Benaloh, Serge Vaudenay, and Jean-Jacques Quisquater. Final report of IACR Electronic Voting Committee. International Association for Cryptologic Research, 2010.

[9] Josh Daniel Cohen Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, 1987.

[10] Matt Blaze, Arel Cordero, Sophie Engle, Chris Karlof, Naveen Sastry, Micah Sherr, Till Stegers, and Ka-Ping Yee. Source code review of the Sequoia voting system. In *Report commissioned as part of the California Secretary of State's Top-To-Bottom Review of California voting systems*, July 20, 2007.

[11] Felix Brandt. Efficient Cryptographic Protocol Design Based on Distributed El Gamal Encryption. In Dongho Won and Seungjoo Kim, editors, *ICISC*, volume 3935 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2005.

[12] Sergiu Bursuc, Gurchetan S. Grewal, and Mark D. Ryan. Trivitas: Voters directly verifying votes. In *VOTE-ID*, pages 190–207, 2011.

[13] Joseph A. Calandrino, Ariel J. Feldman, J. Alex Halderman, David Wagner, Harlan Yu, and William P. Zeller. Source code review of the Diebold voting system. In *Report commissioned as part of the California Secretary of State's Top-To-Bottom Review of California voting systems*, July 20, 2007.

[14] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II municipal election at Takoma Park: The first e2e binding governmental election with ballot privacy. In *USENIX Security Symposium*, pages 291–306, 2010.

[15] David Chaum. Surevote: Technical overview. In *Proceedings of the Workshop on Trustworthy Elections (WOTE01)*, 2001.

[16] David Chaum. Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms. In Dimitris Gritzalis, editor, *Secure Electronic Voting*, volume 7 of *Advances in Information Security*, pages 211–219. Springer, 2003.

[17] David Chaum, Richard T. Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y A Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II: End-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE Transactions on Information Forensics and Security*, 4(4):611–627, December 2009.

[18] David Chaum, Alex Florescu, Mridul Nandi, Stefan Popoveniuc, Jan Rubio, Poorvi Vora, and Filip Zagórski. Paperless Independently-Verifiable Voting. *E-Voting and Identity*, pages 140–157, 2012.

[19] Liqun Chen, Hans Lhr, Mark Manulis, and Ahmad-Reza Sadeghi. Property-Based Attestation without a Trusted Third Party. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *ISC*, volume 5222 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2008.

[20] Jeremy Clark, Aleksander Essex, and Carlisle Adams. Secure and observable auditing of electronic voting systems using stock indices. In *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*, pages 788–791. IEEE, 2007.

[21] Jeremy Clark and Urs Hengartner. On the use of financial data as a random beacon. *IACR Cryptology ePrint Archive*, 2010:361, 2010.

[22] Jeremy Clark and Urs Hengartner. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In George Danezis, editor, *Financial Cryptography*, volume 7035 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2011.

[23] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368. IEEE Computer Society, 2008.

[24] Véronique Cortier and Ben Smyth. Attacking and Fixing Helios: An Analysis of Ballot Secrecy. In *CSF*, pages 297–311, 2011.

[25] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election scheme. In *EUROCRYPT*, pages 103–118, 1997.

[26] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.

[27] Saghar Estehghari and Yvo Desmedt. Exploiting the Client Vulnerabilities in Internet E-voting Systems: Hacking Helios 2.0 as an Example. In *EVT/WOTE'10*, pages 1–9, Berkeley, CA, USA, 2010. USENIX Association.

[28] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology – CRYPTO86*, pages 186–194. Springer, 1987.

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

[30] Kristian Gjosteen. The Norwegian Internet Voting Protocol. In *VOTE-ID*, 2011.

[31] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[32] Gurchetan S. Grewal, Mark D. Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. Caveat Coercitor: coercion-evidence in electronic voting. *IEEE Security and Privacy Symposium*, 2013.

[33] Stuart Haber, Josh Benaloh, and Shai Halevi. The Helios E-voting demo for the IACR, 2010. Available at http://www.iacr.org/elections/eVoting/heliosDemo.pdf.

[34] Rolf Haenni and Reto E. Koenig. Voting over the Internet on an Insecure Platform. In *Design, Development and use of Secure Electronic Voting Systems*. IGI Global, 2013.

[35] Sven Heiberg, Peeter Laud, and Jan Willemson. The Application of I-Voting for Estonian Parliamentary Elections in 2011. In *VOTE-ID*, 2011.

[36] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In *Proceedings of the 11th USENIX Security Symposium*, pages 339–353, Berkeley, CA, USA, 2002. USENIX Association.

[37] David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. A security analysis of the secure electronic registration and voting experiment (SERVE), January 2004. Available at http://www.servesecurityreport.org/.

[38] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine, editors, *WPES*, pages 61–70. ACM, 2005.

[39] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an Electronic Voting System. In *IEEE Symposium on Security and Privacy*, pages 27–, 2004.

[40] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *ESORICS*, pages 389–404, 2010.

[41] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash Attacks on the Verifiability of E-Voting Systems. In *IEEE Symposium on Security and Privacy*, pages 395–409. IEEE Computer Society, 2012.

[42] P. Luhn. Computer for verifying numbers, August 23, 1960. US Patent 2,950,048.

[43] Stephan Neumann, Christian Feier, Melanie Volkamer, and Reto Koenig. Towards a Practical JCJ/Civitas Imple-

mentation. Cryptology ePrint Archive, Report 2013/464, 2013.

[44] Stephan Neumann and Melanie Volkamer. Civitas and the Real World: Problems and Solutions from a Practical Point of View. In *ARES*, pages 180–185. IEEE Computer Society, 2012.

[45] Rafael Pass. On deniability in the common reference string and random oracle model. In *CRYPTO*, pages 316–337, 2003.

[46] Torben P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[47] Peter Y. A. Ryan and Vanessa Teague. Pretty Good Democracy. In *Security Protocols Workshop*, pages 111–130, 2009.

[48] Michael Schläpfer, Rolf Haenni, Reto E. Koenig, and Oliver Spycher. Efficient Vote Authorization in coercion-resistant internet voting. In *VOTE-ID*, pages 71–88, 2011.

[49] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[50] Princeton University. Helios Princeton Undergraduate Elections. https://princeton.heliosvoting.org. Last accessed on 15 July 2013.

[51] Scott Wolchok, Eric Wustrow, J. Alex Halderman, Hari K. Prasad, Arun Kankipati, Sai Krishna Sakhamuri, Vasavya Yagati, and Rop Gonggrijp. Security analysis of India's electronic voting machines. In *ACM CCS*, pages 1–14, 2010.

[52] Scott Wolchok, Eric Wustrow, Dawn Isabel, and J. Alex Halderman. Attacking the Washington, D.C. Internet Voting System. In *Proc. 16th Intl. Conference on Financial Cryptography and Data Security*, 2012.

[53] Filip Zagórski, Richard T. Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L. Vora. Remotegrity: Design and use of an end-to-end verifiable remote voting system. In *ACNS13*, pages 441–457, 2013.

Appendix

Du-Vote requires the platform to find ciphertexts for which, when written in decimal form, the last κ digits are equal to certain codes. One might be concerned that adding such a requirement may imply that the encryption scheme is insecure. To alleviate such fears, in this section we prove that the encryption scheme is IND-CPA.

Definitions. Let Encrypt0 be standard El Gamal, and Encrypt1 be our variant. Formally, they are defined as follows:

```
Encrypt0(m,y) = // standard El Gamal
   random r;
   (c1,c2) = (g^r, m.y^r);
   output (c1,c2)

Encrypt1(m,y,code) =   // our variant
   new r;
   (c1,c2) = (g^r, m.y^r);
   while c2* != code {
        (c1,c2) = (c1*g, c2*y)
        }
   output (c1,c2)
```

Recall that Encrypt0(m, y) satisfies IND-CPA. Thus, we may consider a typical CPA game between a simulator and an attacker. In such a game, the attacker receives the public key y from the simulator; the attacker sends a pair of challenge messages (m_0, m_1) to the simulator; the simulator chooses a random bit b, encrypts m_b and returns the ciphertext; the attacker outputs b'. If $b' = b$, the attacker wins the game.

Assumption. Let m and y be fixed, and i be a randomly chosen q-bit value. Then $(m.y^i)^*$ is uniformly distributed in the space of κ-digit decimal numbers.

Theorem 1. *Under our assumption, Encrypt1*$(m, y, code)$ *satisfies IND-CPA.*

Proof: Suppose Adversary A has a target of breaking Encrypt0(m, y). If there exists Adversary B who is able to break Encrypt1$(m, y, code)$, then A can use B to break Encrypt0(m, y).

We set up two games. Game 1 is between a simulator S and the adversary A (playing as an attacker), and Game 2 is between A (playing as a simulator) and B (playing as an attacker).

- In Game 1, A gets the El Gamal public key y from S.
- In Game 2, A forwards y to B.
- In Game 2, A receives a pair of challenge messages (m_0, m_1) from B.
- In Game 1, A forwards (m_0, m_1) to S.
- In Game 1, A receives $(c_1, c_2) = (g^r, m_b \cdot y^r)$, where $b \in \{0, 1\}$, from S.
  - A searches for a value i satisfying $(c'_1, c'_2) = (c_1.g^i, c_2.y^i)$ and $c'^*_2 = code$.
  - A terminates the games if such a value doesn't exist.
- In Game 2, A sends (c'_1, c'_2) to B.
- In Game 2, B returns b' to A.
- In Game 1, A returns b' to S.

If A succeeds in finding the value i, the probability of A winning Game 1 is equal to the probability of B winning Game 2.

Let us now calculate the probability that A succeeds in finding the value i. For a given value of r and i, the probability that $(m.y^{r+i})^* = code$ is $1/10^\kappa$ (here we rely on the assumption). The probability that $(m.y^{r+i})^* \neq code$ is, therefore, $1 - 1/10^\kappa$. We have $2^{|q|}$ possible values of i, so the probability of none of them satisfying the equation is $(1-1/10^\kappa)^{2^{|q|}}$. Therefore the probability that there is at least one i that satisfies the equation is $\pi = 1 - (1 - 1/10^\kappa)^{2^{|q|}})$.

The probability that A wins Game 1 is π times the probability of B winning Game 2. The value $1 - \pi$ is negligible in q, so the theorem follows.