Computer Communications xxx (2009) xxx–xxx

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Simple and efficient hash-based verifiable mixing for remote electronic voting

Francesc Sebé [a,*], Josep M. Miret [a], Jordi Pujolàs [a], Jordi Puiggalí [b]

[a] Department of Mathematics, Universitat de Lleida, Av. Jaume II 69, E-25001 Lleida, Spain
[b] Scytl Secure Electronic Voting, C. Tuset 20, E-08006 Barcelona, Spain

ARTICLE INFO

Article history:
Received 26 May 2009
Received in revised form 5 November 2009
Accepted 12 November 2009
Available online xxxx

Keywords:
Cryptography
Electronic voting
Remote voting

ABSTRACT

Remote voting permits an election to be carried out through telecommunication networks. In this way, its participants are not required to physically move to the polling place. Votes are automatically collected and counted so that once the election ends, the results can be published after a very short delay. Security is a key aspect of any remote application. A remote voting system must be secure in the sense that the result of the election cannot be manipulated and the privacy of participants is preserved. This paper presents a novel mix-type remote voting system that permits to verify the correctness of a voting process without requiring complex and costly zero-knowledge proofs. It is based on a very efficient and lightweight hash-based construction that makes use of the homomorphic properties of ElGamal cryptosystem.

1. Introduction

Classical on paper voting with manual counting is a slow and error-prone way of performing elections. Electronic voting arises as a way to take advantage of technology in order to automate vote casting and counting. Electronic voting systems can be classified into two groups. The first group is composed of systems that substitute some components of traditional voting with some electronic process. The second one refers to systems in which voters are able to cast their votes remotely through telecommunication networks. Such systems are called *remote voting systems*.

A remote voting system has to guarantee some basic security properties:

- *Authentication:* Only voters listed in the electoral roll are able to cast a vote.
- *Unicity:* Each voter can vote only once.
- *Integrity:* Any attempt to dishonestly manipulate the election result is detected.
- *Privacy:* At the end of the voting procedure, no ballot can be linked to the identity of the voter who has cast it.
- *Verifiability:* Fairness of the whole voting procedure can be checked. A voting scheme is *universally verifiable* if anyone can independently verify that all ballots have been counted correctly. If voters can only verify that their own ballot has been counted, the scheme is *individually verifiable*.
- *Uncoercity:* A voter cannot prove that she voted in a particular way.
- *Fairness:* All ballots remain secret until the election is complete.

Current remote voting proposals in the literature achieve security by using cryptography. They can be classified into three main different paradigms:

- *Blind signature-based schemes:* First proposed in [11], in these systems each participant composes her vote and authenticates herself to a trusted party that checks the electoral roll and blindly signs her vote. After that, the signed vote is encrypted and sent to the Polling Station through an anonymous channel. Finally, each vote is decrypted and tallied. Privacy is preserved because the identity of the voter is kept confidential by the anonymous channel. This paradigm presents a security drawback permitting that a corrupted trusted party casts votes for abstaining voters. This fact could be detected by an auditor checking the signatures on all validation requests submitted but, after detection, there does not exist any procedure to identify invalid ballots (its signature is indistinguishable from that of valid ones) so that the election would have to be repeated.
- *Homomorphic tallying schemes:* First proposed in [7], these systems are only suitable for elections with a small number of candidates or choices. Each voter employs an additive homomorphic encryption algorithm to encrypt her vote. Once the Polling Station has collected all encrypted votes, they are homomorphically added so that the sum of votes for any candidate can be recovered with a single decryption. As no single vote is decrypted, privacy of voters is protected. The drawback of this paradigm is that cast

* Corresponding author. Tel.: +34 973 702713.
E-mail addresses: fsebe@matematica.udl.cat (F. Sebé), miret@matematica.udl.cat (J.M. Miret), jpujolas@matematica.udl.cat (J. Pujolàs), jordi.puiggali@scytl.com (J. Puiggalí).

0140-3664/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2009.11.013

votes must be proven to be valid using a zero-knowledge proof. As explained in [28], when the number of candidates or choices is large (e.g. preferential voting), the cost for such a proof becomes very high. So, this paradigm is accepted to be only suitable for elections with a small number of candidates or choices ("yes/no" voting).

- *Mix-type voting schemes:* First proposed in [4], they start with each voter authenticating herself and sending her vote in an encrypted message. Once all encrypted votes have been collected by the remote Polling Station, they are shuffled and remasked so that the link between each vote and the participant who cast it is lost. Finally, shuffled votes are decrypted and tallied. This paradigm permits to perform flexible elections (unlike the homomorphic tallying paradigm) with a very high security level and without the possibility for any party to completely disrupt the election (like in blind signature-based schemes). The next subsection provides more details about it.

In the recent years, some e-voting systems have been proposed for its use on highly dynamic networks. For instance, the proposals [21,18] are designed for mobile ad hoc and P2P networks, respectively.

1.1. Mix-type voting schemes

In mix-type voting schemes, there exists a trusted party storing the private key that permits vote decryption. This entity is assumed to be honest in the sense that it will only decrypt votes after they have been shuffled and remasked. If honesty of a single entity is difficult to achieve, the private key can be distributed between several parties in such a way that simultaneous corruption of all of them is very unlikely.

The Polling Station receives encrypted votes from voters. These encrypted votes are sent in an authenticated manner so that the authentication and unicity properties can be enforced. Once all encrypted votes have been collected, they are shuffled and remasked. The result is a mixed set of cryptograms whose cleartexts are the received votes. Due to the remasking operation, the initially received and the mixed ciphertexts cannot be related. Finally, the mixed cryptograms are decrypted and the tally process can be carried out.

The privacy of the system would be compromised if the party performing the mixing revealed the permutation applied to votes. For instance, if the Mixing Party omitted the mixing phase, the order of the decrypted votes would be the same as the original encrypted and signed votes. This would permit to link cleartext votes to the identity of voters using the identity information in certificates attached to the signatures. In this example, the privacy breach is due to the fact that the Mixing Party is known to have applied the identity (or null) permutation to votes. The same attack could be applied after performing an arbitrary permutation if this was revealed. In this sense, secrecy of the permutation is a mandatory aspect so as for privacy to be preserved. Privacy guarantees can be increased by having a Mixing Party composed of several Mixing Elements. In this way, the permutation applied to votes is the result of composing the individual permutation of each Mixing Element. Privacy is preserved as long as at least one of them maintains its permutation secret. In a real deployment, each political party may provide its own Mixing Element so that a collusion of all of them is very unlikely.

The voting system must also guarantee that votes will only be decrypted after verifying they have been permuted by all Mixing Elements. In some proposals, Mixing Elements store fragments of the secret key and perform a partial decryption of votes after mixing them. This ensures votes cannot be decrypted unless all Mixing Elements have taken part in the process.

The mixing of encrypted votes has to be done verifiably. This means that there must exist some method to prove that no manipulation, such as vote replacement, has taken place during this procedure. This verification method has to accomplish two main properties:

- A dishonest mixer will be caught with high probability even if a single plain-text message gets modified.
- It does not endanger unlinkability.

Mixing verification methods should be efficient in terms of *computational effort* and *communication cost* between the prover and the verifier. Last but not least, as mentioned in [6], *conceptual complexity* should also be reduced, otherwise users may not trust a system they cannot fully understand.

Current mixing verification processes in the literature make use of complex zero-knowledge proofs. The cost of such schemes becomes especially unaffordable when several Mixing Elements compose the Mixing Party. This is because these systems require the complex zero-knowledge proof of correct mixing to be performed by each mixer. This drawback is partially solved by *aggregate shuffle argument schemes* where the complexity at the verifier does not depend on the number of Mixing Elements.

1.2. Contribution and plan of this paper

In this paper, we present a novel aggregate shuffle argument scheme. Verifiers are only required to check a set of simple conditions that are very difficult to be satisfied simultaneously when some alteration has happened during the mixing procedure. The cost overhead for verifiers (voters) due to mixing verification is reduced to a linear quantity of lightweight modular products and hash computations (the number of costly modular exponentiations is reduced and constant). Regarding Mixing Elements, its extra cost is given by the addition of a constant and small number of dummy ciphertexts. The cost for the party storing the private key is similar to that at verifiers. We claim its conceptual complexity is lower than that of other proposals involving zero-knowledge proofs for correct mixing.

The paper is structured as follows. Section 2 reviews previous related work on remote voting systems. Section 3 is an introduction to the cryptographic tools used by the scheme. Section 4 describes the novel proposal while a detailed security analysis is presented in Section 5. Section 6 provides an assessment on parameter tuning and Sections 7 and 8 analyse the computational and communication costs of the proposal, respectively. Finally, Section 9 concludes the paper.

2. Related work

Blind signature-based remote voting schemes [11] provide a very simple way of carrying out an election. Such systems do not require the use of complicated zero-knowledge proofs so that its conceptual complexity is very low. Instead, they can be implemented using very simple cryptographic primitives such as symmetric/public key encryption and digital (blind) signatures. Voter privacy is provided as long as blindly signed votes are transmitted to the Polling Station through an anonymous channel like [32]. Several platforms built on this paradigm, such as [3,8,19], exist. Unfortunately, its theoretical impossibility of tracing illegal votes cast by a dishonest party storing the secret key used for blind signature computation has made the scientific community advise against it and recommend other paradigms offering tracing capabilities.

Homomorphic tallying remote voting schemes [7] permit to obtain the vote count very efficiently after the decryption of a

Please cite this article in press as: F. Sebé et al., Simple and efficient hash-based verifiable mixing for remote electronic voting, Comput. Commun. (2009), doi:10.1016/j.comcom.2009.11.013

cryptogram that contains the homomorphic addition of all single votes. Since no single votes are decrypted, vote privacy is provided. However, vote correctness has to be guaranteed using zero-knowledge proofs that become very expensive in complex elections with multiple choice or preferential voting [28]. In this sense, this paradigm is only recommended for very simple elections (for instance, "yes/no" voting). In [2] a platform using this paradigm is described.

Mix-type remote voting is the most versatile and secure paradigm. It permits to carry out complex elections with good security, privacy and tracing properties. Its main drawback is the need for a mechanism to guarantee that the result of the election has not been altered during the vote shuffling process. Proposals in the literature prove shuffling correctness by using very complex zero-knowledge proofs that introduce a very high computational cost. This cost is usually measured as the number of modular exponentiations to be performed as a function of the number of cast ballots, n. For instance, according to [6], $12n$ exponentiations are required in [26], $10n$ in [13,27], $8n$ in [16] and about $6n$ in [29]. Communication cost of such schemes is linear in n. For instance, $6388n$ bits for [13] and $2528n$ for [16]. In [23], the system [26] was implemented and shown to require more than an hour to prove the correct mixing of 4000 votes encrypted using ElGamal cryptosystem. This high and unaffordable delay prevents these systems from being used in large scale elections.

Different techniques have been proposed in order to reduce this delay. The proposal in [30] divides the ciphertexts in groups before mixing. This reduces the cost of proving shuffling correctness, but only a small fraction of permutations can be reached in this way, so that privacy is reduced. Another proposal detailed in [20] asks Mixing Elements to reveal part of the permutation they applied. This technique keeps a non-negligible probability of small alterations not being detected and requires most Mixing Elements to keep honest, otherwise, privacy may be compromised (other systems, like the one presented in this paper, maintain privacy as long as at least one is not corrupted). The authors in [6] propose *local forking* as a technique in which each verifier only checks the correctness of a small part of the mixing. Clearly, this technique does fully satisfy the *verifiability* property.

Aggregate shuffle argument schemes are the best option in elections where the Mixing Party is composed of a large amount of Mixing Elements (which is very good for privacy). These systems keep the cost at verifiers independent of the number of Mixing Elements. Such a scheme was first proposed in [1]. The more recent proposal [12] presents a lower cost than [1]. Being λ the number of Mixing Elements, the drawback of [12] is that although the cost for verifiers is $10n$ (independent of λ), this is not the case for the Mixing Elements that have to compute, respectively, $28n$ and $13n(\lambda-1)$ exponentiations for proving the correctness of its shuffling and verifying the correctness of shuffling of all other mixers. Regarding conceptual complexity, this is rather high in [12] due to the use of complicated zero-knowledge proofs. The system in [15] also accomplishes the aggregate shuffle argument property. It is a very fast hash-based proposal (similar to the one presented in this paper) that does not require zero-knowledge proofs. Unfortunately, it contains several security failures reported in [33].

Other proposals permitting the mixing of long messages [14] or secure against manipulation in the voting terminal or during vote transmission [22] exist in the literature.

3. Preliminaries

3.1. ElGamal cryptosystem

ElGamal [9] is a widely known public key cryptosystem whose security holds on the difficulty of solving the discrete logarithm problem. It is composed of four algorithms:

- *Set up:* A large cyclic group G of prime order q generated by g is chosen. Then, G, g and q are published.
- *Key generation:* Alice chooses $x \in \mathbb{Z}_q^*$ at random and computes $y = g^x$. Alice's secret key, x, is kept secret while her public key, y, is published.
- *Encryption:* So as to encrypt message m using Alice's public key, Bob converts message m into an element of G and chooses a random $r \in \mathbb{Z}_q^*$. Next, Bob computes $c = g^r$ and $d = m \cdot y^r$. The cryptogram is the tuple (c, d).
- *Decryption:* Alice decrypts (c, d) using her private key x by computing $m = \frac{d}{c^x}$ in G. Note that $\frac{d}{c^x} = \frac{m \cdot y^r}{g^{rx}} = \frac{m \cdot g^{rx}}{g^{rx}} = m$.

3.2. Verifiable decryption of an ElGamal ciphertext

In [5], a method is presented that permits a Prover to prove in zero knowledge that, given a tuple (g, u, y, v), she knows a secret value x satisfying $x = \log_g y = \log_u v$. Given a one-way hash function $\mathscr{H}$, the non-interactive version of this proof is as follows:

1. The Prover chooses $s \in \mathbb{Z}_q^*$ at random and computes the tuple $(a, b) = (g^s, u^s)$.
2. The Prover computes $e = \mathscr{H}(a \| b)$ (operator $\|$ denotes concatenation).
3. The Prover computes $r = s + ex$ and sends (a, b, r) to the Verifier.

We will denote $CP(g, u, y, v)$ (Chaum–Pedersen's proof) the tuple (a, b, r) sent by the Prover to the Verifier. This proof is verified as follows:

1. The Verifier computes $e = \mathscr{H}(a \| b)$.
2. The Verifier checks whether $g^r = a y^e$ and $u^r = b v^e$.

Given a publicly known cryptogram (c, d) encrypted under Alice's public key, she can verifiably decrypt it by publishing the cleartext m and $CP(g, c, y, \frac{d}{m})$. A verifier that succeeds in checking $CP(g, c, y, \frac{d}{m})$ is convinced that (c, d) is an encryption of m under public key y.

3.3. Homomorphic property of ElGamal

Given two ElGamal ciphertexts (under the same public key y) $(c_1, d_1) = (g^{r_1}, m_1 \cdot y^{r_1})$ and $(c_2, d_2) = (g^{r_2}, m_2 \cdot y^{r_2})$ encrypting cleartexts m_1 and m_2, respectively, it is easy to see that $(c, d) = (c_1 \cdot c_2, d_1 \cdot d_2) = (g^{r_1+r_2}, m_1 \cdot m_2 \cdot y^{r_1+r_2})$ is an encryption of $m_1 \cdot m_2$. In this sense, ElGamal has a multiplicative homomorphism property.

3.4. Remasking an ElGamal ciphertext

Given an ElGamal encryption of m under public key y, $(c, d) = (g^r, m \cdot y^r)$, this cryptogram can be transformed into a different cryptogram (c', d') by choosing $r' \in \mathbb{Z}_q^*$ at random and computing $(c', d') = (c \cdot g^{r'}, d \cdot y^{r'})$. The cleartext of (c', d') is m. Under the Decisional Diffie–Hellman assumption, a third party cannot determine whether the cleartext of (c, d) and (c', d') are the same, i.e. ElGamal cryptosystem is semantically secure.

3.5. Cryptography over elliptic curves

A public key cryptosystem can be constructed over a prime order subgroup of the group of points of an elliptic curve [17]. An elliptic curve E over a finite field $\mathbb{Z}_p$ is defined by an equation of the form

$$y^2 = x^3 + ax + b$$

Please cite this article in press as: F. Sebé et al., Simple and efficient hash-based verifiable mixing for remote electronic voting, Comput. Commun. (2009), doi:10.1016/j.comcom.2009.11.013

where $a, b \in \mathbb{Z}_p$, with $4a^3 + 27b^2 \neq 0 \pmod{p}$. The set of points $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ satisfying this equation together with a *point at infinity* is the set of points of the curve, denoted $E(\mathbb{Z}_p)$. An addition operation given by the chord-tangent method endows the set of points of the curve E with a group structure where the discrete logarithm problem is believed to be hard.

Its great advantage is that the Index-Calculus algorithm [31] for solving the discrete logarithm cannot be applied here. As a consequence, the same security of a public key defined over $\mathbb{Z}_p^*$, with p being a 3072 bits long prime, can be achieved over an elliptic curve defined over a 256 bits long finite field.

Given an elliptic curve E and a subgroup of $E(\mathbb{Z}_p)$ of primer order q generated by P, we encrypt message m under public key $Q_E = x_E P$ (x_E is the private key) using the following variant of ECIES [17] (elliptic curve integrated encryption scheme) that makes use of the RC4 stream cipher:

1. Take $r \in \mathbb{Z}_q^*$ at random.
2. Compute point $R = rQ_E = (x_R, y_R)$.
3. Compute point $Z = rP = (x_Z, y_Z)$.
4. Let $K = x_R \pmod{2^{128}}$.
5. Compute $C = \mathrm{RC4}_K(m)$.
6. The resulting ciphertext is $E_{Q_E}(m) = (x_Z, b_{y_Z}, C)$.

Value b_{y_Z} is a bit indicating whether $y_Z \geqslant -y_Z$. Note that for a curve defined over $\mathbb{Z}_p$, being p a 256 bits long prime, $E_{Q_E}(m)$ is $(257 + l_m)$ bits long (value l_m denotes the bitlength of m).

Decryption of $E_{Q_E}(m) = (x_Z, b_{y_Z}, C)$ is done as follows:

1. From x_Z and b_{y_Z}, obtain $Z = (x_Z, y_Z)$.
2. Using private key x_E, compute $R = x_E Z = (x_R, y_R)$.
3. Compute $K = x_R \pmod{2^{128}}$.
4. Compute $m = \mathrm{RC4}_K^{-1}(C)$.

Verifiable decryption is done by publishing point R and the zero-knowledge proof $CP(P, Z, Q_E, R)$ (see Section 3.2) constructed over the subgroup generated by P. This proof permits to verify the correctness of R. After that, the verifier can compute K and decrypt C by itself.

4. Our proposal

Our voting system is composed of the following parties:

- *Polling Station*. This is the central element that coordinates the system. At the beginning, it collects the votes of participants and, at the end, it publishes the result of the election. When necessary, the Polling Station requires the participation of the Key Storage and Mixing parties. It implements a publicly accessible bulletin board where information is published in an authenticated manner (for every piece of published data, all parties can verify its integrity and the identity of its publisher). This can be achieved by using digital signatures.
- *Key Storage Trusted Party*. This trusted party is proprietary of two private/public key pairs, x/y, x_E/Q_E, that have been generated for this election. Keys generated for one election should never be used in other elections. Public key y is defined over a multiplicative prime order subgroup of $\mathbb{Z}_p^*$, while Q_E is defined over a prime order subgroup of the group of points of an elliptic curve E. Both public keys are known and accepted by all the parties (they may be certified). This party securely stores the corresponding private keys, x and x_E.
- *Mixing Party*. This is the party that performs the mixing of votes. It is composed of one or several Mixing Elements. We will assume that at least one of these elements is honest. The others may be corrupted and act dishonestly by trying to replace some cast votes by fraudulent ones or by revealing how they performed the mixing.
- *Participants*. These are the entities that cast a vote.

We will assume all parties have a certified public/private key pair that permits them to send authenticated data. This key pair will be used to sign the data they publish on the bulletin board.

Our voting system is composed of the following stages:

1. *Set up*. At this preliminary phase, the Key Storage Trusted Party publishes the parameters of the groups where its public keys, y and Q_E, are defined. It also publishes the description of a hash function $\mathscr{H}$ generating $l_{\mathscr{H}}$-bits digests. The Polling Station initializes the bulletin board.
2. *Voting*. Each participant P_i generates a ciphertext encrypting her vote v_i. This encrypted vote is signed and sent to the Polling Station. The Polling Station (and the other participants) will check the electoral roll for authentication and vote unicity. If this checking is successful, the encrypted vote will be published (authenticated by its signature by P_i) on the publicly accessible bulletin board.
3. *Mixing*. The Mixing Party takes the encrypted votes from the bulletin board and shuffles and remasks them. The resulting ciphertexts are sent to the Polling Station where they will be published. During this phase, some additional data required for verification is published.
4. *Opening*. At the end, the Key Storage Trusted Party verifies the mixing (supervised by all participants) and then decrypts the mixed encrypted votes. The output of this final stage is the anonymous list of cast ballots.

These stages are next presented in more detail.

4.1. Setup

Key Storage Trusted Party publishes the parameters where its public key y is defined. These are two large primes p and q satisfying $p = 2q + 1$. Key y is defined over the subgroup of $\mathbb{Z}_p^*$ generated by an element g having order q. It is easy to see that this subgroup corresponds to the set of quadratic residues of $\mathbb{Z}_p^*$. This property is useful because, given an element $m \in \mathbb{Z}_p^*$, its belonging to the subgroup generated by g can easily be verified by checking if its Legendre symbol is 1, i.e. $\left(\frac{m}{p}\right) = 1$. We assume this property is checked by parties prior to performing any operation over the components of an ElGamal ciphertext.

Function $\mathscr{H}$ is taken as the $l_{\mathscr{H}}$ least significant bits of some cryptographic hash function such as SHA256 [10].

The Key Storage Trusted Party also publishes the parameters of the elliptic curve cryptosystem where its public key Q_E is defined.

Finally, its public keys y and Q_E are published.

4.2. Voting

Next, we detail how participant P_i casts her vote v_i. This is done as follows:

1. Given vote v_i, P_i first encrypts it under public key Q_E, $V_i = E_{Q_E}(v_i)$, computes $h_i = \mathscr{H}(V_i)$ and then encodes it as a vote-message

   $$m_i = V_i \| h_i \| b_i.$$

   Value b_i is chosen so that m_i is a quadratic residue of $\mathbb{Z}_p^*$.
2. Next, P_i encrypts m_i under public key y, obtaining the ciphertext $C_i = (c_i, d_i)$.

Please cite this article in press as: F. Sebé et al., Simple and efficient hash-based verifiable mixing for remote electronic voting, Comput. Commun. (2009), doi:10.1016/j.comcom.2009.11.013

3. Finally, P_i digitally signs C_i and sends C_i and its digital signature $sig_{P_i}(C_i)$ to the Polling Station where it will be published on the publicly accessible bulletin board.
4. All participants must check the validity of this digital signature. They must also check that P_i has not cast any other encrypted vote and her presence in the electoral roll.

4.3. Vote mixing

Once all n votes have been collected, the bulletin board contains a set composed of the n signed votes $\left\{C_i, sig_{P_i}(C_i)\right\}_{0\leqslant i<n}$. Each encrypted vote corresponds to an ElGamal tuple $C_i = (c_i, d_i)$. Then,

1. The Key Storage Trusted Party computes and publishes the cryptogram

   $$C_{check} = \prod_{i=0}^{n-1} C_i = \left(\prod_{i=0}^{n-1} c_i, \prod_{i=0}^{n-1} d_i\right).$$

   Note that, any one having access to the set of votes can verify that this computation has been performed correctly. All participants must verify it. C_{check} is a cryptogram that can be decrypted using the secret key x returning (see Section 3.3),

   $$m_{check} = \prod_{i=0}^{n-1} m_i.$$

2. The first Mixing Element, ME_1, generates s ElGamal ciphertexts encrypting (under public key y) different randomly generated dummy messages $\{\hat{m}_{1,1}, \ldots, \hat{m}_{1,s}\}$. Let us denote them C_{n+k} for $0 \leqslant k < s$.
3. ME_1 publishes the following list on the bulletin board:

   $$\widehat{H}_1 = \{\mathrm{SHA256}(\hat{m}_{1,1}), \ldots, \mathrm{SHA256}(\hat{m}_{1,s})\}.$$

4. Next, ME_1 generates a secret $n+s$ elements permutation π_1 and computes a new list of shuffled and remasked ciphertexts (including dummies) as

   $$C'_i = (c'_i, d'_i) = \left(c_{\pi_1(i)} \cdot g^{r'_i}, d_{\pi_1(i)} \cdot y^{r'_i}\right), \quad 0 \leqslant i < n+s.$$

   Each value $r'_i \in \mathbb{Z}_q^*$ is randomly chosen. If there are no more Mixing Elements, ME_1 publishes the set $\{C'_i\}_{0\leqslant i<n+s}$ on the bulletin board and the mixing procedure finishes here.
5. If the Mixing Party is composed of λ, $\lambda > 1$, Mixing Elements, ME_1 will send its resulting set of ciphertexts to ME_2 who will take it as input and will perform the same operation (dummy addition, shuffling and remasking). This process will be carried out sequentially by all Mixing Elements. The last one, ME_λ will publish its output on the bulletin board. At the end, the bulletin board will contain λ lists $\widehat{H}_1, \ldots, \widehat{H}_\lambda$ and a set of $n+s\lambda$ mixed ciphertexts, $\{C'_i\}_{0\leqslant i<n+s\lambda}$ (containing n encrypted votes and $s\lambda$ dummy ciphertexts).

4.4. Vote opening

The Key Storage Trusted Party does:

1. Decrypt each ciphertext $C'_i = (c'_i, d'_i)$ using private key x and publish the set $\{\hat{m}'_i\}_{0\leqslant i<n+s\lambda}$ on the bulletin board (this set contains $s\lambda$ dummies).
2. Remove from the set of decrypted messages, those $\hat{m}'_i$ satisfying that $\mathrm{SHA256}(\hat{m}'_i) \in \widehat{H}_1$. This operation must remove exactly s different elements. If this condition is not satisfied, the procedure stops. The same operation is carried out for the remaining lists $\widehat{H}_2, \ldots, \widehat{H}_\lambda$. Let us denote $\{m'_i\}_{0\leqslant i<n}$ the set of the n remaining messages once dummies have been removed. These removings are supervised by all participants.
3. Decrypt $C_{check} = (c_{check}, d_{check})$ using private key x, and publish m_{check} and its proof of correct decryption

   $$CP\left(g, c_{check}, y, \frac{d_{check}}{m_{check}}\right).$$

After this is done, all parties check:

1. The correctness of the verifiable decryption of m_{check}.
2. The product of decrypted messages $m'_{check} = \prod_{i=0}^{n-1} m'_i$ equals m_{check}.
3. All messages $m'_i = V'_i \| h'_i \| b'_i$ satisfy that $h'_i = \mathscr{H}(V'_i)$.
4. All messages m'_i are different.

Note that if the shuffling and remasking operations have been done correctly, then it follows that $m_{check} = m'_{check}$. The reverse implication is not necessarily true.

If some of the previous checkings failed, the privacy of the election would not be compromised because the vote opening procedure would be interrupted at a point in which elliptic curve ciphertexts $\{V'_i\}_{0\leqslant i<n}$ have not been decrypted yet.

If all these checkings are satisfied, then,

1. The Key Storage Trusted Party verifiably decrypts each $V'_i = E_{Q_E}(v'_i)$ publishing its plaintext v'_i and the corresponding proof of correct decryption.
2. All participants verify the correctness of these decryption proofs.

If all these checkings are satisfied, the set $\{v'_i\}_{0\leqslant i<n}$ is taken as the result of the election.

5. Security

5.1. Authentication and Unicity

These security properties state that only voters in the electoral roll can vote. Furthermore each voter can only cast a single vote.

In our system, the encrypted votes collected by the Polling Station are signed (we assume participants' public keys are properly certified by some trusted Certification Authority) by the participant who casts it. The encrypted vote and its signature are made publicly available. In this way, all participants can check that each collected vote is signed by a different participant. Moreover, if the electoral roll is public, participants can also check each voter appears in the list (if this is not the case, they should rely this checking is done by the Polling Station). If identity of voters should be kept secret, pseudonyms could be used for anonymous authentication.

5.2. Privacy

The privacy property requires that at the end of the voting procedure, no ballot can be linked to the identity of the voter who has cast it. The following lemmas address it.

Lemma 1. *Assuming the Key Storage Trusted Party only decrypts C_{check} and cryptograms $\{C'_i\}_{0\leqslant i<n+s\lambda}$ and the Mixing Party does not reveal the permutation it applied, the probability of correctly linking a vote v'_i with the identity of the participant who cast it is at most $1/n$.*

Proof. In our system, each ciphertext C_i can be linked to the participant who cast it (it is signed). After vote opening, it is also possible to link each C'_i to its vote v'_i (we consider dummies have already been removed).

The ElGamal cryptosystem is semantically secure under the Decisional Diffie–Hellman assumption. This means that it is not possible to determine whether two different cryptograms will result

Please cite this article in press as: F. Sebé et al., Simple and efficient hash-based verifiable mixing for remote electronic voting, Comput. Commun. (2009), doi:10.1016/j.comcom.2009.11.013

or not in the same plaintext once decrypted. From semantic security of ElGamal, assuming the permutation relating $\{C_i\}_{0\leqslant i<n}$ and $\{C'_i\}_{0\leqslant i<n}$ (after dummies removal) is not revealed, a ciphertext C'_i can only be linked to its corresponding C_i by random guessing, whose probability of success is $1/n$.

So, given a participant, the probability of correctly linking her with a decrypted mixed vote is $1/n$. □

Note that cryptography cannot avoid information disclosure in situations where, for instance, all votes have the same value. In these cases, privacy can only be provided by participant anonymity.

Next we will address the security of the mixing process. The permutation relating the initial cast votes $\{C_i\}_{0\leqslant i<n}$ and the list of decrypted votes $\{v'_i\}_{0\leqslant i<n}$ depends on the permutation applied by each Mixing Element. As long as at least one of the Mixing Elements keeps its permutation secret, the overall permutation will not be known. In this case, as stated by Lemma 1, the only way to relate identities and decrypted votes is random guessing. So, a procedure permitting to verify that all Mixing Elements have contributed to this permutation will ensure its secrecy. This is equivalent to proving that no Mixing Element has been bypassed during the mixing process.

We will consider the worst case in which only one Mixing Element is honest. Next lemma addresses how difficult it is for a coalition of malicious mixers to bypass this honest Mixing Element.

Lemma 2. *In a voting with n participants, the probability of not detecting that a honest Mixing Element has been bypassed is upper bounded by* $(\frac{s}{n})^s$.

Proof. During the mixing procedure, each Mixing Element sequentially participates by adding s dummy ciphertexts to the set of cryptograms and then applying a secret permutation to the whole set. At vote opening, the dummy messages introduced by each mixer are identified and removed. If some of them are missing, the vote opening phase is interrupted. So, a cheating coalition that tries to bypass a honest Mixing Element will only succeed if they manage to locate the dummy ciphertexts of the Mixing Element they wish to bypass and introduce them in the set of votes they are mixing dishonestly.

Since the content and location of the dummies at the output of the honest Mixing Element is not known by the coalition, they must randomly guess which are the dummy ciphertexts. These ciphertexts will be taken and inserted in the "dishonest" set of votes they are producing.

Assuming the honest mixer performs in the jth position, the attackers will succeed if they randomly guess which are the s dummies out of a list of $n+js$ ciphertexts. This will happen with probability

$$\frac{s!\ (n+(j-1)s)!}{(n+js)!} \leqslant \frac{s!\ n!}{(n+s)!} < \left(\frac{s}{n}\right)^s \qquad \square$$

5.3. Integrity

This property is related to verifiability. In fact, we will next prove that, when all checks carried out in our voting system are satisfied, the output of the vote opening stage is integral. This means that the output list of votes $\{v'_i\}_{0\leqslant i<n}$ is a permutation of those cast by participants, $\{v_i\}_{0\leqslant i<n}$. As we will see our system is *universally verifiable*, i.e. any party (including those that have not taken part in it) can check the correctness of the whole procedure. This integrity property depends on parameter $l_{\mathscr{H}}$ and it is achieved when $2^{l_{\mathscr{H}}}$ is an overwhelming large value. The proof of integrity will lean on some lemmas that are next presented.

Lemma 1 shows that, in our voting protocol, two sets of vote-messages satisfying $m_{check} = m'_{check}$ (property checked by participants during vote opening at step 2), cannot differ in a single vote-message. This means that they can only differ in zero, two or more elements.

Without loss of generality, the lemma is formulated assuming the sets have been sorted so that the first $(n-1)$ elements are pairwise equal.

Lemma 3. *Let us assume two sets of vote-messages* $\mathscr{L} = \{m_0, \ldots, m_{n-1}\}$ and $\mathscr{L}' = \{m'_0, \ldots, m'_{n-1}\}$ *satisfying:*

1. $\forall i,\ 0 \leqslant i < n-1,\ m_i = m'_i$, and
2. $m_{check} = m'_{check}\ \left(\prod_{i=0}^{n-1} m_i = \prod_{i=0}^{n-1} m'_i\right)$

Then, $m_{n-1} = m'_{n-1}$.

Proof. From condition 2 we have that

$$m_{n-1} \cdot \prod_{i=0}^{n-2} m_i = m'_{n-1} \cdot \prod_{i=0}^{n-2} m'_i.$$

From condition 1 we have that $\prod_{i=0}^{n-2} m_i = \prod_{i=0}^{n-2} m'_i$. Then, we conclude $m_{n-1} = m'_{n-1}$. □

Next, we will prove that an element of G generated by some procedure that does not query $\mathscr{H}$ is a vote-message with probability $2^{-l_{\mathscr{H}}}$.

Lemma 4. *Given* $m \in G$, *where m is parsed as* $x = V\|h\|b$, *if m was generated without querying* $\mathscr{H}$ *providing V as input, the probability that m is a vote-message is* $2^{-l_{\mathscr{H}}}$.

Proof. Given $m \in G$, so as for $m = V\|h\|b$ to be a vote-message, it must satisfy $h = \mathscr{H}(V)$. Given V, if h was computed without querying $\mathscr{H}$ providing V as input, then h will match $\mathscr{H}(V)$ with probability $2^{-l_{\mathscr{H}}}$. □

Next we will study the complexity of obtaining two sets of vote-messages differing in two or more elements that satisfy all checkings of the protocol. The proof will be done under the assumption that the attacker knows the (plaintext) vote-messages from the initial list she is trying to replace by fraudulent ones. When the vote-messages to be replaced are encrypted and unknown to the attacker, the complexity can only increase since this fact reduces the information the attacker has.

The following lemma proves the temporal cost of computing two sets of vote-messages differing in two or more elements.

Without loss of generality, the lemma is formulated assuming the two sets have been sorted so that the elements to be replaced are located at the first k, $k > 1$, positions.

Lemma 5. *Given a set containing n vote-messages* $\mathscr{L} = \{m_0, \ldots, m_{n-1}\}$, *computing another set of vote-messages* $\mathscr{L}' = \{m'_0, \ldots, m'_{n-1}\}$ *satisfying:*

1. $m_i = m'_i,\ \forall k \leqslant i < n$, and
2. $m_{check} = m'_{check}\ \left(\prod_{i=0}^{n-1} m_i = \prod_{i=0}^{n-1} m'_i\right)$ and
3. $m_i \neq m'_j$ for all $0 \leqslant i,j < k$

requires at least an expected $O(2^{l_{\mathscr{H}}})$ *temporal cost.*

Proof. Let us denote $A = m_0 \cdot \ldots \cdot m_{k-1}$. The objective is to find a list of vote-messages $m'_0, \ldots, m'_{k-1}$ satisfying $m'_0 \cdot \ldots \cdot m'_{k-1} = A$. Trivial solutions where $m'_0 = m_{\pi(0)}, \ldots, m'_{k-1} = m_{\pi(k-1)}$ for some permutation π are discarded by condition 3.

From the above condition, we have that $m'_0 = A/(m'_1 \cdot \ldots \cdot m'_{k-1})$. Then, from several seeds $V'_1, \ldots, V'_{k-1}$ we compute (calling $\mathscr{H}$) the

vote-messages $m'_1,\ldots,m'_{k-1}$ that encode $V'_1,\ldots,V'_{k-1}$. By denoting $A' = m'_1 \cdot \ldots \cdot m'_{k-1}$, we compute m'_0 as $m'_0 = A/A'$.

Since during the computation of $m'_0 = V'_0 \| h'_0 \| b'_0$ no call to $\mathscr{H}$ with input V'_0 is done, from Lemma 4 we know that the probability that m'_0 is a vote-message is $2^{-l_{\mathscr{H}}}$.

Note that any other procedure not starting from $V'_1,\ldots,V'_{k-1}$ and next calling $\mathscr{H}$ to generate $m'_1,\ldots,m'_{k-1}$ will succeed with probability at most $2^{-l_{\mathscr{H}}}$ in obtaining all $m'_1,\ldots,m'_{k-1}$ being vote-messages.

So, the only strategy that can be followed to generate $m'_0,\ldots,m'_{k-1}$ consists of successively trying different initial seeds $V'_1,\ldots,V'_{k-1}$ until m'_0 satisfies the condition needed to be a vote-message. Since the probability of success of each trial is $2^{-l_{\mathscr{H}}}$, the expected amount of trials before succeeding is $2^{l_{\mathscr{H}}}$. This gives the expected $O(2^{l_{\mathscr{H}}})$ temporal cost. □

Next, we present the concluding theorem that states the computational security of the integrity of the voting system.

Theorem 1. *If all checks performed in the voting system are satisfied, a corrupted Mixing Party whose output is a set of encrypted votes that once decrypted are not a permutation of the initially cast votes has spent at least an expected $O(2^{l_{\mathscr{H}}})$ time in it.*

Proof. Lemma 3 proves that a corrupted Mixing Party cannot modify the value of a single vote. Next, in Lemma 5 it is proven that modifying two or more votes requires exponential time in $l_{\mathscr{H}}$, $O(2^{l_{\mathscr{H}}})$. □

5.4. Uncoercity and fairness

Our system satisfies uncoercity in the sense that a voter does not keep any receipt proving she voted in a particular way. Fairness is provided as long as the Key Storage Trusted Party does not decrypt any vote until the end of the election.

6. Parameter choice

Prior to deployment, some parameters of the system have to be defined. We will tune them so as to provide a 128 bits computational security level. According to [24], this is the long term (beyond 2030) recommended security level for unclassified applications.

So as for integrity to be guaranteed, the bitlength of h, $l_{\mathscr{H}}$, must be chosen large enough so that $2^{l_{\mathscr{H}}}$ is an overwhelming large value. Taking $l_{\mathscr{H}} = 128$ provides our desired security level.

Regarding b, its bitlength l_b has to be chosen so that once V and h are given, there exists at least one value for b so that $m = V\|h\|b \in \mathbb{Z}_p^*$ is a quadratic residue. Since half of the elements of $\mathbb{Z}_p^*$ are quadratic residues, given l_b the probability that none of its values results in m being a quadratic residue is $2^{-2^{l_b}}$. Taking an appropriate length for b makes this probability negligible. For instance, if b is 8 bits long, this probability is as low as 2^{-256}.

Regarding the elliptic curve public key Q_E, it shall be defined over a prime field whose cardinal is 256 bits long [24]. In this case, the length of ciphertext $V = E_{Q_E}(v)$ is $257 + l_v$ bits. Prime number p must be 3072 bits long [24].

Under this parameter configuration, the bits of m left for representing the vote v are 2678 ($l_v = |p| - 1 - |V| - l_{\mathscr{H}} - l_b$). If this is not enough to encase v, the length of p, $|p|$, has to be taken larger.

Regarding parameter s that tunes the probability of success of a Mixing Element bypassing, its value depends on n. For instance, for $n = 1024$, taking $s = 4$ provides a probability of success below $\left(\frac{1}{256}\right)^4 = 2^{-32}$. Note that in this case we are not providing computational security. Since a coalition can only perform a single trial, this probability is low enough so as to ensure that any attempt to bypass a Mixing Element will surely be detected.

The number of Mixing Elements, λ, has to be taken so that it is difficult that all of them are corrupted at the same time. For instance, in a national election, each political party could include its own Mixing Element.

7. Computational cost analysis

In a remote voting system, verifiability of the vote shuffling and remasking procedure is not enough. Integrity and authenticity of input ciphertexts must also be guaranteed, otherwise, cast ballots could be replaced by fraudulent ones before the mixing begins. At the vote opening phase, decryption of mixed ciphertexts must be done verifiably so as to avoid the result of the election being falsified at this point.

We claim that authenticating (using digital signature or any other method) the input ciphertexts and verifiably decrypting the mixed ciphertexts is mandatory in any remote voting system. So, we will not consider its cost as part of the proof of correct mixing.

In this section, we will analyse the computational cost of our system in an election where n participants take part. We will consider the parameters tuned for a 128 bits security level as detailed in Section 6.

According to [25], the cost of elliptic curve cryptography over a 256 bits finite field is 10 times lower than the cost of ElGamal cryptosystem over $\mathbb{Z}_p^*$, being p a 3072 bits long prime. Our analysis will consider this ratio.

7.1. Vote casting

In any voting system, each participant casts her vote and verifies the vote cast by all other participants. So, the cost at this stage is dominated by the $(n-1)$ signature verifications performed by each participant. The cost of generating her own (and unique) vote can be completely neglected.

7.2. Vote opening

For verifiable decryption of n ciphertexts, systems that separate the proof of mixing and the proof of decryption (like [26]) should perform:[1]

1. Decryption of n ElGamal ciphertexts over $\mathbb{Z}_p^*$.
2. Prove the correct decryption of those n ciphertexts.

This requires the Key Storage Trusted Party to perform 1 exponentiation for each decryption plus 2 exponentiations for generating each proof. Thereby, the overall amount of modular exponentiations is $3n$. The participants verifying these decryptions will perform $4n$ modular exponentiations.

Our system follows a slightly different paradigm requiring:

1. Decryption of $n + s\lambda$ ciphertexts over $\mathbb{Z}_p^*$.
2. Decryption of n elliptic curve ciphertexts.
3. Proof of correct decryption of n elliptic curve ciphertexts.

The extra $s\lambda$ decryptions due to dummy messages are introduced for proving the correctness of the mixing process and will be considered in the next section. So, our system requires the Key Storage Trusted Party to perform n modular exponentiations at step 1 plus n and $2n$ elliptic curve exponentiations at steps 2 and 3, respectively. Each verifier will compute $4n$ elliptic curve exponentiations while verifying the proofs at step 3.

[1] Computing a full length modular exponentiation in $\mathbb{Z}_p^*$, being p an n bits long prime requires an average of $1.5n$ products. In our case, where $n = 3072$, we will consider the cost of a product negligible with respect to the cost of an exponentiation.

Please cite this article in press as: F. Sebé et al., Simple and efficient hash-based verifiable mixing for remote electronic voting, Comput. Commun. (2009), doi:10.1016/j.comcom.2009.11.013

Due to the lower cost of elliptic curve operations, our system approximately permits a 50% computational cost reduction for the Key Storage Trusted Party and a 90% reduction for provers at this stage with respect to systems like [26].

7.3. Verification of correct mixing

The operations required for verifying a mixing are next summarized (the Mixing Party does not take part in them):

1. Computation of $2(n-1)$ modular products for computing C_{check}.
2. Verifiable decryption of C_{check} (only done by the Key Storage Trusted Party).
3. Checking the proof of correct decryption of C_{check} (done by the Participants).
4. Decryption of $s\lambda$ dummy ElGamal ciphertexts (only done by the Key Storage Trusted Party).
5. Computation of $2n + s\lambda$ hash functions ($n + s\lambda$ during the identification and removal of dummy messages and n while checking that each m'_i satisfies the vote-message property).
6. Computation of $(n-1)$ modular products during the computation of m'_{check}.

Taking into account all operations but modular exponentiations are negligible, our system requires the Key Storage Trusted Party to compute $4 + s\lambda$ exponentiations (at steps 2 and 4). Considering the values chosen at Section 6 ($n = 1024$, $s = 4$) and $\lambda = 5$ (number of Mixing Elements), the total amount of exponentiations is 24. The cost for verifiers is reduced to four exponentiations at step 3. This clearly outperforms other mix-type proposals reviewed in Section 2 whose cost grows linearly with n.

The extra cost for Mixing Elements is only caused by the addition of dummy ciphertexts. Since parameter s takes small values, this causes a negligible overhead when n is large.

8. Communication cost analysis

In this section, we analyse the amount of bits transmitted between each Participant and the bulletin board located at the Polling Station. We will not consider some extra bits required for authentication of data published on the bulletin board or other secondary transmissions such as public key certificates, etc.

8.1. Vote casting

Each Participant casts her encrypted vote together with its digital signature on it and verifies the signature of the ciphertexts sent by the other $(n-1)$ Participants. So, each Participant participates in the communication of n signed ciphertexts. Considering the parameters chosen at Section 6, each ElGamal ciphertext is 2×3072 bits long. If signatures were computed over an elliptic curve cryptosystem (for instance, ECDSA [17]), each signature would be 2×256 bits long. Summarizing, each participant is involved in the transmission of $6656n$ bits during this stage.

As justified in the previous section, verifiability requires submitted data to be authenticated. So, the cost detailed here is mandatory for any remote voting system where cast votes are encrypted using ElGamal.

8.2. Vote opening

At this stage, each Participant must receive:

1. The set of cleartexts $\{\hat{m}'_i\}_{0 \leqslant i < n+s\lambda}$ (ciphertexts $\{C'_i\}_{0 \leqslant i < n+s\lambda}$ are not required).
2. C'_{check}, m'_{check} and its proof of decryption.
3. Verifiable decryption of $\{V'_i\}_{0 \leqslant i < n}$.

Like in the previous section, the extra $s\lambda$ cleartexts due to dummy messages will be considered as part of the proof of mixing. Then, step 1 requires transmission of $3072n$ bits. Step 2 requires 6×3072 bits (2×3072 bits for C'_{check}, 3072 bits for m'_{check} plus 3×3072 bits for the proof of correct decryption). For each verifiable decryption, step 3 requires transmission of 257 bits for the (compressed) elliptic curve point R (see Section 3.5) plus $2 \times 257 + 256$ for the decryption proof. The overall transmission at step 3 is $(3 \times 257 + 256)n$ bits.

Summarizing, the total amount of transferred bits at the vote opening stage is $4099n + 18{,}432$.

For systems requiring verifiable decryption of n ElGamal mixed ciphertexts (those that separate the proof of mixing and the proof of decryption), the communication cost would be $(6 \times 3072) = 18{,}432n$ bits (in this case, ciphertexts $\{C'_i\}_{0 \leqslant i < n}$ have to be transferred).

Comparing both methods, for large n, our system permits a reduction in 78% of transferred bits at this phase. This saving comes from the fact that in our system, the large ElGamal ciphertexts do not need to be transferred and the proof of correct decryption is done over the (smaller) elliptic curve cryptosystem instead of the (larger) ElGamal one.

8.3. Verification of correct mixing

Extra data transferred due to mixing verification is next summarized:

1. The $s\lambda$ dummy cleartexts contained in set $\{\hat{m}'_i\}_{0 \leqslant i < n+s\lambda}$ (not dummy ones have been computed in previous subsection).
2. The sets of hash values $\widehat{H}_1, \ldots, \widehat{H}_\lambda$.

Step 1 requires transmission $3072 \times s\lambda$ bits while step 2 requires $256 \times s\lambda$.

Considering the values chosen at Section 6 ($n = 1024$, $s = 4$) and $\lambda = 5$ (number of Mixing Elements), the total amount of transferred bits for mixing verification is 66,560. This clearly outperforms other mix-type proposals reviewed in Section 2 whose cost grows linearly with n.

9. Conclusion

In this paper, a novel mix-type remote voting scheme has been presented. Its great contribution is given by its simplicity and low computational and communication cost required to verify the correctness of vote mixing. All statements about the security and efficiency of the system have been properly proven.

Acknowledgements

We acknowledge partial support by Spanish Ministry of Science and Education under projects MTM2007-66842-C02-02, TSI2007-65406-C03-01 and CSD2007-0004.

References

[1] M. Abe, Universally verifiable mix-net with verification work independent of the number of mix-servers, Lecture Notes in Computer Science 1403 (1998) 437–447.
[2] P. Akritidis, Y. Chatzikian, M. Dramitinos, E. Michalopoulos, D. Tsigos, N. Ventouras, The VoteSecure secure Internet voting system, Lecture Notes in Computer Science 3477 (2005) 420–423.
[3] H. Al-Shalabi, E-voting scheme over internet, Business and Information (2008).
[4] D. Chaum, Untraceable electronic mail, return addresses and digital pseudonyms, Communications of the ACM 24 (2) (1981) 84–88.

Please cite this article in press as: F. Sebé et al., Simple and efficient hash-based verifiable mixing for remote electronic voting, Comput. Commun. (2009), doi:10.1016/j.comcom.2009.11.013

[5] D. Chaum, T. Pedersen, Wallet databases with observers, Lecture Notes in Computer Science 740 (1992) 89–105.
[6] J. Cichoń, M. Klonowski, M. Kutyłowski, Distributed verification of mixing – local forking proofs model, Lecture Notes in Computer Science 5107 (2008) 128–140.
[7] J.D. Cohen(Benaloh), M.J. Fischer, A robust and verifiable cryptographically secure election scheme, in: FOCS'85, 1985, pp. 372–382.
[8] L.F. Cranor, R.K. Cytron, Design and Implementation of a Security-Conscious Electronic Polling System, Washington University Computer Science Technical Report WUCS-96-02, 1996.
[9] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithm, IEEE Transactions on Information Theory 31 (1985) 469–472.
[10] Federal Information Processing Standards. FIPS 180-2: Secure Hash Standard, Ammended, 2004.
[11] A. Fujioka, T. Okamoto, K. Ohta, A practical secret voting scheme for large scale elections, Lecture Notes in Computer Science 718 (1993) 244–251.
[12] J. Furukawa, H. Imai, An efficient aggregate shuffle argument scheme, Lecture Notes in Computer Science 4886 (2007) 260–274.
[13] J. Furukawa, K. Sako, An efficient scheme for proving a shuffle, Lecture Notes in Computer Science 2139 (2001) 368–387.
[14] J. Furukawa, K. Sako, An efficient publicly verifiable mix-net for long inputs, Lecture Notes in Computer Science 4107 (2006) 111–125.
[15] P. Golle, S. Zhong, D. Boneh, M. Jakobsson, A. Juels, Optimistic mixing for exit-polls, Lecture Notes in Computer Science 2501 (2002) 451–465.
[16] J. Groth, A verifiable shuffle of homomorphic encryptions, Lecture Notes in Computer Science 2567 (2002) 145–160.
[17] D. Hankerson, A. Menezes, S. Vanstone, Guide to Elliptic Curve Cryptography, Springer, New York, 2004.
[18] Q. Huang, D. Jano, H. Wang, Applications of secure electronic voting to automated privacy-preserving troubleshooting, in: Proceedings of the 12th ACM Conference on Computer and Communications, 2005, pp. 68–80.
[19] S. Ibrahim, M. Kamat, M. Salleh, S.R.A. Aziz, Secure E-voting with blind signature, NCTT 2003 (2003) 193–197.
[20] M. Jakobsson, M. Juels, R.L. Rivest, Making mix nets robust for electronic voting by randomized partial checking, USENIX Security Symposium, 2002, pp. 339–353.
[21] C.T. Li, M.S. Hwang, C.Y. Liu, An electronic voting protocol with deniable authentication for mobile ad hoc networks, Computer Communications 31 (10) (2008) 2534–2540.
[22] V. Morales-Rocha, M. Soriano, J. Puiggalí, New voter verification scheme using pre-encrypted ballots, Computer Communications 32 (7–10) (2009) 1219–1227.
[23] R. Moreno, J. Pujolàs, P. Sanz, M. Serio, Mix verificables con pares ElGamal y curvas elípticas. VI Jornadas de matemática discreta y algorítmica, 2008, pp. 469–476 (in Spanish).
[24] National Institute of Standards and Technology. Recommendation for key management – part1: general. Special Publication 800-57, 2005.
[25] National Security Agency. The case for elliptic curve cryptography, 2009. Available from: <www.nsa.gov/business/programs/elliptic_curve.shtml>.
[26] A. Neff, Verifiable mixing (shuffling) of ElGamal pairs, 2004. Available from: <http://www.votehere.net/documentation/vhti>.
[27] L. Nguyen, R. Safavi-Naini, K. Kurosawa, Verifiable shuffles: a formal model and a paillier-based efficient construction with provable security, Lecture Notes in Computer Science 3089 (2004) 61–75.
[28] K. Peng, R. Aditya, C. Boyd, E. Dawson, B. Lee, Multiplicative homomorphic e-voting, Lecture Notes in Computer Science 3348 (2004) 61–72.
[29] K. Peng, C. Boyd, E. Dawson, Simple and efficient shuffling with provable correctness and ZK privacy, Lecture Notes in Computer Science 3621 (2005) 188–204.
[30] K. Peng, C. Boyd, E. Dawson, K. Viswanathan, A correct, private and efficient mix network, Lecture Notes in Computer Science 2947 (2004) 439–454.
[31] O. Schirokauer, D. Weber, T. Denny, Discrete logarithms: the effectiveness of the index calculus methods, Lecture Notes in Computer Science 1122 (1996) 337–362.
[32] Tor: anonymity online, http://www.torproject.org.
[33] D. Wikström, Five practical attacks for "optimistic mixing for exit-polls", Lecture Notes in Computer Science 3006 (2004) 160–175.

Please cite this article in press as: F. Sebé et al., Simple and efficient hash-based verifiable mixing for remote electronic voting, Comput. Commun. (2009), doi:10.1016/j.comcom.2009.11.013