An Efficient and Highly Sound Voter Verification Technique and Its Implementation

Rui Joaquim[1] and Carlos Ribeiro[2]

[1] Inesc-ID \ISEL
Rua Conselheiro Emídio Navarro 1, 1959-007 Lisboa, Portugal
rjoaquim@cc.isel.pt
[2] Inesc-ID \UTL
Rua Alves Redol 9 - 6 andar, 1000-029 Lisboa, Portugal
carlos.ribeiro@ist.utl.pt

Abstract. This paper presents MarkPledge3 (MP3), the most efficient specification of the MarkPledge (MP) technique. The MP technique allows the voter to verify that her vote is correctly encrypted with a soundness of $1-2^{-\alpha}$, with $20 \leq \alpha \leq 30$, just by performing a match of a small string (4-5 characters). Due to its simplicity, verifying the election public data (vote encryptions and tally) in MP3 is 2.6 times faster than with MP2 and the vote encryption creation on devices with low computational power, e.g. smart cards, is approximately 6 times better than the best of the previous MP specifications (MP1 and MP2).

Keywords: Verifiability, Voter Vote verification, MarkPledge.

1 Introduction

The MarkPldege (MP) technique was introduced by Neff in 2004 [22] with the goal of providing high vote encryption assurance to the voter, i.e. give the voter high certainty that the encrypted vote, generated by the voting machine, is an encryption of the voter's choice. In its essence MP defines how to encrypt two types of votes: a vote in favor of a candidate, a $YESvote$, and a vote against/neutral to a candidate, a $NOvote$. The MP candidate vote encryption is special because it contains random data that is used to create a verification code, which can to prove to the voter the type of the candidate vote encryption. The voter verifies that a candidate vote encryption is in fact a $YESvote$ by doing a short string match. The verification of a $NOvote$ usually requires some extra effort from the voter, but can be made unneeded by the specific vote protocol where it is used.

In MP based vote protocols [1,3,4,19,22], the voter's choice is encrypted with a $YESvote$, for the selected candidate, and with several independent $NOvote$s for the non selected candidates. Then, a vote receipt is created with the verification codes of all candidate vote encryptions. To simplify the voter's receipt verification, the vote protocol provides a mathematical proof that there is only one $YESvote$ in the set of candidate encryptions, therefore the voter only needs to verify the $YESvote$ candidate encryption.

A. Kiayias and H. Lipmaa (Eds.): VoteID 2011, LNCS 7187, pp. 104–121, 2012.

The soundness of the voter verification process is $1 - 2^{-\alpha}$, where α is a configurable security parameter that defines the size in bits of the verification code to match. To achieve a usability *vs* security balance, α is usually set to a value between 20 and 30, corresponding to a verification code of 4 to 5 characters.

The high soundness of the voter's receipt verification is only guaranteed if the vote encryption is valid and the vote receipt correct, i.e. if there is only one $YESvote$ and if all verification codes match the corresponding individual candidate vote encryptions. However, the proofs of vote validity and receipt correctness require some complex math, which the common voter cannot perform. Thus, the MP technique, and the vote protocols that use it, define public verifiable vote validity and receipt correctness proofs to protect the voter's privacy. Anyone with the sufficient knowledge and computational power can verify the validity and correctness of all vote-receipt pairs.

Our major contribution is a faster MP solution (MP3) that can be proven to be as sound and privacy-keeping as any of previous MP solutions [4,22], without consuming more memory. Both previous MP solutions [4, 22] have high computational vote generation costs, which makes them unsuitable to be used in mobile voting scenarios where the voting machine has low computational power, e.g. a smart-card or a secure element of a mobile phone, both usually standard JavaCards. MP3 also offers a considerable 2.6 times improvement on the public vote-receipt validity and correctness verifications over the best previous solution (MP2). This improvement enlarges the number of public organizations with enough computer power to verify all the votes of a national general election .

Our second contribution is an abstraction layer for the MP technique, composed of 5 functions: the vote encryption function $\mathcal{VE}_{pk}$, which creates the candidate vote encryption; the vote receipt creation function $\mathcal{RC}_{pk}$, which given a candidate encryption generates the corresponding verification code; the vote validity function $\mathcal{VV}_{pk}$, which verifies the validity of a candidate encryption; the receipt validity function $\mathcal{RV}_{pk}$, which validates the correspondence between a candidate vote encryption and a verification code; and, finally, a canonicalization function $\mathcal{C}_{pk}$ which prepares the candidate vote encryption for the vote tally process. The MP abstraction layer adds nothing to the MP solutions (MP1, MP2 and MP3) or to the MP based vote protocols. It only identifies common processes to all MP solutions, thus, it facilitates the comparison of the different MP solutions and their substitution in a MP based vote protocol.

We have partially implemented each one of the three MP solutions (MP1, MP2 and MP3) in two types of smart-cards, a MULTOS smart card and a JavaCard. The former is faster, but the latter is more ubiquitous, being deployed in secure elements of recent mobile phones and many National Identity Cards. In both cases MP3 is the only viable solution given that the time required to vote with MP1 and MP2 exceeds the time a user will be, usually, willing to wait.

The next section presents the related work and describes the simplified version of a MP vote protocol. Sections 3 and 4 describe the new MP3 proposal and present a detailed description of its cryptographic functions. Section 5 provide a comparative analysis of all MP solutions. Finally, we conclude in section 6.

2 Related Work

The research on electronic voting protocols always had the goal to provide verifiable protocols [7,8,13,14,16,17,20,24]. However, the verification procedures are usually too difficult for a human to do, requiring the use of a trusted Vote Machine (VM) to help the voter in the process. In 2004, Chaum [9] and Neff [22] (MarkPledge) introduced techniques that enable a human to verify a cryptographic vote, eliminating the need for a trusted VM.

In the original Chaum's work [9] the voter verifies her vote through a two-sheet ballot print, by a special printer, that uses transparent sheets and visual cryptography to show the voter choices in a human readable format. The voter then destroys one sheet and keeps the other as a verifiable privacy-preserving receipt. This procedure models a simple "cut-and-choose" technique giving the voter a $\frac{1}{2}$ probability to detect a fraud. Punchscan [10] and Pret a Voter [12] simplify the Chaum's system setup by using a simpler pre-printed ballot. In both systems the voter still have only a probability of $\frac{1}{2}$ to detect any fraud with her vote. Both systems allow for pre-election cut-and-choose verification aiming to reduce the danger of a large scale fraud. The verification procedure consists in printing an excess of ballots and then randomly auditing the extra ballots.

Adida and Neff [3] and Joaquim et al. [19] present simplifications to the voter interaction of MP, improving its usability. In 2006, and also based on the MP construction, Moran and Naor presented a voter verifiable voting system with everlasting privacy [21], replacing the vote encryption with vote commitments. The main disadvantage of the original MP specification (MP1) is the high computational costs of the technique, specially when compared to a vote protocol that encrypts the vote as a simple encryption of the candidate identifier. In MP1 the vote is encrypted with $2 \cdot \alpha$ encryptions for each candidate. A direct consequence of the MP1 vote encryption structure is that the vote encryption needs $2 \cdot \alpha \cdot k$ more disk space when compared to a simple vote encryption, where k is the number of running candidates. The performance issues of MP were first addressed in MP2 [4]. However, MP2 is still too heavy for low computational power devices (cf. section 5.1).

A completely different voter verification approach was proposed by Benaloh in [5, 6]. His proposal consists in separating the vote encryption process from the vote casting process. The VM is responsible only for the vote encryption, which it delivers to the voter (e.g. in a paper receipt). The voter can then choose to cast the vote or to verify it by asking to decrypt the encrypted vote. In this solution the voter can verify as many vote encryptions as she wants until she gains confidence in the VM. In theory, this approach can have the same $1 - 2^{-\alpha}$ MP soundness if a voter is allowed to use the VM to create 2^{α} independent vote encryptions. This procedure is clearly unpractical and would yield a computational cost much higher than the one of the original MP. The ideas of Benaloh's work were used for the voter verification mechanisms used by the VoteBox [25] and Helios [2] voting systems.

2.1 MarkPledge Simplified Vote Protocol Overview

Usually, a voter has no way to be assured that the cryptographical representation of her vote, produced by the voting machine, encodes her candidate choice. MP attains that goal by performing a zero-knowledge proof (ZKP) with the voter herself, not with some proxy, that the encrypted vote for the chosen candidate is, in fact, a $YESvote$. In order to ensure receipt-freeness a simulated ZKP is conducted for every other candidate encryption ($NOvotes$), in such a way that only the voter is able to tell which is the real proof among the simulated ones.

More precisely, zero-knowledge, in the MP context, means that it is not possible to identify the type of a candidate encryption from the corresponding public data: candidate encryption, verification code and corresponding mathematical proofs of vote validity and verification code correctness. To protect the voter's privacy, the correctness verification is the same for every candidate encryption, whether it bares a $YESvote$ or a $NOvote$. In fact, the correctness verification is, in both cases, a ZKP that aims to prove that the candidate encryption bares a $YESvote$. The proof is only real if the candidate encryption bares a $YESvote$. This is because only the commit value (verification code) of the ZKP of the $YESvote$ is shown (pledged) to the voter. The commit values (verification codes) of the ZKP of the $NOvotes$ are not shown to anyone a may therefore be crafted so that the ZKP on $NOvotes$ seem real although they are simulated.

To clarify the MP technique use within a vote protocol follows a short description of the simplified vote protocol of [3]. Note however, that each MP version (MP1, MP2 and MP3) may be used in different voting protocols, e.g. the original one [22], the simplified versions [3,22], and the Internet protocol of [19]. The described MP vote protocol has four phases [3]: the first three match the usually ZKP (commit, challenge, validation), and the last one is an anonimization and counting step. The following protocol description also introduces the MP abstraction layer, i.e. it shows where each of the five MP functions are used. The specification of each function is given in section 4, for the MP3 solution, and in an extended version of this paper [18], for the MP1 and MP2 solutions.

Phase 1. Vote Encryption

1. The vote machine (VM) presents the list of candidates to the voter.
2. The voter enters her vote selection (candidate j).
3. The VM, using the candidate vote encryption function $\mathcal{VE}_{pk}$, creates the vote encryption as a sequence of individual MP candidate vote encryptions (dubbed as bit encryptions, $BitEnc(b)$ in [1,3,4,19]). The selected candidate (candidate j) gets a $YESvote = BitEnc(1)$, and each other candidate gets an individual $NOvote = BitEnc(0)$.
   Each candidate vote encryption $BitEnc(b)_i$ encrypts a random commit code θ_i, which later allows the voter to verify the vote encryption by matching the verification code ϑ_j in the vote receipt. At this point in the protocol only the verification code of the $YESvote$ is known $\vartheta_j = \theta_j$.
4. The VM commits to the vote encryption, e.g. by printing or publishing it in a public bulletin board.

5. The VM pledges (reveals on an untappable channel) to the voter the $YESvote$ verification code $\vartheta_j = \theta_j$ as the *pledge* value.

Phase 2. Receipt Creation and Voter Verification

1. The voter sends a random vote challenge c to the VM. Originally the challenge value was chosen by the voter herself [22]; subsequent versions remove this task from the voter with the help of a trusted third party [3, 19].
   This step is crucial for the voter's verification soundness because the random challenge is what prevents the VM to pledge a valid verification code for a $NOvote$ in the previous protocol step.
2. The VM, using the candidate receipt creation function $\mathcal{RC}_{pk}$, computes the verification codes for the non-selected candidates ($\vartheta_i : i \neq j$), i.e. the verification codes for all the $NOvote$s. Each computed verification code $\vartheta_i : i \neq j$ is the result of a function between the random commit code θ_i inside the corresponding $NOvote_i$ and the challenge value c.
3. The VM prints/publishes in a public bulletin board the receipt as the sequence of all the verification codes ϑ_i, in the same order of the candidate vote encryptions in the vote encryption. Along with the vote receipt, the VM publishes the data necessary to verify the vote validity ($voteValidity_i$) and the receipt correctness (ω_i). The $voteValidity_i$ and ω_i are, respectively, outputs of the $\mathcal{VE}_{pk}$ and $\mathcal{RC}_{pk}$ functions.
4. The voter verifies the correction of the vote encryption by verifying if the *pledge* value, pledged in step 5 of phase 1, matches the verification code ϑ_j associated to her chosen candidate (candidate j) in the vote receipt.

Phase 3. Third Party Vote/Receipt Validation

To certify the voter receipt verification, one or several third parties validate the vote/receipt pair validity and correctness, using the vote validity ($\mathcal{VV}_{pk}$) and receipt validity ($\mathcal{RV}_{pk}$) functions. The $\mathcal{VV}_{pk}$ function attests that each $BitEnc(b)_i$ is valid, i.e that it is either a $BitEnc(0)$ or a $BitEnc(1)$. The $\mathcal{RV}_{pk}$ function attests that each verification code ϑ_i corresponds to $BitEnc(b)_i$. Both certifications are performed only on public data, thus they do not compromise the voter's privacy. Optionally, using the techniques described in section 4.1, the third parties can also verify that the encrypted vote has only one $YESvote$. Without this verification the VM can create invalid votes, i.e. votes with more than one $YESvote$. Although, several MP based vote protocols omit this verification step and only verify that there is only one $YESvote$ in the vote encryption in the tally phase, after the anonymous vote decryption.

Phase 4. Vote Canonicalization and Counting

Finally, the vote encryptions are made uniform, by the vote canonicalization function $\mathcal{C}_{pk}$, and then anonymized and counted. The results are published in a public bulletin board.

The canonicalization process is necessary because every candidate vote encryption encrypts a random one-time commit code (θ_i), that is used to deterministically compute the verification codes of the $BitEnc(b)$s. Thus, revealing the θ_i commit codes would enable to identify the voter's candidate choice by a simple correlation with the verification codes ϑ_i in the voter's vote receipt.

The vote canonicalization process is public and therefore verifiable. Once all vote encryptions are in the canonical form it is possible to decrypt them and compute the election vote tally, usually using a mix-net or a homomorphic vote tally process to protect the voter's privacy.

The $BitEnc(b)$, $voteValidity$ and ω details, and the procedures to verify the vote validity and receipt correctness, are presented in sections 3 and 4, for the MP3 solution, and in the extended version of this paper [18] for the MP1 and MP2 solutions.

3 MarkPledge3

The key aspect of every MP system is the two step verification of the vote encryption, carried by the VM to the voter, to prove that the encrypted vote expresses the voter's intentions, cf. phases 2 and 3 of the protocol described in section 2.1. This section describes the key insight of this two step proof verification in MP3. It starts by describing the cryptosystem used by MP3, and continues by showing how its homomorphic properties are used to implement the candidate vote encryption, the proof and the two step proof validation.

3.1 Homomorphic Cryptosystem Details

The MP3 implementation is based on the homomorphic properties of the ElGamal cryptosystem [15]. The ElGamal cryptosystem works in the $\mathbb{Z}_p^*$ subgroup G_q of order q, where p and q are large primes such that $q|p-1$. Both primes p, q and a generator g of G_q are public parameters of the system. The ElGamal key pair consist of a private key s and the corresponding public key $h = pk = g^s \bmod p$. The private key s is a randomly chosen integer such that $0 < s < q$. Algorithms to generate secure ElGamal parameters can be found in [23].

A message $m \in G_q$ is encrypted by selecting a random integer value $r \in \mathbb{Z}_q$, and constructing the following pair $\mathcal{EG}_h(m, r) = \langle x, y\rangle = \langle g^r \ mod\ p, h^r \cdot m \ mod\ p\rangle$. To recover the message m one computes $m = \frac{y}{x^s}$. In order to have the desired homomorphic properties we use a variant known as exponential ElGamal [14]. In exponential ElGamal the message to encrypt m is chosen from $\mathbb{Z}_q$ and it is encrypted as g^m, instead of m, in order to respect the ElGamal message space, i.e. $\mathcal{E}_h(m, r) = \mathcal{EG}_h(g^m, r)$. The exponential ElGamal has the following homomorphisms (we have omitted the $mod\ p$ notation from the equations):

Additive homomorphism between two encryptions

$$\mathcal{E}_h(m_1,\ r_1) \oplus \mathcal{E}_h(m_2,\ r_2) = \langle g^{r_1},\ h^{r_1} \cdot g^{m_1}\rangle \cdot \langle g^{r_2},\ h^{r_2} \cdot g^{m_2}\rangle =$$
$$\langle g^{r_1} \cdot g^{r_2},\ h^{r_1} \cdot g^{m_1} \cdot h^{r_2} \cdot g^{m_2}\rangle = \langle g^{r_1+r_2},\ h^{r_1+r_2} \cdot g^{m_1+m_2}\rangle =$$
$$\mathcal{E}_h((m_1 + m_2)\ mod\ q,\ (r_1 + r_2)\ mod\ q)$$

Prover		Verifiers	
Vote Machine		Voter	Third Party
$b' \in \{-1, 1\}$			
$\tau, \delta, \theta \in_R \mathbb{Z}_q$			
$u \leftarrow \langle g^\tau, h^\tau \cdot g^{b'} \rangle$			
$v \leftarrow \langle g^\delta, h^\delta \cdot g^\theta \rangle$			
	$\xrightarrow{\text{pledge}}$		
	$\xrightarrow{u, v}$		
		$c \in_R \mathbb{Z}_q$	
	$\xleftarrow{c}$		
$\vartheta \leftarrow \frac{b' \cdot c - c + \theta}{b'}$			
$\omega \leftarrow \tau \cdot (c - \vartheta) + \delta$			
	$\xrightarrow{\vartheta, \omega}$		$\xrightarrow{u, v, c, \vartheta, \omega}$
		pledge $\stackrel{?}{=} \vartheta$	$u^{c-\vartheta} \cdot v \stackrel{?}{=} \langle g^\omega, h^\omega \cdot g^c \rangle$

Fig. 1. MP3 $YESvote$ ($b' = 1$) candidate encryption and verification protocol. The $NOvote$ ($b' = -1$) case is identical, the only difference is that the voter verifies that pledge $= 2 \cdot c - \vartheta$ instead of pledge $= \vartheta$, cf. section 3.2. ϑ and ω are computed *mod* q.

Multiplicative Homomorphism between an Encryption and a Value n

$$\mathcal{E}_h(m,\ r) \otimes n = \langle g^r,\ h^r \cdot g^m \rangle^n = \langle (g^r)^n,\ (h^r)^n \cdot (g^m)^n \rangle = \langle g^{r \cdot n},\ h^{r \cdot n} \cdot g^{m \cdot n} \rangle = \mathcal{E}_h((m \cdot n)\ mod\ q,\ (r \cdot n)\ mod\ q)$$

3.2 MarkPledge3 Insights

Figure 1 depicts the MP3 candidate vote encryption and verification. A MP3 candidate vote encryption $BitEnc(b) = \langle u, v \rangle$ is composed by two independent encryptions: u is the encryption of either $b' = 1$ for a $YESvote$ or $b' = -1$ for a $NOvote$; v is the encryption of a random commit code (θ), which, in the case of a $YESvote$, is pledged over an untappable channel to the voter as being the corresponding verification code.

After the encrypted vote creation by the vote machine, the voter selects a random challenge c that will be used by the vote machine to create a public verifiable receipt (ϑ, ω). ϑ is the public verification code and ω the public data that allows to prove that ϑ is the correct verification code for the pair (u, v). ϑ correctness is guaranteed by an independent third party verification of the following equality: $u^{c-\vartheta} \cdot v = \langle g^\omega, h^\omega \cdot g^c \rangle = \mathcal{E}_{pk}(c, \omega)$. Thus, assuming that u encrypts a value $b' \in \{-1, 1\}$, which can be proved by known techniques, cf. section 4, the preceding equality and the homomorphic properties of the ElGamal cryptosystem guarantee that $\vartheta \in \{\theta, 2 \cdot c - \theta\}$, cf. equation 1. The encryption factor ω can be easily computed by the vote machine as $\omega = \tau \cdot (c - \vartheta) + \delta$.

$$u^{c-\vartheta} \cdot v = \mathcal{E}_{pk}(c, \omega) \Rightarrow$$
$$b' \cdot (c - \vartheta) + \theta = c \Leftrightarrow \vartheta = \frac{b' \cdot c - c + \theta}{b'} \quad (1)$$

thus:

$$b' = 1 \Rightarrow \vartheta = \theta \quad \wedge \quad b' = -1 \Rightarrow \vartheta = 2 \cdot c - \theta$$

By equation 1, before knowing the challenge value c, the vote machine can only pledge the verification code ϑ for a $YESvote$ ($b' = 1$), which is the random commit code θ encrypted in v. Thus, the voter receipt verification for a $YESvote$ is just the verification that pledge $= \vartheta$, which is a simple string match.

A $NOvote$ verification requires some extra work for the voter. In this case, the vote machine pledges the value θ and the voter must reconstruct the verification code $\vartheta = 2 \cdot c - \theta$. Although, the MarkPledge based vote protocols usually do not require the voter to perform this verification by providing a proof that there is only one $YESvote$ candidate encryption in the vote, cf. sections 1 and 2.1. The details on how to create such proof for MP3 are presented in section 4.1.

The voter verification, as described in this section, requires the match of a long string by the voter, i.e. both the pledge value and the verification code domains are $\mathbb{Z}_q$. This usability issue is addressed in section 4.2.

Finally, note that in the special case where $c = \theta$ nothing is proved about the value of b'. However, assuming that the c value is selected randomly from a large domain this possibility is negligible.

3.3 MarkPledge3 Soundness and Zero-Knowledge Properties

This section provides proof sketches for the MP3 soundness and zero-knowledge properties. The proof sketches follow the MP3 candidate encryption and verification protocol described in figure 1.

Theorem 1. *Under the semantic security of the ElGamal cryptosystem, MP3 offers a verification soundness of* $1 - 2/q$*, provided that the value* u *is a valid encryption, i.e.* u *encrypts either* $b' = 1$ *or* $b' = -1$*.*

Proof Sketch. In order to prove the soundness of MP3, we will show that the probability that the voting machine or any other adversary have to cheat the voter in believing that she has issued a $YESvote$ while she has actually issued a $NOvote$ is $1 - 2/q$. The converse probability of cheating the voter in believing that she had issued a $NOvote$ while she had issued a $YESvote$ may be easily shown to also be $1 - 2/q$.

Assuming a valid u, the validation of the equality $u^{c-\vartheta} \cdot v = \langle g^{\omega}, h^{\omega} \cdot g^{c} \rangle$ ensures by equation 1 that $\vartheta = \theta$ if $b' = 1$ ($YESvote$), and that $\vartheta = 2 \cdot c - \theta$ if $b' = -1$ ($NOvote$). Given that the voter checks that the pledge of her's vote is pledge $= \vartheta$, the only way the vote machine or another adversary have to fool the voter is to pledge to the voter a value pledge $= \vartheta = 2 \cdot c - \theta$, without knowing the challenge c (the challenge is revealed only after the pledge). Since $c \in_{\mathcal{R}} \mathbb{Z}_q$

it is trivial to note that pledge $= 2 \cdot c - \theta$ would also have a random distribution in $\mathbb{Z}_q$. Therefore, the vote machine has only a probability of $1/q$ to guess such value. Finally, removing the case where nothing is proved about b', when $c = \theta$, we can conclude that MP3 offers a vote verification soundness of $1 - 2/q$.

Theorem 2. *Under the semantic security of ElGamal cryptosystem, the candidate vote encryption verification is zero-knowledge to every one not knowing the pledge, provided that the challenge c is randomly and independently chosen from the pledge value.*

Proof Sketch. Assuming the semantic security of ElGamal cryptosystem neither u or v reveal anything about the vote encrypted in u. Identically, under the challenge c independence generation assumption, c reveals nothing about the value encrypted in u, or v. Given that $\omega = ((c - \vartheta) \cdot \tau + \delta)$ is a linear combination of the two ElGamal random factors, used to generate u and v, it can reveal one of the encryptions u or v but only if the other one is broken. Since no individual public factor used by the verification code correctness proof reveals the vote, the only other solution is testing all together with the verification code validation equation: $u^{c-\vartheta} \cdot v = \langle g^{\omega}, h^{\omega} \cdot g^{c} \rangle$. However, by equation 1, for the same combination of public values u, v, c, ϑ and ω there is always two possible scenarios for which the equation holds, i.e. the scenario in which u encrypts $b' = -1$ ($NOvote$) and v is the encryption of $\theta = 2 \cdot c - \vartheta$ is indistinguishable from the scenario in which u encrypts $b' = 1$ ($YESvote$) and v encrypts $\theta = \vartheta$.

4 MarkPledge 3 Functions Details

In order to better compare MP3 with MP1 and MP2, this section presents the MP abstraction layer, along with its MP3 specification. The MP abstraction layer is composed by a set of five functions that all MP solutions have, although these functions in the previous MP specifications were implicit in the vote protocols. Due to space constrains the functions implementations for MP1 and MP2 are presented in an extended version of this paper [18].

The first two functions execute the candidate vote encryption ($\mathcal{VE}_{pk}$) and the receipt (verification code) creation ($\mathcal{RC}_{pk}$). The first one corresponds to the $BitEnc(b)$ encryption in MP1 and MP2, whilst the second one bares no name in MP1 and MP2 previous descriptions. The next two functions are the candidate vote ($\mathcal{VV}_{pk}$) and receipt ($\mathcal{RV}_{pk}$) validation functions. Finally, the last function ($\mathcal{C}_{pk}$) prepares the candidate votes for an anonymous vote tally process with a canonization process. All functions use the election's public key pk, which is usually generated by a set of trustees using a threshold scheme.

Additionally, in section 4.1 it is described how to verify that a set of candidate votes contains only one $YESvote$ and how to perform a verifiable homomorphic vote tally with MP3. Finally, section 4.2 shows how to adjust the ϑ verification code to a size that is easily usable by humans.

Vote Encryption

$$\mathcal{VE}_{pk}(b, \theta, r) = \langle BitEnc(b), voteValidity \rangle$$

Where:

$$BitEnc(b) = \langle u, v \rangle = \langle \mathcal{E}_{pk}(b', \tau) = g^{\tau}, h^{\tau} \cdot g^{b'}, \mathcal{E}_{pk}(\theta, \delta) \rangle$$

$$b' = \begin{cases} 1 & \text{if } b = 1 \ (YESvote) \\ -1 & \text{if } b = 0 \ (NOvote) \end{cases}$$

$$r = \tau \parallel \delta, \ h = \textit{election public key } (pk)$$

$$voteValidity = \langle a_1, a_2, b_1, b_2, d_1, d_2, r_1, r_2, c \rangle$$

$$\text{if } b = 1 \ (YESvote) \begin{cases} \sigma, r_1, d_1 \in_{\mathcal{R}} \mathbb{Z}_q \\ a_1 = g^{r_1 + \tau \cdot d_1}, \ a_2 = g^{\sigma} \\ b_1 = h^{r_1 + \tau \cdot d_1} \cdot g^{2 \cdot b' \cdot d_1}, \ b_2 = h^{\sigma} \\ d_2 = c - d_1, \ r_2 = \sigma - \tau \cdot d_2 \end{cases}$$

$$\text{if } b = 0 \ (NOvote) \begin{cases} \sigma, r_2, d_2 \in_{\mathcal{R}} \mathbb{Z}_q \\ a_1 = g^{\sigma}, \ a_2 = g^{r_2 + \tau \cdot d_2} \\ b_1 = h^{\sigma}, \ b_2 = h^{r_2 + \tau \cdot d_2} \cdot g^{2 \cdot b' \cdot d_2} \\ d_1 = c - d_2, \ r_1 = \sigma - \tau \cdot d_1 \end{cases}$$

$$c = Hash(u, a_1, a_2, b_1, b_2)$$

The vote encryption function $\mathcal{VE}_{pk}$ is used in the first phase of a vote protocol to generate each candidate vote encryption, cf. step 3 of the simplified MP vote protocol presented in section 2.1. In MP3 each candidate vote $BitEnc(b)$ is a simple pair of exponential ElGamal encryptions, dubbed u and v. The first one encrypts a value $b' \in \{-1, 1\}$ accordingly to the value of b and the second is just the encryption of $\theta \in_{\mathcal{R}} \mathbb{Z}_q$. Both encryptions use exponential ElGamal with the randomization factors τ and δ derived from the input value $r = \tau \parallel \delta$. The $voteValidity$ data proves that u is an ElGamal exponential encryption of a value $b' \in \{-1, 1\}$. In MP3 it consists in the output of the ballot validity proof protocol of Cramer et al. [14]. In its original context, the Cramer et al. protocol proves that a vote is an ElGamal exponential encryption of a message $m \in \{-1, 1\}$, which is exactly our definition of a valid u.

Receipt Creation

$$\mathcal{RC}_{pk}(BitEnc(b), r, c) = \langle \vartheta, \omega \rangle$$

Where:

$$BitEnc(b) = \langle \mathcal{E}_{pk}(b', \tau), \mathcal{E}_{pk}(\theta, \delta) \rangle$$

$$r = \tau \parallel \delta, \ c \in_{\mathcal{R}} \mathbb{Z}_q$$

$$\vartheta = \begin{cases} \theta & \text{if } b' = 1 \ (YESvote) \\ 2 \cdot c - \theta \ mod \ q & \text{if } b' = -1 \ (NOvote) \end{cases}$$

$$\omega = \tau \cdot (c - \vartheta) + \delta \ mod \ q$$

In the simplified protocol of section 2.1, the $\mathcal{RC}_{pk}$ function is used in step 2 of the protocol's second phase, after the pledge has been shown to the voter and after the challenge c disclosure to the vote machine. This function generates a receipt (verification code ϑ) as explained in section 3.2, i.e. it outputs the random commit code θ, if the candidate vote is a $BitEnc(1)$ (i.e. a $YESvote$), or outputs the θ symmetric value, taking c as the symmetry axis, if the candidate vote is a $BitEnc(0)$ (i.e. a $NOvote$). The ω data is the combination of the randomization factors used in the u and v encryptions, which is needed to verify that the verification equation results in an encryption of the challenge value c, as described by the $\mathcal{RV}_{pk}$ function below. In order to work in accordance with the ElGamal homomorphic properties, both the ϑ and ω values are computed *mod* q.

Vote Validity

$$\mathcal{VV}_{pk}(BitEnc(b), voteValidity) = validity$$

Where:

$$BitEnc(b) = \langle u = \langle x, y\rangle, v\rangle,\ voteValidity = \langle a_1, a_2, b_1, b_2, d_1, d_2, r_1, r_2, c\rangle$$

$$validity = \begin{cases} True & \text{if } c = Hash(u, a_1, a_2, b_1, b_2) \\ & \quad \wedge\ c = d_1 + d_2 \\ & \quad \wedge\ a1 = g^{r_1} \cdot x^{d_1} \\ & \quad \wedge\ a2 = g^{r_2} \cdot x^{d_2} \\ & \quad \wedge\ b1 = h^{r_1} \cdot (y \cdot g)^{d_1} \\ & \quad \wedge\ b2 = h^{r_2} \cdot (y \cdot g^{-1})^{d_2} \\ False & \text{otherwise} \end{cases}$$

$$h = election\ public\ key\ (pk)$$

The $\mathcal{VV}_{pk}$ function corresponds to the Cramer et al. ballot validity proof [14]. It is used to ensure that the u component of the candidate vote ($BitEnc(b)$) is in fact the encryption of $b' = 1$ or $b' = -1$, i.e. it is a valid candidate vote. The function outputs true if the candidate vote is valid and false if it is invalid. In the simplified MP protocol of section 2.1, this function can be used immediately after the first phase of the protocol, to ensure the correctness of the vote as soon as possible, or at the end of the voting process in phase 3.

Receipt Validity

$$\mathcal{RV}_{pk}(BitEnc(b), c, \vartheta, \omega) = validity$$

Where:

$$BitEnc(b) = \langle u, v\rangle$$

$$validity = \begin{cases} True & \text{if } \phi = \mathcal{E}_{pk}(c, \omega) = u^{c-\vartheta} \cdot v \\ False & \text{otherwise} \end{cases}$$

The receipt validity function corresponds to the zero knowledge verification code ϑ validation, which can be conducted by any trusted third party to prove that

$u^{c-\vartheta} \cdot v$ is the encryption of c, without any special knowledge but the public values. This is possible by a reconstruction of the c encryption using the ω encryption factor, revealed by the $\mathcal{RC}_{pk}$ function. From equation 1, proving that $u^{c-\vartheta} \cdot v = \mathcal{E}_{pk}(c, \omega)$ is enough to prove that $\vartheta = \theta$ if $b' = 1$ or that $\vartheta = 2 \cdot c - \theta$ if $b' = -1$, which is enough to complement the voter's verification, unless $c = \theta$ where in both cases $\vartheta = \theta$. However, since c and θ are chosen randomly from $\mathbb{Z}_q$, this case probability is negligible. The $\mathcal{RV}_{pk}$ function is used in the 3rd phase of the simplified MP vote protocol presented in section 2.1.

An alternative to the encryption reconstruction method, it is possible to prove that $u^{c-\vartheta} \cdot v = \mathcal{E}_{pk}(c, \omega)$ is an encryption of the value c without revealing ω using the Chaum and Pedersen protocol for proving the equality of discrete logarithms [11]. The Chaum and Pedersen protocol can be used to prove the encryption $\mathcal{E}_{pk}(c, \omega) = \langle x, y \rangle$ proving that $log_g\ x = log_h\ y/(g^c)$.

Canonicalization

$$\mathcal{C}_{pk}(BitEnc(b), c, \vartheta) = \langle canonicalVote \rangle$$
$$Where:$$
$$BitEnc(b) = \langle u, v \rangle\ \wedge\ canonicalVote = u$$

The MP3 candidate vote canonicalization is very simple because the u element of the $BitEnc(b)$ is already an encryption of a fixed value, $b' \in \{-1, 1\}$, that depends on the candidate vote type, i.e. $\forall\ BitEnc(1) : b' = 1$ and $\forall\ BitEnc(0) : b' = -1$. Therefore, in MP3 the u encryption can be used directly as the $canonicalVote$ because after an anonymization process, e.g. mix-net anonymization, it can be safely decrypted without revealing any link to the voter, i.e. the only thing we will see are the 1 and -1 values. MP1 and MP2 require a slightly more complex vote canonicalization.

4.1 Homomorphic Vote Tally

If the vote protocol uses the vote validity proof (i.e. ensures that all votes either encrypt a 1 or a -1) it is possible to use the efficient homomorphic vote tally process of the Cramer et al. vote protocol [14], since the MP3 candidate vote encryption u is equal to its vote encryption construction. Therefore, instead of decrypting each vote before counting it, which requires a previous anonymization process for each vote (usually using a mix-net), the MP3 homomorphic counting process performs the homomorphic addition of every encrypted vote $vote^j = BitEnc(b)_1^j \parallel BitEnc(b)_2^j \parallel \ldots \parallel BitEnc(b)_k^j$, where $BitEnc(b)_i^j = \langle u_i^j, v_i^j \rangle, j = 1..n$, n is the number of valid votes and k is the number of candidates in each vote. Given that the vote validity function $\mathcal{VV}_{pk}$ ensures that each $u_i^j = \mathcal{E}_{pk}(1)$ or $u_i^j = \mathcal{E}_{pk}(-1)$, then the vote counting for candidate i will be $count_i = \frac{n+d_i}{2}$, where d_i is the decryption of the homomorphic addition $\bigoplus_{j=1}^{n} u_i^j$. However, to ensure democracy, the protocol must also guarantee that each vote is counted for only one candidate, which means that the system must ensure that there is only

Table 1. MarkPledge functions computational costs in *mod p* exponentiations. The MP2 $voteValidity$, $\mathcal{VV}_{pk}$ and $\mathcal{C}_{pk}$ values reflect our adjustments to the MP2 solution, cf. [18]. The MP2 matrix exponentiation, in *mod q*, is denoted as *me*.

	$\mathcal{VE}_{pk}$ [BitEnc(b)] + [$voteValidity$]	$\mathcal{RC}_{pk}$	$\mathcal{VV}_{pk}$	$\mathcal{RV}_{pk}$	$\mathcal{C}_{pk}$
MP3	[5] + [5]	0	8	5	0
MP1	$[4 \cdot \alpha] + [-]$	0	-	$2 \cdot \alpha$	$\approx \frac{\alpha}{2}$
MP1a	$[2 + 4 \cdot \alpha] + [5]$	0	$8 + 2 \cdot \alpha$	$2 \cdot \alpha$	0
MP2	$[6 + 1 \cdot me] + [8 + 1 \cdot me]$	$1 \cdot me$	8	$8 + 1 \cdot me$	$3 + 1 \cdot me$

one $u_i^j = \mathcal{E}_{pk}(1)$ in each vote. Once again, given that each u_i^j is the encryption of the value 1 or -1, it is only necessary to prove that $\bigoplus_{i=1}^{k} u_i^j = \mathcal{E}_{pk}(2-k)$, e.g. using the Chaum and Pedersen protocol for proving the equality of discrete logarithms [11] or by revealing the sum of the encryption factors of the u_i^j elements, as suggested for the validation of the c encryption in the $\mathcal{RC}_{pk}$ and $\mathcal{RV}_{pk}$ functions.

4.2 Adjusting the Voter's View of MP3 Output to the α Parameter

Usually, the MP security parameter α is set to a value between 20 and 30, which means that the voter must compare 4 to 5 character strings to verify that pledge $= \vartheta$. In MP3 the c, ϑ and θ domains, and consequently the pledge domain, are defined by the cryptosystem parameter q and not by α. Since the size of q is in the hundreds of bits range we clearly have a usability issue. To solve this usability issue we propose a change in the voter's view of the MP3 functions output, namely the voter's view of the verification code ϑ and pledge value should be truncated to α bits by applying the *mod* 2^α operation to the referred values. Assuming an uniform and random distribution of ϑ and θ over $\mathbb{Z}_q$, the voter verification has a statistical soundness of $1 - 2^{1-\alpha}$, just because $q \ggg 2^\alpha$, i.e. the voter still performs the verification of a random value uniformly distributed over $\mathbb{Z}_{2^\alpha}$, cf. [18]. The soundness is $1 - 2^{1-\alpha}$ and not $1 - 2^{-\alpha}$ due to the case where $c = \theta$, where nothing is proved about the value b' encrypted in u.

5 Evaluation

This section discusses the improvements of MP3 over previous MP proposals in terms of each of the described functions. We first give a theoretical comparison of the techniques and then present the times for real implementations on smart cards. Note that by implementing MP in smartcards we may provide to each voter her own mobile voting machine without forcing her to trust any hardware/software device.

Table 1 shows the computational cost of the different MP functions in terms of the number of exponentiations per function. The values for MP3 are accordingly to the MP3 functions definition of section 4, and the MP1, MP1a and MP2

values are accordingly to the correspondent functions definition in [18]. All MP solutions use the exponential ElGamal encryption, which requires 3 exponentiations. Although, when the message to encrypt is a constant the corresponding message exponentiation was not considered. This is the case for all ElGamal encryptions in MP1 and MP1a and for the u encryption of MP3.

As detailed in [18], in MP1 each candidate encryption corresponds to α pairs of ElGamal encryptions, which corresponds to $4 \cdot \alpha$ exponentiations. The MP1 $\mathcal{RV}_{pk}$ function performs α 1-out-of-2 cut-and-choose verifications, one for each of the α encryption pairs, which correspond to $2 \cdot \alpha$ exponentiations. The MP1 $\mathcal{C}_{pk}$ function consists in homomorphically inverting the value inside approximately $\alpha/2$ ElGamal encryptions.

The MP1a row in table 1 represents the variant of MP1 proposed by Adida in [1]. MP1a adds an extra ElGamal encryption to each candidate vote, which allows the use of the $voteValidity$ proof and verification proposed for MP3 in section 4. The MP1a $\mathcal{VV}_{pk}$ function also needs to attest that the extra encryption "match" each one of the base α MP1 encryption pairs, cf. [1, 18].

The MP2 solution, cf. [4, 18], encrypts each candidate vote as the encryption of a 2D vector, which corresponds to 2 ElGamal encryptions, one for each of the vector coordinates. To compute the 2D vector it is necessary to perform a modular matrix exponentiation in q. The matrix exponentiation costs turned out to have a huge impact on the MP2 real performance, cf. section 5.1, thus they where also included in table 1. The MP2 $voteValidity$ proof uses the same technique of MP1a and MP3, but needs some homomorphic vector algebra on the candidate vote encryption. The $\mathcal{RC}_{pk}$, $\mathcal{RV}_{pk}$ and $\mathcal{C}_{pk}$ costs also reflect the need of homomorphic vector algebra.

Excluding the matrix exponentiation, both MP2 and MP3 present an α order improvement over MP1(a) on the $\mathcal{VV}_{pk}$ and $\mathcal{RV}_{pk}$ functions, and on the candidate vote $BitEnc(b)$ creation. In the $\mathcal{C}_{pk}$ function both MP1a and MP3 are clearly better as they do not require any computation. The $\mathcal{RC}_{pk}$ implementation on all MP versions is very simple and do not require any complex operation, except in MP2 where it is required a matrix exponentiation. Even excluding the matrix exponentiation operations, the improvements of MP3 over MP2 in the $\mathcal{VE}_{pk}$ and $\mathcal{RV}_{pk}$ functions are noteworthy. In the $\mathcal{VE}_{pk}$ MP3 presents an 17% improvement in number of exponentiations that grows up to 28.5% with the vote validity, and in the $\mathcal{RV}_{pk}$ MP3 presents an improvement of 37.5%.

When compared in terms of disk usage, MP3 and MP2 present approximately an α factor improvement over the MP1 and MP1a. In MP3 and MP2 a $BitEnc(b)$ is composed by two ciphertexts, while in MP1 and MP1a a $BitEnc(b)$ is composed respectively by $2 \cdot \alpha$ and $1 + 2 \cdot \alpha$ ciphertexts.

5.1 Implementation Results

In order to verify the real performance impact of MP3 we have partially implemented all MP solutions (MP1, MP2 and MP3). The two main reasons that lead us to the implementation were: i) the difficulty to compare the matrix exponentiation (required by MP2) to the large integer modular exponentiations needed for

the ElGamal encryptions, which are the base of all MP solutions, and ii) the curiosity to see if any MP solution can actually be run on limited devices, such as smart cards or secure elements inside a mobile phone (usually also implemented as smart cards) which can open the door for new vote protocols based on the MP technique.

We have implemented the matrix exponentiation using the non-recursive exponentiation by squaring algorithm ("computation by powers of 2") described in [27]. We have optimized the matrix multiplication algorithm to take advantage of the special form of the $\mathrm{SO}(2, q)$ matrices used by MP2. The $\mathrm{SO}(2, q)$ test matrices, as defined by MP2, have elements in $\mathbb{Z}_q$ and are exponentiated to random exponents in $\mathbb{Z}_{q\pm1}$ (note that in MP2 the α bits vector indexes must be transformed into the corresponding exponents, which are uniformly distributed in $\mathbb{Z}_{q\pm1}$, cf. [4,18]). We have tested the matrix exponentiation both on a PC and on smart cards (JavaCard and MULTOS). In the PC we have implemented the algorithm in Java. Our test code is available on request.

Our implementation results show that on a modern computer (Intel i5 2400 3.1GHz CPU) the matrix exponentiation time for $|q| = 160$ is about the same of an integer modular 1024 bits exponentiation with a 160 bits exponent. Thus, the $\mathcal{VE}_{pk}$ function in MP3 presents an improvement of about 40% when compared to MP2. Moreover, the MP3 $\mathcal{RV}_{pk}$ and $\mathcal{C}_{pk}$ functions together, which are necessary to validate the election public data, are about 2.6 times faster than MP2. In other words, if with MP3 the election data verification takes one day it would take two and a half days with MP2.

Again, in a modern computer, the voter's perception of the different vote encryption times would be close to none, as a large integer modular exponentiation takes only a few milliseconds. However, our implementation shows that the same is not true for more limited devices, e.g. smart cards or secure elements inside a mobile phone (also implemented as smart cards). We have partially implemented the different MP solutions in two different smart cards technologies: a JavaCard v2.2.1 (JCOP 31 v2.2) and a MULTOS v4.2.1 smart card (MC1-36K-61). The JavaCard technology was selected because it is widely deployed and it is being integrated as the secure element in many mobile phones; however the JavaCard API for large integer modular arithmetic is very limited, which has a negative impact on the MP performance. Thus, we also use a MULTOS card with a full large integer modular arithmetic native API.

Due to the very restricted large integer modular JavaCard 2.2.1 API (only modular exponentiation is available through the RSA engine) we had to program the modular addition, subtraction and multiplication operations. The addition and subtraction operations were implemented exclusively in "software". On the other hand, the pure software approach for the modular multiplication was not viable. Thus, we use the formula $(a + b)^2 - (a - b)^2 = 4 \cdot a \cdot b$, as in [26]. This allowed us to perform the modular multiplication in an acceptable time, with the help of the JavaCard cryptographic processor; however it restricts the modulus size to values equal or above 512 bits. The "new" JavaCard 3 API still does not provide a large integer modular arithmetic support. On the other hand, the MULTOS card has all modular operations available in its native API.

Table 2. $\mathcal{VE}_{pk}$ times in a NXP JCOP 31 v2.2 (JavaCard v2.2.1) with parameters $|p| = 1024$, $|q| = 512$ and $\alpha = 24$

JCOP 31 v2.2 (JavaCard) MarkPledge $\mathcal{VE}_{pk}$ times ($\|p\| = 1024$, $\|q\| = 512$, $\alpha = 24$)			
	MP1a	MP2	MP3
$BitEnc(b)$	44.2 sec (1921%)	45 min (117391%)	2.3 sec (100%)
voteValidity	6.5 sec (100%)	45 min (41538%)	6.5 sec (100%)
Total	50.7 sec (576%)	90 min (30681%)	8.8 sec (100%)

Table 3. $\mathcal{VE}_{pk}$ times in a MULTOS v4.2.1 MC1-36K-61 smart card with parameters $|p| = 1024$, $|q| = 512$ and $|q| = 160$, and $\alpha = 24$

MC1-36K-61 (MULTOS) MarkPledge $\mathcal{VE}_{pk}$ times ($\|p\| = 1024$, $\|q\| = 512$, $\alpha = 24$)			
	MP1a	MP2	MP3
$BitEnc(b)$	29.4 sec (1729%)	1.5 min (5294%)	1.7 sec (100%)
voteValidity	2.6 sec (100%)	1.5 min (3462%)	2.6 sec (100%)
Total	32.0 sec (744%)	3.0 min (4186%)	4.3 sec (100%)

MC1-36K-61 (MULTOS) MarkPledge $\mathcal{VE}_{pk}$ times ($\|p\| = 1024$, $\|q\| = 160$, $\alpha = 24$)			
	MP1a	MP2	MP3
$BitEnc(b)$	22.8 sec (1900%)	8.2 sec (683%)	1.2 sec (100%)
voteValidity	1.6 sec (100%)	8.8 sec (550%)	1.6 sec (100%)
Total	24.4 sec (871%)	17.0 sec (607%)	2.8 sec (100%)

Tables 2 and 3 present the performance times of the MarkPledge $\mathcal{VE}_{pk}$ function on a smart card support. We only present times for this primitive as it is the only intensive cryptographic function that must be performed by the vote encryption device. The results for MP3 in Table 2 are real whilst for MP1a and MP2 were lower estimated. Table 3 presents real results for all MP versions.

The JavaCard times, in table 2, for the MP1a and MP2 are estimated as follows: the MP1a $BitEnc(b)$ generation time is estimated as 80% of $\alpha \cdot \langle MP3\ BitEnc(b)\ time \rangle$. Each of the MP1a α encryption pairs is composed of 2 ElGamal encryptions, just like the $BitEnc(b)$ of MP3. The 80% adjustment comes from the fact that it takes one less exponentiation because it encrypts two constant values, cf. [1, 18, 22] and table 1. The MP1a $voteValidity$ generation time is equal to the MP3 $voteValidity$ generation time, since both use exactly the same technique. The MP2 $BitEnc(b)$ generation time is estimated as $\langle MP3\ BitEnc(b)\ time \rangle + \langle matrix\ exponentiation\ time \rangle$. The MP2 $BitEnc(b)$ is also composed of 2 ElGamal encryptions, however the values to encrypt are the result of a matrix exponentiation and therefore this time must be added to the encryption time. The MP2 $voteValidity$ (as we have specified in [18]) needs another matrix exponentiation and three integer exponentiations to enable the use of the vote validity technique proposed for MP3 and MP1a. Therefore, we use

$\langle MP3\ voteValidity\ time\rangle + \langle matrix\ exponentiationtime\rangle$ as a lower estimative for the MP2 $voteValidity$ time.

As can be easily seen in tables 2 and 3, the use of any non native cryptographic function comes at a huge cost. In the best scenario MP2 only presents a 30% improvement over MP1 and is 6 times worst than MP3.

Note that the results are for one invocation of the $\mathcal{VE}_{pk}$ function, i.e. the results are for each individual candidate vote encryption. Thus, the values presented must be multiplied by the number of candidates to give the full vote encryption time. In practice, a full 10 candidate ballot encryption with proofs takes on the JavaCard 8.45 minutes, 15 hours or 1.47 minutes using respectively MP1a, MP2 and MP3. The results for the MULTOS card, with the same setup, are respectively 5.3 minutes, 30 minutes and 43 seconds. With a more standard parameters setup ($|p| = 1024$ and $|q| = 160$), the results for the MULTOS card are respectively 4 minutes, 2.8 minutes and 28 seconds.

6 Conclusions

This paper presented the MP3 specification, which is more efficient and simpler than all other previous MP solutions. Moreover, our implementation tests show that, in low computational power devices, e.g. smart cards or secure elements inside a mobile phone (usually also smart cards), only MP3 presents an acceptable performance. The MP3 performance improvements are mainly due to its simple mathematical structure. That simple structure comes with a small price: the MP3 soundness is $1 - 2^{1-\alpha}$ vs the $1 - 2^{-\alpha}$ soundness of the previous MP solutions.

References

1. Adida, B.: Advances in Cryptographic Voting Systems. Ph.D. thesis, MIT (August 2006)
2. Adida, B.: Helios: Web-based open-audit voting. In: 17th USENIX Security Symposium (2008)
3. Adida, B., Neff, A.: Ballot casting assurance. In: EVT 2006. USENIX/ACCURATE, Vancouver, B.C. (2006)
4. Adida, B., Neff, A.: Efficient receipt-free ballot casting resistant to covert channels. In: EVT/WOTE 2009. USENIX/ACCURATE/IAVOSS, Montreal, Canada (August 2009)
5. Benaloh, J.: Simple verifiable elections. In: EVT 2006. USENIX/ACCURATE, Vancouver, B.C. (2006)
6. Benaloh, J.: Ballot casting assurance via voter-initiated poll station auditing. In: EVT 2007. USENIX/ACCURATE, Boston, MA (2007)
7. Benaloh, J.C.: Verifiable Secret-Ballot Elections. Ph.D. thesis, Yale University (1987)
8. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM 24(2), 84–88 (1981)
9. Chaum, D.: Secret-ballot receipts: True voter-verifiable election. IEEE Security & Privacy 02(1), 38–47 (2004)

10. Chaum, D.: Punchscan (September 2009), http://www.punchscan.org/
11. Chaum, D., Pedersen, T.P.: Wallet Databases with Observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
12. Chaum, D., Ryan, P.Y.A., Schneider, S.: A Practical Voter-Verifiable Election Scheme. In: De Capitani di Vimercati, S., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 118–139. Springer, Heidelberg (2005)
13. Clarkson, M., Chong, S., Myers, A.: Civitas: Toward a secure voting system. In: IEEE Symposium on Security and Privacy, pp. 354–368 (May 2008)
14. Cramer, R., Gennaro, R., Schoenmakers, B.: A Secure and Optimally Efficient Multi-authority Election Scheme. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 103–118. Springer, Heidelberg (1997)
15. ElGamal, T.: A public-key cryptosystem and signature scheme based on discrete logarithms. IEEE Transactions on Information Theory IT-31(4), 469–472 (1985)
16. Fujioka, A., Okamoto, T., Ohta, K.: A Practical Secret Voting Scheme for Large Scale Elections. In: Seberry, J., Zheng, Y. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 244–251. Springer, Heidelberg (1993)
17. Hirt, M., Sako, K.: Efficient Receipt-Free Voting Based on Homomorphic Encryption. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 539–556. Springer, Heidelberg (2000)
18. Joaquim, R., Ribeiro, C.: An efficient and highly sound voter verification technique and its implementation - extended version. Tech. Rep. 40/2011, INESC-ID (September 2011)
19. Joaquim, R., Ribeiro, C., Ferreira, P.: VeryVote: A Voter Verifiable Code Voting System. In: Ryan, P.Y.A., Schoenmakers, B. (eds.) VOTE-ID 2009. LNCS, vol. 5767, pp. 106–121. Springer, Heidelberg (2009)
20. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: WPES, Alexandria, Virginia, USA, pp. 61–70 (November 2005)
21. Moran, T., Naor, M.: Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 373–392. Springer, Heidelberg (2006), http://www.seas.harvard.edu/~talm/papers/MN06-voting.pdf
22. Neff, C.A.: Practical high certainty intent verification for encrypted votes (2004), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.134.1006&rep=rep1&type=pdf
23. NIST: Gigital signature standard (dss) (June 2009), http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf, FIPS 186-3
24. Okamoto, T.: Receipt-free Electronic Voting Schemes for Large Scale Elections. In: Christianson, B., Crispo, B., Lomas, M., Roe, M. (eds.) Security Protocols 1997. LNCS, vol. 1361, pp. 25–35. Springer, Heidelberg (1998)
25. Sandler, D., Derr, K., Wallach, D.S.: Votebox: A tamper-evident verifiable electronic voting system. In: 16th USENIX Security Symposium (2007)
26. Sterckx, M., Gierlichs, B., Preneel, B., Verbauwhede, I.: Efficient implementation of anonymous credentials on java card smart cards. In: 1st IEEE International Workshop on Information Forensics and Security, pp. 106–110 (2009)
27. Wikipedia: (April 2011), http://en.wikipedia.org/wiki/Exponentiation_by_squaring#Computation_by_powers_of_2