An Efficient Agent e-Voting System with Distributed Trust

Aneta Zwierko[a,1] Zbigniew Kotulski[a,b,2]

[a] *Institute of Telecommunication, Warsaw University of Technology, Warsaw, Poland*

[b] *Institute of Fundamental Technological Research, Polish Academy of Sciences, Warsaw, Poland*

Abstract

A new agent-based scheme for secure electronic voting is proposed in the paper. It is universal and can be realized in a network of stationary and mobile electronic devices. The proposed mechanism makes possible to implement a user interface simulating traditional election cards, semi-mechanical voting devices or utilize purely electronic voting booths. The security mechanisms applied in the system are based on the verified cryptographic primitives: the secure shared secret scheme and Merkle's puzzles. Due to pre-computations at the stage of agents' generation, the voter must do almost no computations. The proposed distributed trust architecture makes the crucial stage of sending votes elastic, reliable, and effective.

Keywords: electronic elections, secret sharing scheme, Merkle's puzzles, mix-nets, mobile agent security, distributed trust

1 Introduction

During the recent development of all forms of e-life, like e-commerce, e-democracy or e-government, the e-voting is an area of the permament research. Lately, we could observe that the time of classical voting systems, based on paper-cards and ID is coming to the end. Not only the mechanical voting system can make the results of elections questionable (e.g., USA 2000 presidential election) but problems can also arise from methods used to gather results by central authority or errors during the counting made by people. The need for electronic voting systems is growing, some prototypes are tested within different countries.

The analysis of such systems offered by different vendors in US is presented in [20]. Most of the commercial systems offers security through obscurity, what is widely believed to be the worst possible method of protection. The systems utilize cryptography, but often in an incorrect way, leaving back-doors for intruders. On the other hand, there exist quite a few cryptographic schemes which fulfill wide

[1] Email: azwierko@tele.pw.edu.pl

[2] Email: zkotulsk@ippt.gov.pl, zkotulsk@tele.pw.edu.pl

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

requirements for electronic elections. Their only disadvantage is inconvenience: they use sophisticated cryptographic tools that make them hard to implement and require expertise in various fields. In this paper we propose a practical electronic elections scheme, that is quite easy to implement, secure, based on well-known cryptographic primitives, and on the contrary to most e-voting protocols, does not expect a voter to do any computations.

Our system fulfills the requirements stated below and due to its efficiency, simplicity and lack of computations on voter's side can be used in different scenarios: with voters using a computer for voting or with classical voting booths. It can be also used in semi-mechanical voting systems.

The requirements for electronic elections protocol differ very much: from the most obvious ones, as *privacy*, to mores sophisticated as a *receipt-freeness*. Most important ones are discussed below ([5], [25]). Thus, *completeness* requires that all valid votes must be counted correctly, *soundness* provides that dishonest voter cannot disrupt voting process, *privacy* means that all ballots must be secret and there should be no possibility of tracing the voter that cast the certain vote, *unreusability* does not permit any voter to cast ballot more than once, *eligibility* simply means that only those who are allowed to vote can vote and the system have to provide means to validate a voter and a permitted number of votes, *verifiability* prevents falsification of the result of the voting process, a voter should be able to verify if his vote was correctly accounted. There are two kinds of verifiability: *individual verifiability*, when only the voter can verify the results ([26]) and *universal verifiability*, when everyone can verify that all votes were correctly accounted (in this case some publication of votes is necessary). *Fairness* provides that nothing can effect the voting and no party should be able to compute the partial tally. *Robustness* means that all security requirements are fully satisfied despite failure and/or malicious behavior by any (reasonably sized) coalition of parties (voters, authorities, outsiders). *Receipt-freeness* claims that the voter is not able to prove any coercer how he had voted, this notion is similar to untraceability or privacy and widen their meaning.

It is seen that some of mentioned features are contradictory to others, like *receipt-freeness* and *verifiability*. It is hard to create a system or a protocol fulfilling all requirements, especially unconditionally.

The paper [27] describes also some other, additional requirements for the electronic voting system: *dispute-freeness* (a voting scheme should provide a method of resolving all disputes at any stage of voting) and *accuracy* (a voting scheme must be error-free).

These requirements typically are a part of a *verifiability* postulate. Similar to the notion *receipt-freeness* the idea of *incoercibility* was introduced: no party should coerce the voters. Some of those presented requirements are complementary but there is no defined set of criteria that can be used to fully describe and analyze an electronic voting system. The recent work of Chaum [9] notices lack of an important property in most evoting systems: *voter-verifiability*. While trying to provide receipt-freeness and incoercibility, some systems does not offer the user any confirmation that the ballot was received and tallied correctly if the proofs for the vote are not published immediately. For a large-scale elections publishing

proofs instantly is very unpractical. Instead, Chaum introduces a notion of *voter-verifiable* elections, where the voter receives the receipt, which is a confirmation of fact of casting a ballot and does not contain any information about the vote. From practical point of view, where users vote in the electronic booth or using some computer application, this property is very important.

2 Related work

Concerning cryptographic primitives, the e-voting systems utilize: mix-nets (encryption nets, decryption nets, DC-nets), blind signatures, homomorphic secret sharing schemes, bulletin boards, proofs (interactive and non-interactive) or homomorphic encryptions.

Mix-nets are similar to anonymous channel that can be used to anonymously distribute to users credentials needed for voting. A mix is a trusted party that randomly distributes messages to users, so any eavesdropper is unable to trace the sender or recipient of a given message. It was first proposed by Chaum [7]. mix-nets can be based on decryption or on re-encryption ([24]). *DC-nets* (dining cryptographers networks) is an alternative to an anonymous broadcast channel, proposed also by Chaum.

Blind signature was initially utilized to create the first protocols for e-cash applications. Shortly afterward it was used by Fujioka et al [14] to validate votes in an elections scheme. The idea is that an authority validates the vote not knowing its value (the vote is encrypted).

Homomorphic secret sharing scheme was first introduced in [4]. The vote is shared among n authorities and then tallied by at least t of them. Those systems have a high communication cost and are not easy to implement.

Homomorphic encryption model utilizes special features of homomorphic encryption algorithms. It defines two operations, $\oplus$ and $\otimes$, that for two proper votes v_1 and v_2 and an encryption algorithm E have the following property: $E(v_1) \otimes E(v_2) = E(v_1 \oplus v_2)$. This method was introduced in [10].

The bulletin board is a public, broadcast communication channel with memory [10]. All broadcast information is stored in the memory and any participant can read it. Voters have an write access to specific sections of the board, where they can publish their votes. Such a board can be implemented using multiple servers.

Proofs are mainly used by voters to prove the authorities the correctness of their sent votes. Proofs may be interactive (e.g., classical zero-knowledge proofs) or non-interactive and simply attached to the vote. They are used mainly in systems with homomorphic encryptions.

To present complete survey of e-voting schemes, we start from the Chaum [7]. The scheme is an example of the *mix-net* model and consists of at least two trusted parties: TA, the trusted administrator and the *mix*. The TA creates a set of cryptograms for all users. Voters obtain their cryptograms from the mix, which has to know who is eligible for voting. Afterward, voters prepare their votes utilizing the public key from the cryptogram: $E_K(q, v)$, where q is a random number and v is a vote. Along with the data previously received from the mix, the new cryptogram $E(r, \pi, E_K(q, v))$ is sent to the mix. The mix compiles a list of pseudonyms and

cryptograms with votes to TA, which validates the π and decrypts the vote if the π is proper. A modified version of the protocol was published later in [8].

The work [26] presents another approach to e-voting protocol based on re-encryption mix-nets. All mix-nets in this system have a unique private key form El-Gamal encryption scheme. There exists a public key for an anonymous channel. The mixes produce encrypted ballots for users with proofs. They are delivered to voters by an untappable channel. During the voting stage, the voters chose their votes and send them via decryption networks. Each mix posts a proof of proper decryption. Then votes are counted. Eligibility, privacy, fairness and universal verifiability properties are satisfied. The verifiable mixnet together with the publicly accessible bulletin board, provides universal verifiability property. Receipt-freeness property is satisfied assuming one-way untappable channels, since the voter cannot prove its vote to adversary. However, usage of untappable channels makes the scheme unpractical.

The Fujioka et al protocol [14] is more convenient for large scale elections. Apart of voters, it has two parties: counter and administrator and three phases: registration, voting and summing. It assumes the existence of an anonymous channel used by the counter and voters to communicate, that each voter has a different digital signature and uses the commitment scheme to compute the ballot and usage of a blind signature scheme by the administrator. The protocol is complete, sound, fair, verifiable, privacy is achieved, along with un-reusability and eligibility. Also the *maximal fairness* is accomplished since, even if all authorities collude, they cannot compute the partial tally. However, to obtain this the voter has to take part in tallying phase, which is rather impractical and would make the scheme hardly scalable. The disadvantage of the scheme is ability of the authority to add votes for abstained users.

The election protocols based on the homomorphic encryption are described in various papers: [2], [10], [11]. In the system proposed in [10] the authorities create a pair of shared private and public keys. Utilizing El-Gamal scheme and those keys the voters can create their ballots: encrypt their votes and produce a non-interactive proof of validity, with zero-knowledge property. After checking the proofs from the voters, the coalition of honest authorities can combine all correct votes and utilize proofs to decrypt the product. In the result they obtain the exponentiated tally of votes, use it to search the tally space for a match and compute the final tally. The scheme fulfills most of requirements described in Section 1, but the form of votes and necessity of the proofs (and their complexity) impacts the scalability of the scheme. The protocol described in [12] is similar and utilizes the generalized Pallier cryptosystem. More effective method of decryption and computing the results is presented in [10].

A system utilizing the homomorphic encryption scheme was proposed in [24] and improved in [1], [15] and [23]. During the initial stage the authority publishes the shared public key (a (t, n) threshold scheme is utilized). Then, voters register and compute their votes. They post their votes on the bulletin board (here also correctness of the votes can be checked). All votes are then sent through a re-encryption mixnet (proofs are generated during this process and can also be published on bulletin board). Then the votes are verified and tally is computed. The proposed

system not only fulfills most of the requirements but also is scalable and efficient (due to use of mix-nets). It can also be modified to provide receipt-freeness.

Some other approach to electronic voting also based on the homomorphic encryptions was proposed in [2] and [18]. The system is additionally based on tokens and re-encryption nets. The work [2] improved results of [18]. The system preserves the receipt-freeness property (and incoercibility, providing the adversary does not have access to the registration phase) since a voter can generate a false token. However, the trade-off is quite high: the verifiability and scalability were the price. Also usage of anonymous broadcast channel makes the scheme impractical (since it is hard to implement).

Moreover, there exist different systems, fulfilling the criteria from Section 1 and not based on the mentioned primitives (e.g., [21]). There are approaches based rather on information-theoretical security (not computational security) or on anonymous multi-party computations.

Some systems based on more practical approaches are being currently developed or tested in Switzerland: Geneva and Neuchtel developed an Internet voting systems, which offered as an extension of postal voting. Zurich developed also mobile voting systems [13]. The other prototype was developed in Portugal [17], called REVS. It utilizes blind signatures and is based on EVOX system. The paper [17] describes solutions for failures of communication or problems with servers. Also a secure authentication mechanism for voters was added. A prototype of the system is used for surveys.

3 Cryptographic primitives

Our scheme involves two cryptographic primitives: a secure secret-sharing scheme and the Merkle's puzzles. Below we present a short description of all of them.

Secret sharing scheme with (t, m) threshold [25] distributes a secret (block of bits) among n participants in such a way that any t of them can recreate the secret., but any $t-1$ or fewer members gain no information about it. The piece held by a single participant is called a *share* or *shadow* of the secret. Secret sharing schemes are set up by a trusted authority, called a dealer, who computes all shares and distributes them to the participants via secure channels. The participants hold their shares until some of them decide to combine their shares and recreate the secret. The recovery of the secret is done by the combiner, who on behalf of the co-operating group, computes the secret. The combiner is successful only if the reconstruction group has at least t members. Our system utilizes the Asmuth-Bloom [3] secret sharing scheme. The dealer randomly chooses n prime or co-prime numbers (called public moduli): p_i: $(i = 1, \ldots, n, p_0 < p_i < \ldots < p_n)$. They are publicly known. Then, he selects at random an integer s_0, such that $p_n < s_0 < \prod_{i=1}^{t} p_i$. He computes the secret: $K_s \equiv s_0 \pmod{p_0}$ and shares: $s_i \equiv s_0 \pmod{p_i}$. One has to have at least t shares to recreate the secret. The combiner recreates the secret by solving the following system of equations:

$$s_0 \equiv s_{i_1} \pmod{p_{i_1}}$$

$$\ldots$$

$$s_0 \equiv s_{i_t} \pmod{p_{i_1}}$$

This system has an unique solution according to the Chinese Reminder Theorem.

Merkle's puzzles were introduced in [22]. The goal of this method was to enable a secure communication between two parties: **A** and **B**, over an insecure channel. The assumptions were that the communication channel can be eavesdropped (by any third party, called **E**). Assuming that **A** selected an encryption function (F). F is kept by **A** in secret. **A** and **B** agree on a 2^{nd} public encryption function, called G. **A** will now create the N puzzles (denoted as p_i, $0 \leq i \leq N$) in the following fashion: $p_i = G((R, X_i, F(X_i)), Y_i)$, where R is simply a publicly known constant term, which remains the same for all messages. The X_i are selected by **A** at random. The Y_i are the "puzzle" part, and are also selected at random from the range $(N \cdot (i-1), N \cdot i)$. Guessing Y_i allows **B** to recover the message within the puzzle: the triple $(R, X_i, F(X_i))$. Now, he can transmit X_i in the clear and $F(X_i)$ can then be used as the encryption key in further communications. **E** cannot determine $F(X_i)$ because **E** does not know F, and so the value of X_i tells **E** nothing. **E**'s only recourse is to solve all the N puzzles until he encounters the unique puzzle that **B** has solved. So, for **B** it is easy to solve one chosen puzzle, but for **E** it is computationally hard to solve all N puzzles.

In our paper the as the G function is utilized any symmetric cipher (e.g. *AES*). To solve such a puzzle, the key has be guessed (so the key is Y_i from the previous decryption). The key for the puzzle should be weak. We are not using the X_i and $F(X_i)$ mechanism the same way as presented in the original Merkle's paper. We take advantage of fact that having a set puzzles it is hard for the adversary to solve them all in a reasonable time (if the number of puzzles is high enough), which can be useful in providing anonymity. Also, we utilize X_i and $F(X_i)$ not in bilateral communication but in more complex way.

4 Authentication with revocable anonymity

The protocol that is a basis for the proposed architecture was described in [30]. It was created to fulfill a need for a sensible trade-off between a user's need of privacy and the legal requirements for service providers. It utilized the observation that one of the most important users of all networks, including the Internet are companies and organizations. From the networking point of view, they are built of many single users that trust some authority, use mostly the same authentication method (in context of the service) and are somehow managed. Sometimes the trust relationship between the companies cannot be complete: the service provider cannot be fully trusted. The companies wish to preserve some information and protect them even from the service provider. The ability to identity each user would be probably the most important, e.g., for control reasons. Still the service provider should be able to link each action with each user. These requirements are contradictory, but can be balanced. This protocol enables an organization or an authority to identify each user basing on data collected by the service provider. It provides users (within an organization) with the anonymity (called partial anonymity, because the user still can be identified) also enabling the service providers, when needed, to trace the user with a help of his/her organization.

The protocol consists of at least five parties: TTP – the trusted third party, O – organization, TA_O is a trusted authority within the organization, SP – a service provider, and u – user, a member of O.

Initialization phase. First, the TTP creates the shares and the secret: K_s and $s_1, \ldots, s_m$. Then, it creates and enumerates (x_i) all possible subsets of the SP's shares and the O's shares that can be used to recreate a valid secret.

Table 1
Subsets of shares generated by TTP

No.	SP's shares	O's share
x_1:	$s_0, \ldots, s_{t-1}$	s_t
	...	
x_{m-t}:	$s_0, \ldots, s_{t-1}$	s_m

For each subset the TTP creates two puzzles, one for the O and other for the SP. For O: $E_k(\{R, x_i, s_{t+i}, k_i\})$ (encrypted with a random key) and for SP: $E_{k_i}(\{R, x_i, s_1, \ldots, s_{t-1}, k_j\})$ (encrypted with a key from an appropriate O's puzzle). For each generated subset the TTP stores $\{x_i, K_s, k_i, k_j\}$ (optionally also shares). After creating all puzzles can be send via an open channel to the SP and the O. The TA_O creates tickets for a given period (consisting timestamp, validity period and a share) and new number identifying the puzzle (z_i): $E_K(\{R, z_i, T_i, k_i\})$. The TA_O sends $E_{k_i}(z_i)$ to the TTP for each created puzzle with a ticket.

Authentication phase. When a user wishes to authenticate itself to the SP. He is provided by the TA_O with a set of puzzles: $E_k(\{R, z_i, T_i, k_i\})$. The user sends to the SP whole set. The SP chooses one puzzle and "solves" it extracting z_i, $T_i = \{s_i, t_i, p_i\}$ and k_i. The SP checks if the t_i and p_i are valid. The SP uses the key k_i to decipher all his puzzles - if the k_i is proper, one of the SP puzzles should be encrypted with it. The amount of the time needed for this is almost the same as for "solving" one puzzle. The SP extracts from his puzzle the k_j and x_i and combine all shares to recreate the secret (k). To validate the secret the SP encrypts the pair z_i and K_s encrypted with the key k_j (from his puzzle) and sends to the TTP, along with the x_i. The TTP uses x_i to find the appropriate key k_j, decrypt the z_i and K_s. The TTP checks if the extracted k is exactly the same as one stored for the x_i. It also uses corresponding k_i to check if the z_i can be obtained from encrypted values sent by the TA_O. If checks are successful, the the TTP answer sends to the SP, encrypted with the key k_j. The SP can then send id of puzzle, z_i, to the user and rest of the communication can be encrypted by the key k_i.

The formal description of the protocol is done in common syntax based on [6] and examples from [28]. It is presented in Fig. 1.

5 The proposed scheme

Utilizing the protocol described in Section 4, our system has at least four parties: TA – trusted authority, *mix*, *counter* and voters.

The trusted authority (TA) is responsible for creating a list of registered users

Fig. 1. The formal description of the protocol

Step	Initial phase	Authentication phase
1	TTP → O: {R, xi, si, ki}k	O → U: {R, zi, ti, ki}k
2	TTP → SP: {R, xi, s1, ..., s(t-1), kj}k	U → SP: {R, zi, ti, ki}k
3	O → TTP: {zi}ki	SP → TTP: {u, zi}kj
4		TTP → SP: {Yes/No}kj
5		SP → U: {Yes/No}ki

(ones that are allowed to vote) and authentication data that enables them later to vote. It is similar to a manager in an agent system.

The *mix* distributes the data required for voting to all voters, checking if they are registered. The *mix* is similar to organization, or rather TA_O in the original scheme. Its main goal is to protect the voters' privacy. It is realized as an agent, rather stationary then mobile, residing on a specific host and denoted as A_M.

The *counter* collects the votes and validates the voters' credentials with TA. He also tallies the votes and publishes them for verification. The *counter*, denoted as A_C, is also a kind of a stationary agent.

The voter's application can be also an agent of different type: an application in a cell phone, an mobile agent that user will sent to the host with *mix* agent or *counter*, or an agent in the electronic booth (it is denoted as A_V).

The basic steps of the election are:

(i) TA creates the set of credentials and the list of registered users and sends it to the A_M.

(ii) For each voter requesting credentials, the A_M creates ones for data received from TA. First, it checks if a user has the right to vote.

(iii) User sends his vote along with credentials to the A_C: *counter* checks the credentials with TA and, if they are proper, the A_C sums up the votes, publishing the credentials.

Assumptions: TA is trusted to create valid credentials for voters and validate them properly during the voting. The *mix* is trusted by voters not to link them to credentials created by TA. The *counter* is trusted to accept votes received from the voters, providing their credentials are correct, and to publish proper hashes for verification. The *mix* is trusted by the voters to hide the link between their identity and credentials, which they have obtained. All three parties are independent and are trusted not to cooperate (at least, the *mix* and TA).

5.1 The detailed scheme

The architecture of the proposed solution is presented in Fig. 2. The scheme has four major phases: initialization, registration, voting and publication of results.

The following notation is used in the protocol:

- secret symmetric keys: k, k_i^C and k_i^M are secret keys for a symmetric cipher,

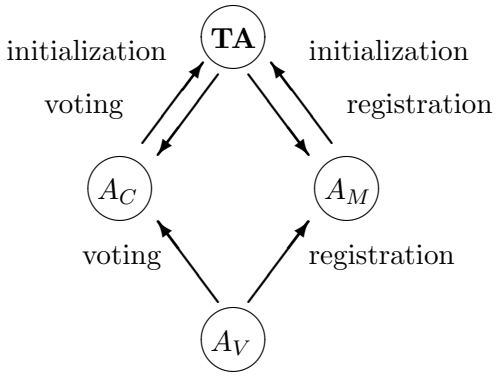

Fig. 2. The architecture of the proposed system

created by TA,

- functions: E (encryption), h (hash function) and g (described later),
- votes: $V = \{v_0, \cdots, v_L\}$ is a set of all possible votes (e.g., v_0 can be *yes*, v_1 can be *no*, or v_0 can a first candidate, etc), v_f denotes a user's selected vote, L denotes a number of different possible votes,
- ballots: $B = \{b_0, \cdots, b_L\}$ is a set of ballots (utilized by the user to vote),
- secret sharing scheme values: s denotes a shares (s_i denotes the i_{th} share), K_s is the secret, t denotes the threshold and n number of shares,
- other: x (is an identifier), R denotes redundancy bits for Merkle's puzzles (as described in Section 3).

Initialization. The authentication mechanism is based on the secure secret sharing scheme (described in Section 3). The TA creates the system with n shares and threshold t. The $t-1$ shares are for A_C, the rest will be distributed between the users by the *mix*. For each secret, the TA creates an identification x for all $n-t$ subsets of shares that enable to recreate the secret. The value of x for each subset is created in a random way. They are presented in the Table 2.

Table 2
Subsets of shares, generated by TA

No.	*counter's* shares	*mix's* share
x_1:	$s_0, \ldots, s_{t-1}$	s_t
x_2:	$s_0, \ldots, s_{t-1}$	s_{t+1}
	...	
x_{n-t}:	$s_0, \ldots, s_{t-1}$	s_n

For each subset of shares (identified by the x_i) the TA creates a Merkle's puzzle for A_M $p_M = E_k(\{R, x_i, s_{t+i}, k_i^M\})$ and a cryptogram for A_C: $d_C = E_{k_i^M}(\{R, x_i, s_1, \ldots, s_{t-1}, k_i^C\})$. The p_M puzzle is encrypted with a random key (k) and sent to the A_M. The d_C cryptogram is encrypted with a key from an appropriate *mix's* puzzle (k_i^M) and destined for the A_C. For each generated subset the TA stores $\{x_i, K_s, k_i^M, k_i^C\}$ (optionally also shares). For a single secret, the TA creates

$n - t$ puzzles for the *mix* and the same number of cryptograms for the *counter*. After creation, all data can be sent via an open channel. The TA has to create at least N_u puzzles in total (one for each voter). Assuming that all utilized secure sharing secret systems have parameters (n, t), the TA has to create at least $\frac{N_u}{n-t}$ secrets. This part of the protocol is illustrated in Fig. 3(a).

Registration. The voter registers within the A_M to obtain the credentials. The voters should be validated if they are eligible for elections. There are many different methods that can be used for this purpose, and it is out of this paper's scope to choose the best one. A simple one would be based directly on a Guillou-Quisquater identification scheme [16]. Each voter obtains from TA its ID and a secret value σ. The TA creates a list of all proper ID's and sends it to the *mix* agent as a list of registered voters. The voter utilizes the GQ protocol to prove knowledge of secret for selected ID.

After validating the user, the *mix* solves a randomly selected puzzle received from the TA and obtains authentication data for user: a share s_i, a key for appropriate A_C puzzle k_i^M and the identifier of subset x_i. This data is used to create set of ballots (B) for the voter. Each ballot is encrypted with weak, random key (similar to puzzles). The ballot consists of vote (v) encrypted with x_i, the share and the key from the solved puzzle and a proof. The proof is a hash of vote and $g(x_i)$: $h(v, g(x_i))$. The function g is known only to the A_M. Its main feature is that it is hard to guess its result not knowing the argument. It can be e.g. a hash function used with additional secret string of bytes or encryption with a secret key. Note that the hash function used with a secret argument have the same functionality as a signature, but is much more efficient. To enable TA verifying the proof, the *mix* sends $E_{k_i^M}(g(x_i))$ to the TA or publishes the list of all created $E_{k_i^M}(g(x_i))$ after registration. This part of the protocol is illustrated in Fig. 3(b). The final set of ballots for a single user is presented below.

vote	ballot
v_1:	$b_1 = E_k(R, k_i^M, s_i, E_{x_i}(v_1), h(g(x_i), v_1))$
...	...
v_L:	$b_L = E_k(R, k_i^M, s_i, E_{x_i}(v_l), h(g(x_i), v_L))$

Voting. The voter sends the ballot with a chosen vote v_f to the A_C: $b_f = E_k(R, k_i^M, s_i, E_{x_i}(v_f), h(g(x_i), v_f))$. Counter breaks the puzzle and extracts s_i and k_i^M. He uses the key k_i^M to decipher all his puzzles: if the k_i^M is proper, one of the puzzles should be encrypted with it. The A_C extracts from his puzzle the k_i^C, x_i and shares. Next, he combines all shares to recreate the secret (K_s). To validate the secret the *counter* encrypts K_s with the key k_i^C (from his puzzle) and sends it, along with x_i, to the TA. The TA uses x_i to find the appropriate key k_i^C, and decrypts the secret. The TA compares the extracted K_s with one stored in the database. It also uses corresponding k_i^M to mark the $g(x_i)$ published by the *mix* agent. If both operations are successful, the TA sends answer to the A_C, encrypted with the key k_i^C. The *counter* now decrypts the vote, adds it to the tally and sends appropriate information to the voter. The *counter* can use the key k_i^M to

encrypt the information about success or failure of the operation or send the result in clear-text. This part of the protocol is illustrated in Fig. 3(c). Note, that the protocol does not require user to make any computations, only to communicate with appropriate parties, as in classical elections. The A_C can also convey the x_i to the voter as a confirmation of fact of casting the ballot (fulfilling the *voter-verifiability* property).

Publication of results. After receiving and verifying all votes the *counter* agent publishes the results along with all proofs so users can verify if their votes have been counted (Fig. 3(d)). In case of any claims about correctness of the results the TA can verify all pairs of votes and proofs utilizing the $g(x_i)$ values.

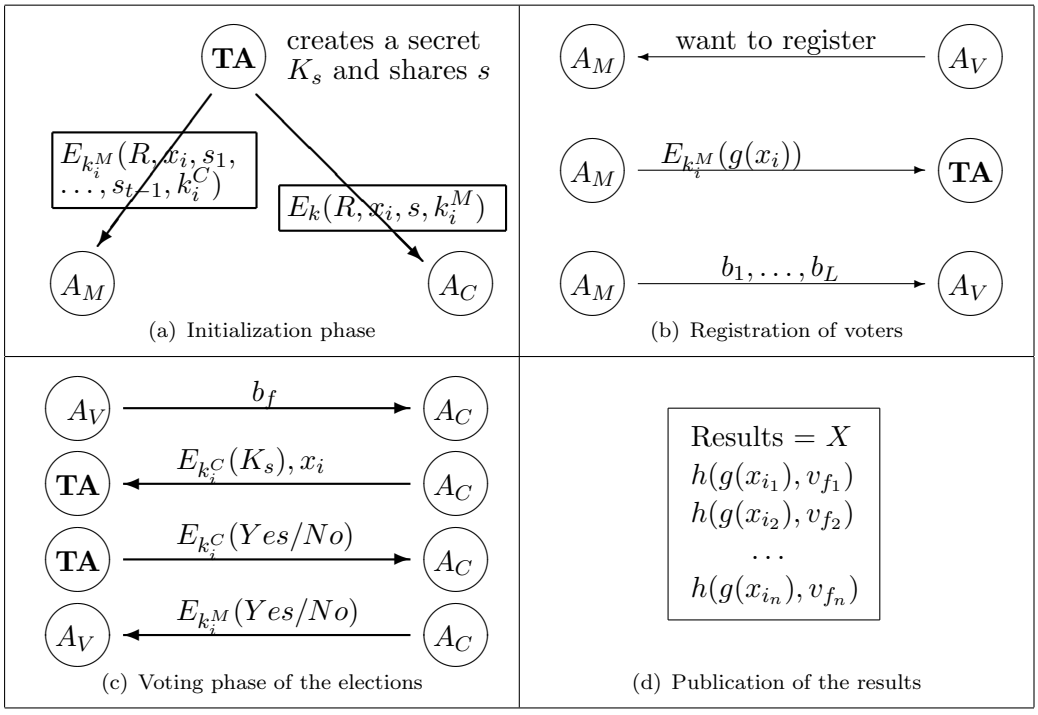

Fig. 3. Phases of the protocol

5.2 Formal specification

The notation used in this specification is based on one used for SVO logic [29]. An encrypted message with key k is denoted as $\{m\}_k$.

Initialization

(i) $TA \rightarrow A_M$: $\forall_{s \in \{s_t, \ldots s_n\}} \} R, x_i, s, k_i^M \}_k$

(ii) $TA \rightarrow A_C$: $\{R, x_i, s_1, \ldots, s_(t-1), k_i^C\}_{k_i^M}$

Registration

(i) $A_M \rightarrow TA$: $\{g(x_i)\} k_i^M$

(ii) $A_M \rightarrow A_V$: $\forall_{v \in V} \{R, k_i^M, s_i, \{v\}_{x_i}, \{h(g(x_i), v)\}_k$

Voting

(i) $A_V \rightarrow A_C$: $\{R, k_i^M, \{v_f\}_{x_i}, h(g(x_i), v_f)\}_k$

(ii) $A_C \rightarrow TA$: $\{K_s\}_{k_i^C}$

(iii) $TA \rightarrow A_C$: $\{Yes/No\}_{k_i^C}$

(iv) $A_C \rightarrow A_V$: $\{Yes/No\}_{k_i^M}$

6 Analysis

6.1 Performance

All presented results were obtained with the test version of the system, working on 3.0GHz PC with Windows operating system. The implementation is based on *JADE* agents.

Initialization. In this phase only the *TA* has to make computations. It has to create the secure secret sharing scheme, generate the identifier, symmetric keys, create subsets, puzzles and cryptograms for A_M and A_C. The time to create shares and the secret for system with $n = 100$ and $t = 75$ with size of public moduli 100 bits is just around 10 minutes. Generating even 100 such sets (of secret and shares) is not time consuming, especially, that it can be done offline. Creating the subsets, keys and identifier are fast and omitable in further analysis. Computing the puzzles and cryptograms is simply comparable to encrypting a data (D_{total}) of size equal to sum of all cryptograms and puzzles. The size of a counter's cryptogram is equal to sum of its elements: $D_{C_{size}} = (t-1) \cdot s_{size} + R_{size} + x_{size} + k^C_{size}$. Similarly, size of A_M puzzle can be calculated: $P_{M_{size}} = s_{size} + R_{size} + x_{size} + k^M_{size}$. For each secret TA has to encrypt $n - t$ puzzles for A_M and cryptograms for A_C: $D_{total} = (n-t) \cdot (P_{M_{size}} + D_{C_{size}})$. The encryption time of symmetric cipher amplifies almost linearly with the size of data (that is an expected result), so having size of public moduli (for secure secret sharing scheme) and number of shares it is possible to estimate the time required by the TA to generate puzzles for each secret.

Registration phase. During the registration phase only the *mix* has to make some computations: solve a puzzle for the TA and create the ballots for users. Solving a single puzzle is not computationally hard,. As the tests indicate the time to solve a single puzzle is negligible, depending entirely on the key size. Note, that the actual size of puzzle is not important since only the first block of the puzzle has to be decrypted (to validate R). The mix can either solve the puzzle on-line, while the user is registering or solve all puzzles offline. Both options are feasible, the only concern could be a number of users that may want to register at the same time. That problem can be easily solved by adding additional mix agent A_M. The *mix* also has to create a proof for each ballot: $h(g(x_i), v_l)$. We assume, that g is a efficient function (the best candidates are symmetric ciphers or hash function), so the time to create $g(x_i)$ is negligible. As discussed previously, the number of votes, L, is relatively small and computing the hash function is efficient. So we can assume that the total time required by the *mix* to create the proofs is equal to $N_U \cdot L \cdot T_{proof}$, where T_{proof} is time required to generate the $g(x_i)$ and the actual hash (can be approximated as the double time to generate the hash). To estimate the time required to create puzzles for voters the similar reasoning as for initial phase can be used. The mix has to create L ballots for each user, each one of size: $b_{size} = s_{size} + R_{size} + k^M_{i_{size}} + E_{size}(v) + h_{size}$. It is easy to calculate the total

size of set of ballots for a single voter is rather small, since for the typical cipher and one-way hash function $b_{size} < 1Kb$. Creating puzzles for such small amount of data is very fast. So, even for a large number of users the whole operation is very efficient.

Voting. During the voting phase there is not any operation that is either time consuming or requiring a large number of computations. First, the *counter* needs to solve the puzzle, which is efficient (as discussed previously). Next, A_C has to find its own suitable cryptogram: at the average it needs to decrypt half of its cryptograms. The operation of decryption is similar to encryption in the means of time and computations, so the required time can be easily estimated. The third operation of the *counter* is combining the secret. It requires to solve a set of equations, using a Chinese Reminder Theorem. The results of the tests show that this operation is rather fast. The TA needs just to find appropriate keys in the database, decrypt the secret and validate (simply compare it to the stored one). All those are rather fast actions. Also, the final operation of A_C is very efficient, since it is only encryption of the reply to user. The performance problems can occur if many users at the same moment would like to vote and the counter would have to find a lot of suitable cryptograms. But sensibly selected number of counters per voter can minimize the risk of delays.

6.2 Communication effort

(i) Initialization phase: the TA has to send to A_C and A_M all cryptograms and puzzles. Basing on calculations presented in the previous section, it is possible to estimate amount of data that has to be conveyed to each party. The time required to transmit the data to appropriate party with speed *2 Mbit/s* is less then 50 milliseconds for systems with n from 50 to 500 for data sent by the TA to A_M and up to 5 sec for data transmitted to the A_C.

(ii) Registration: during this phase, there is no significant communication: the *mix* only sends the $E_{k_i^M}(g(x_i))$ of size of approximately 128 bits to the TA and a set of ballots, less than 10KB to the user. Even with a limited bandwidth (e.g. *2 Mbit/s*) and large number of users (100000) system can be efficient (sending the puzzles would take approximately 45 minutes).

(iii) Voting: during this operation, the amount of transferred data between the parties is very small (around 384 bits between the TA and A_C and around 1 Kb between A_C and voter), so no communication overhead can be expected, especially that the operations during this phase are rather fast and voter should not have wait long for reply from the *counter*.

6.3 Security

Completeness. If counter receives the vote and verifies it with the *TA*, then both, the voter and the *TA* can check if the vote has been properly tallied.

The user can verify if its vote has been counted correctly by looking up the appropriate proof. A method to settle a possible dispute is discussed within *verifiability* feature.

Soundness. To produce a valid ballot, an adversary has to forge the credentials: the share s_i and the key k_i^M. He would also have to produce a valid proof for the vote $h(g(x_i), v_f)$. Since the keys, x_i and cryptogram $E_{k_i^M}(g(x_i))$ are stored in the *TA's* database, an adversary would have to guess properly all of them. Chances of success are extremely low. Even if the adversary is a dishonest voter and has access to already used proofs, ballots and even $E_{k_i^M}(g(x_i))$ list, it is still computationally hard to find for a selected proof $(h(g(x_i), v_f))$ the values that were used to create it. For the adversary it is computationally infeasible to guess proper x_i for existing $E_{k_i^M}(g(x_i))$.

A dishonest voter may try to claim that he/she voted other way than was tallied, but the *counter* can prove otherwise by publishing the proof, which is hard to forge without help of *mix* or *TA*.

Privacy. The ballots are distributed by the *mix* agent A_M and no other party is able to link a user and a ballot with his vote.

The ballots are encrypted with key x_i. Since x_i is a secret and it is not known to any third party, an eavesdropper observing the ballots is not able to gain any information on selected vote.

Un-reusability. The voter obtains a single share as a credential, identified by the x_i. He can used only one vote and when his/hers vote is verified by the *counter*, the appropriate data in the *TA's* database is marked and cannot by utilized again.

Eligibility. Only proper users, registered and listed can obtain credentials from the *mix* agent A_M. They have to obtain proper identification data and the zero-knowledge scheme provides that only users knowing the valid secret (σ) can successfully complete the protocol.

Verifiability. All users can verify if their votes have been properly tallied by looking at a published list of proofs $h(g(x_i), v_f)$. There are two kinds of possible frauds: the *counter* does not tally a casted vote or claims the vote was different then one actually casted by the voter. If the voter's proof is not published, the voter can present his chosen ballot to the *TA* and he can verify, wherever suitable data in the database was marked as used (or the confirmation stage can be added to the protocol, as described in the Section 5.1). In the second case the *TA* can verify the proof presented by the *counter* and settle the dispute. The *TA* can also estimate the minimal and maximal number of voters taking part in the elections basing on amount of $E_{k_i^M}(g(x_i))$ values created by the A_M and number of successful interaction with the A_C.

Receipt-freeness. The scheme enables the voter to verify if his vote was tallied by publishing the proofs $(h(g(x_i), v_f))$, but does not enable him to prove to any third party what vote was casted. The voter can present the coercer a ballot $b_f = E_k(R, k_i^M, s_i, E_{x_i}(v_f), h(g(x_i), v_f))$ for the published hash. The coercer can verify that the hash is published, but he is unable to verify the claimed vote, since it is encrypted with x_i. More, it is easy for the voter to create a false ballot with selected vote and published hash, since the coercer cannot verify the proof $h(g(x_i), v_f)$, because the g is secret and known only to the *mix* and the list of produced $g(x_i)$ values is known only to the TA.

Robustness. The non-participating voter does not influence of course of elections. All appropriate authorities (*counter*, *mix* and *TA*) have to take part during the

selected stages of elections. The single authority is not able itself to change the results of elections, all three have to collaborate to falsify the proof and/or ballot or to identify the user who casted a selected vote.

Scalability. There are multiple ways to enlarge the application based on the agents. One of methods is to create hierarchical structure, based on triples of agents: A_M, A_C and A_{TA} (an agent representing the TA). Each triple will be responsible for registering users in certain region and tallying. Also, a single secret can be utilized to authenticate more than one user and the system can utilize multiple *mix* agents (with different or the same sets of puzzles) and *counter* agents.

7 Possible modifications

Registration phase (*mix's* actions) can be divided into two independent subphases: the first one for receiving authentication data from the TA and creating ballots and the second one for distributing ballots to users. These two phases can be additionally distributed between two different kinds of agents (denoted as A_M^1 and A_M^2), that would provide system with even higher degree of privacy due to distributed trust. The modified architecture is illustrated in Fig. 4.

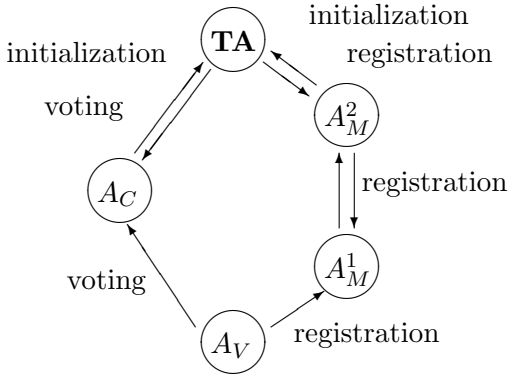

Fig. 4. Modified architecture with more distributed trust

The ballot for our scheme is practical and easy to create. But also other cryptographic primitives along with presented credentials can be used, e.g.: blind signatures (the user can ask TA or the *mix* agent to sign the vote), or a public key, distributed with credentials, unique for each x_i, that is later used to encrypt the vote.

All those methods can be viewed as more secure for typical for e-voting system than the simple hash, but they require some computations on users side, which is a disadvantage, especially for mobile devices or cards.

8 Conclusions

The presented system is an efficient and practical scheme for electronic elections. It utilizes the well-known, secure cryptographic primitives to achieve privacy and anonymity by distributing trust. It is complete, sound and very scalable due to usage an agent-based architecture. The system offers possibility of easy extension,

simply by adding an additional *mix* agent or *counter* agent, when required. One of the main advantages of the proposed scheme is avoiding users' computations. Therefore it is very flexible and easy to use for all kinds of elections. A user can vote in a traditional way (the votes can be printed) or in electronic booths. The system also provides a user with mobility: the user can have an agent program in his mobile device that will send his chosen ballot to the *counter* agent: the user just has to be in a range of any wireless network (GSM/3G/802.11). The system can operate with many existing technologies to transmit votes and still maintain security due to utilized cryptographic solutions. A prototype of the *EVAS:* E-Voting Agent System, based on mobile agents and utilizing UMTS is currently under development.

References

[1] Abe, M. "Universally verifiable mix-net with verification work independent of the number of mix-servers". *IEICE Transactions on Fundamentals*, E83-A(7). pp. 1431-40, 2000.

[2] Acquisti, A. "Receipt-free homomorphic elections and write-in ballots". Cryptology ePrint Archive, Report 2004/105, http://eprint.iacr.org/ 2004.

[3] Asmuth, C., Bloom, J. "A modular approach to key safeguarding". In: *IEEE Transactions on Information Theory*, IT-29(2), pp.208 – 211, 1983.

[4] Benaloh, J. "Verifiable Secret-Ballot Elections". Ph.D. dissertation, Yale University, YALEU/CDS/TR-561, Dec 1987.

[5] Burmester, M., Magkos, E. "Towards Secure and Practical e-Elections in the New Era". In: "Secure Electronic Voting", *Advances in Information Security*, Vol. 7, Gritzalis, Dimitris (Ed.), Springer-Verlag 2003.

[6] Burrows, M., Abadi, M., Needham., R. "A logic of authentication". Technical Report 39, Digital Systems Research Center, February 1989.

[7] Chaum, D. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". *Communications of the ACM*, 24(2), pp. 84-88, 1981.

[8] Chaum., D. "Elections with unconditionally secret ballots and distruption equivalent to breaking RSA". In: *Advances in Cryptology - EUROCRYPT'88*, LNCS 330, pp. 177 - 182, Springer-Verlag, 1988.

[9] Chaum., D. "Secret-Ballot Receipts: True Voter-Verifiable Elections". In: *IEEE Security and Privacy*, vol. 02, pp. 38 – 47, IEEE, 2004.

[10] Cramer, R., Gennaro, R., Schoenmakers, B. "A Secure and Optimal Efficient Multi-Authority Election Scheme". In: *Advances in Cryptology - EUROCRYPT'97*, LNCS 1233, Springer-Verlag, pp. 103-118, 1997.

[11] Damgard, I., Groth J., Salomonsen G. "The Theory and Implementation of an Electronic Voting System".In: *Secure Electronic Voting*, D. Gritzalis (Ed.), pp. 77 – 100, Kluwer Academic Publishers, 2003.

[12] Damgard, I., Jurik, M. "A generalization, a simplification and some applications of Paillier probabilistic public-key system". In: *Proceedings of public key cryptography, PKC 2001*, LNCS 1992, pp. 119-36. Springer-Verlag, 2002.

[13] eVoting – The Geneva Internet voting system, http://www.geneve.ch/evoting/english/presentation_projet.asp.

[14] Fujioka, A., Okamoto, T., Ohta, K. "A practical secret voting scheme for large scale elections". In: *Advances in Cryptology - AUSCRYPT 92*, LNCS 718, pp. 24859, Springer-Verlag. 1993.

[15] Golle, P., Zhong, S., Boneh, D., Jakobsson, M., Juels, A. "Optimistic mixing for exit-polls". In: *Advances in Cryptology - ASIACRYPT 02*, LNCS 2501, pp. 451-465, Springer-Verlag, 2002.

[16] Guillou, L.C., Quisquater, J-J. "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory". In: *Advances in Cryptology – EUROCRYPT'88*, LNCS 330, pp. 123 - 128, Springer-Verlag, 1988.

[17] Joaquim, R., Zquete, A., Ferreira, P. "REVS - A Robust Electronic Voting System". *IADIS International Journal of WWW/Internet*, Vol. 1(2), 2003.

[18] Juels, A., Jakobsson, M. "Coercion-resistant electronic elections". Cryptology ePrint Archive, Report 2002/165, http://eprint.iacr.org/, 2002.

[19] Knuth, D. E. "Seminumerical Algorithms", Vol. 2 of the "The Art of Computer Programming", Addison-Wesley, NY, 1989.

[20] Kohno, T., Stubblefiled, A., Rubin, A. D., Wallach, D. S. "Analysis of an Electronic Voting System". *IEEE Symposium on Security and Privacy*, 2004.

[21] Malkhi, D., Margo, O., Pavlov, E. "E-Voting Without 'Cryptography'". In: *Financial Cryptography '02*, LNCS 2357, pp. 1 – 15, Springer, 2003.

[22] Merkle, R. "Secure Communications over Insecure Channels". In: *Communications of the ACM*, pp. 294-299, April 1978.

[23] Ogata, W., Kurosawa, K., Sako, K., Takatani, K. "Fault-tolerant anonymous channel". In: *Proceedings of the ICICS 97*, LNCS 1334, pp. 440-4. 1997.

[24] Park, C., Itoh, K., Kurosawa, K. "Efficient anonymous channel and all/nothing election scheme". In: *Advances in cryptology EUROCRYPT 93*, LNCS 765, pp. 248-59, Springer-Verlag, 1994.

[25] Pieprzyk, J., Hardjono, T., Seberry, J. "Fundamentals of Computer Security", Springer, Berlin 2003.

[26] Sako, K., Killian, J. "Receipt-free mix-type voting scheme a practical solution to the implementation of a voting booth". In: *Advances in cryptology EUROCRYPT 95*, LNCS 921, pp. 393-403, Springer-Verlag, 1995.

[27] Sampigethaya, K., Poovendran, R. "A framework and taxonomy for comparison of electronic voting schemes". *Computers & Security*, vol. 25, pp. 137 - 153, Elsvier Direct, 2006.

[28] Security Protocols Open Repository, http://www.lsv.ens-cachan.fr/spore/.

[29] Syverson, P., van Oorschot, P. "On unifying some cryptographic protocols". In *IEEE Security and Privacy*, pp. 14–28, IEEE CS Press, May 1994.

[30] Zwierko, A., Kotulski, Z. "A new protocol for group authentication providing partial anonymity", 1st EuroNGI Conference on Next Generation Internet Networks - Traffic Engineering /NGI 2005 / . In: *Next Generation Internet Networks*, 2005, pp. 356 – 363, Proceedings IEEE, IEEEXplore.