Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 168 (2007) 109–126
www.elsevier.com/locate/entcs

A Light-Weight 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. The scheme is universal and can be realized in a network of stationary and mobile electronic devices. The proposed mechanism supports the implementation of a user interface simulating traditional election cards, semi-mechanical voting devices or utilization purely electronic voting booths. The security mechanisms applied in the system are based on verified cryptographic primitives: the secure secret sharing scheme and Merkle's puzzles. Due to pre-computations during the generation of agent, the voter need not to do 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, mixnets, 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, e-voting is an area of the permanent research. Lately, we observed 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 a central authority or from 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 offer security through obscurity, what is widely believed to be the worst possible method of protection. Those systems utilize cryptography, but often in an incorrect way, leaving back-doors for intruders.

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

doi:10.1016/j.entcs.2006.12.004

On the other hand, there exist quite a few cryptographic schemes which fulfill a wide range of 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 election scheme that is quite easy to implement, secure, based on well-known cryptographic primitives. On the contrary to most e-voting protocols, the scheme does not expect a voter to do any computations; all necessary computations are done by the authorities. User only needs to obtain the ballots and send a selected vote.

Our system fulfills the requirements stated below. Due to its efficiency, simplicity and lack of computations on the voter's side it 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 election protocols differ very much: from the most obvious ones, as *privacy*, to more 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 a dishonest voter cannot disrupt the voting process. *Privacy* means that all ballots must be secret and there should be no possibility of tracing a voter that cast a certain vote. *Un-reusability* does not permit any voter to cast more than one ballot. *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 and 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 tallied (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 completely 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 privacy and widen its meaning.

It is seen that some of the 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 are typically a part of the *verifiability* postulate. Similar to the notion *receipt-freeness*, the idea of *incoercibility* was introduced: no party should be able to 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] noticed the lack of an important

property in most proposed e-voting systems: *voter-verifiability*. While trying to provide *receipt-freeness* and *incoercibility*, some systems do not offer the user any confirmation that the ballot was received and tallied correctly (if proofs for votes are not published immediately). For large-scale elections publishing proofs instantly is very unpractical. Instead, Chaum introduced a notion of *voter-verifiable* elections, where the voter receives the receipt, which is a confirmation of the fact of casting a ballot and does not contain any information about the vote. From practical point of view, when users vote in the electronic booth or use some computer application, this property is very important.

Main contribution of this paper is a novel scheme for electronic election, that is secure and enables designing multi-interface, mobile voting architecture. The proposed system is based on an idea of an authentication protocol with revocable anonymity, which utilizes a combination of Merkle's puzzles and a secure secret sharing scheme. The Merkle's puzzles provide anonymity and a secure secret sharing scheme is a method of group authentication. Both methods can also be used for the e-voting scheme to protect voters' privacy and create effective method of authorization.

Organization of the paper

The Section 2 describes most important solutions for electronic election schemes. The following section introduces cryptographic primitives utilized in the proposed protocol: zero-knowledge and secure secret sharing scheme. Section 4 gives a short overview of the authentication scheme providing revocable anonymity, which was an inspiration for the new solution. Section 5 exactly describes the developed protocol. The next section contains deep analysis of the protocol, both in means of computational and communication complexity, as well as the security analysis. The last section concludes the paper and describes some possible improvements to the discussed solution.

2 Related Work

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

Mixnets are similar to anonymous channels 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]. Mixnets can be based on decryption or on re-encryption [24]. *DC-nets* (dining cryptographers networks) are an alternative to anonymous broadcast channels, 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 election scheme. The idea is that an authority validates the vote not knowing its value (the vote is blinded).

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

The 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 access to write 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 the votes they sent. 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 the complete survey of e-voting systems we start from Chaum [7]. This scheme is an example of the *mixnet* model and consists of at least two trusted parties: TA, the trusted administrator and the *mix*. TA creates a cryptogram $E(r, K, \pi)$ for each voter, where π is a pseudonym for a voter, K is a public key and r is a random number. TA sends all cryptograms to the *mix*. 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 K 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 π and decrypts the vote if π is proper. A modified version of the protocol was published later in [8].

The work [26] presents another approach to e-voting based on re-encryption mixnets. All *mixes* in this system have a unique private key for the El-Gamal encryption scheme. There exists a public key for an anonymous channel. *Mixes* produce encrypted ballots with proofs for users. They are delivered to voters by an untappable channel. During the voting stage, the voters choose 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 last property is provided by usage of the verifiable mixnet together with the publicly accessible bulletin board. The *receipt-freeness* property is satisfied assuming one-way untappable channels, since a 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 existence of an anonymous channel

used by the counter and voters to communicate, usage of a blind signature scheme by the administrator and that each voter has a different digital signature and uses the commitment scheme to compute the ballot. 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 (post-vote-casting), 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 the 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) makes the scheme non-scalable. The protocol described in [12] is similar and utilizes the generalized Pallier's cryptosystem. A more effective method of decryption and computing the result is presented in [10].

Another 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 the tally is computed. The proposed system not only fulfills most of the requirements but also is *scalable* and *efficient* (due to use of mixnets). 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., based on anonymous multi-party computations.

A distinct approach is based rather on information-theoretical security than on

computational security, as in previously discussed cases.

Some systems based on more practical approaches are being currently developed or tested in Switzerland: Geneva and Neuchatel developed an Internet voting systems, which is offered as an extension of the 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 utilized for student surveys.

3 Cryptographic Primitives

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

A secure secret sharing scheme with (t, n) 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, so that $p_0 < \ldots < p_{i-1} < p_i < \ldots < p_n$ for $i = 1, \ldots, n$. They are publicly known. Then, he selects at random an integer s_0, such that $0 < s_0 < \prod_{i=1}^{t} p_i$. 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:

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

$$\ldots$$

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

This system has a unique solution (modulo $\prod_{j=1}^{t} p_{i_j}$) according to the Chinese Remainder 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 assumption was that the communication channel can be eavesdropped (by any third party, called **E**). Assume 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 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 (called redundancy bits). X_i are selected by **A**

at random. 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 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 any symmetric cipher (e.g., *AES*) can serve as G function. To solve such a puzzle, the key has be guessed (so, the key is Y_i from the previous description). The key for the puzzle should be weak. We are not using X_i and $F(X_i)$ mechanism the same way as presented in the original Merkle's paper. We take advantage of the fact that it is hard for the adversary to solve a whole set of puzzles in a reasonable time (if the number of puzzles is high enough). Thus, this mechanism can be utilized to provide anonymity. Moreover, we utilize X_i and $F(X_i)$ not in bilateral communication but in a 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 the need for a sensible trade-off between a user's need of privacy and the legal requirements for service providers. It is based on the observation that 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. The protocol [30] allows an organization or an authority to identify each user based on data collected by the service provider. It provides users (within an organization) with anonymity (called partial or revocable 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 goal of the protocol is to provide a user and service provider with efficient method of anonymous authentication. The anonymity of the user can be revoked by the service provider with the cooperation of the organization. The protocol consists of at least five parties. TTP is the trusted third party, O is an organization, TA_O is a trusted authority within the organization, SP is a service provider, and u is a user, a member of O.

Initialization phase. First, TTP creates the secret and the shares: K_s and $s_1, \ldots, s_n$. Each, subset of generated shares, containing t elements, can be used to

restore the secret. TTP mark each subset with t shares with identifier, denoted as x_i, $1 \leq i \leq n-t-1$. From each subset $t-1$ shares are destined for SP and a single one for O. Table 1 shows possible subsets.

Table 1
Subsets of shares generated by TTP

No.	SP's shares	O's share
x_1:	$s_1, \ldots, s_t$	$s_t + 1$
	...	
x_{n-t-1}:	$s_1, \ldots, s_t$	s_n

Shares for each participant are wrapped into puzzles. TTP generates three random keys k, k_{i_1} and k_{i_2}, $1 \leq i \leq n-t-1$. The first key is weak and only utilized to create puzzle for O: $E_k(R, x_i, s_{t+i}, k_{i_1})$. The second key, k_{i_1}, is utilized to create the puzzle for SP: $E_{k_{i_1}}(R, x_i, s_1, \ldots, s_t, k_{i_2})$. The third key will be later used for secure communication between SP and TTP, during the verification of the restored secret. TTP stores in its database $\{x_i, K_s, k_{i_1}, k_{i_2}\}$.

After creation, all puzzles can be sent via an open channel to SP and O. For each puzzle TA_O creates multiple tickets T_j for a given period (consisting of a timestamp, a validity period and a share) and an identifier z_j for a new puzzle: $E_k(R, z_j, T_j, k_{i_1})$ ($1 \leq j \leq M$, where M is the required number of puzzles for users). M depends on multiple parameters, including life-time of system, number of users and desired security level. TA_O sends $E_{k_{i_1}}(z_j)$ to TTP for each created puzzle with a ticket.

Authentication phase. When a user wishes to authenticate itself to SP, he/she is provided by TA_O with a set of puzzles of a form $E_k(R, z_j, T_j, k_{i_1})$. The user sends to SP the whole set of puzzles. SP chooses one puzzle and "solves" it extracting z_j, $T_j = \{t_j, p_j, s_i\}$ and k_{i_1}. SP checks if t_j and p_j are valid. SP uses the key k_{i_1} to decipher all his puzzles: if k_{i_1} is proper, one of $SP's$ puzzles should be encrypted with it. The amount of time needed for this is almost the same as for "solving" one puzzle. SP extracts from his puzzle k_{i_2} and x_i and combines all shares to recreate the secret (K_s). To validate the secret SP encrypts the pair z_j and K_s with the key k_{i_2} (from its puzzle) and sends them to TTP, along with x_i. TTP uses x_i to find the appropriate key k_{i_2}, decrypts z_j and K_s. The TTP checks if the extracted K_s is exactly the same as the one stored for the x_i. It also uses corresponding k_{i_1} to check if z_j can be obtained from encrypted values sent by TA_O. If the validation is successful, TTP replies to SP. The message is encrypted with the key k_{i_2}. SP can then send the identifier of the puzzle, z_j, to the user and the rest of the communication can be encrypted with the key k_{i_1}.

The formal description of the protocol is done in a common syntax based on [6] and examples from [28]. It is presented in Fig. 4.

Step	Initial phase	Authentication phase
1	$TTP \rightarrow O$: $\{R, x_i, s_{t+i}, k_{i_1}\}_k$	$O \rightarrow U$: $\{R, z_j, T_j, k_{i_1}\}_k$
2	$TTP \rightarrow SP$: $\{R, x_i, s_1, \ldots, s_t, k_{i_2}\}_{k_{i_1}}$	$U \rightarrow SP$: $\{R, z_j, T_j, k_{i_1}\}_k$
3	$O \rightarrow TTP$: $\{z_j\}_{k_{i_1}}$	$SP \rightarrow TTP$: $\{K_s, z_j\}_{k_{i_2}}$
4		$TTP \rightarrow SP$: $\{Yes/No\}_{k_{i_2}}$
5		$SP \rightarrow U$: $\{Yes/No\}_{k_{i_1}}$

Fig. 1. The formal description of the protocol

5 The Proposed Scheme

Our system has at least four parties: TA (the trusted authority), *mix*, *counter* and voters (users).

The trusted authority (TA) is responsible for creating a list of users who are eligible to participate in the election and for authentication data that allows them later to vote. It is similar to a manager in an agent system.

The *mix* distributes the data required for an election to all voters, checking if they are eligible. The *mix* is similar to the 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 than mobile, residing at a specific host and denoted as A_M.

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

The basic steps of the election are:

(i) TA creates the set of credentials and the list of registered users and sends them to the *mix* agent A_M.

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

(iii) The user sends its vote along with credentials to the *counter* agent A_C: the *counter* checks the credentials with TA and if they are proper the *counter* agent A_C sums up the vote, publishing a proof attached to the vote.

Assumptions: TA is trusted to create the 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 the proper proofs for verification. All three parties are independent and are trusted not to cooperate (at least, the *mix* and TA are trusted not to conspire).

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.

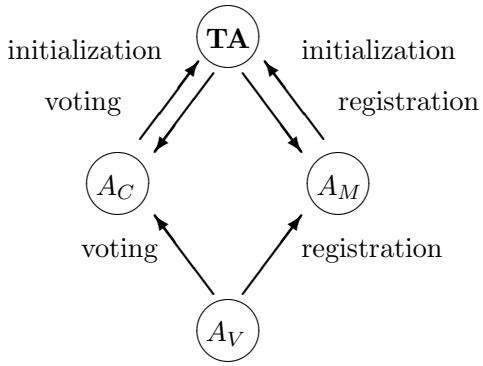

Fig. 2. The architecture of the proposed system

The following notation is used in the protocol:

- secret symmetric keys: k, k_i^C and k_i^M are the secret keys for a symmetric cipher, created by TA,
- functions: E (encryption), h (hash function) and g (will be 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 be 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 share (s_i denotes the i share), K_s is the secret, t denotes the threshold and n is a 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). TA creates the system with n shares with the threshold t. The $t-1$ shares are for A_C, the rest will be distributed among the users by the *mix*. For each secret, TA creates an identifier x for all $n-t$ subsets of shares that allow to recreate the secret. The value of x for each subset is created in a random way. They are presented in Table 2.
For each subset of shares (identified by x_i) TA creates a puzzle for A_M, $p_M = E_k(R, x_i, s_{t+i}, k_i^M)$, and a cryptogram $d_C = E_{k_i^M}(R, x_i, s_1, \ldots, s_{t-1}, k_i^C)$ for A_C. The puzzle p_M is encrypted with a weak random key (k) and then sent to A_M. The cryptogram d_C is encrypted with a key from an appropriate *mix's* puzzle (k_i^M) and destined for A_C. For each generated subset TA stores $\{x_i, K_s, k_i^M, k_i^C\}$ (and optionally also shares). For a single secret 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. TA has to create at least N_u puzzles in total

Table 2
Subsets of shares, generated by TA

Identifier	*Counter's* shares	*Mix's* share
x_1:	$s_1, \ldots, s_{t-1}$	s_t
x_2:	$s_1, \ldots, s_{t-1}$	s_{t+1}
	...	
x_{n-t}:	$s_1, \ldots, s_{t-1}$	s_n

(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. A voter registers within A_M to obtain 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 IDs and sends it to the *mix* agent as a list of eligible voters. The voter utilizes the GQ protocol to prove knowledge of the secret for the presented ID.

In the presented scheme the *mix* is a method to create an anonymous channel between the voter and TTP. It also creates ballots for users, performing instead of them necessary computations.

After validating the user, the *mix* solves a randomly selected puzzle received from TA and obtains authentication data for the user: a share s_i, a key for appropriate A_C puzzle, k_i^M, and the identifier x_i. Data from a single puzzle is utilized for one user. This data is used to create a set of ballots (B) for the voter. Each ballot is encrypted with a weak, random key (similar to the one utilized to create puzzles for A_M). Each ballot consists of a vote (v) encrypted with x_i (utilized as a secret key for a symmetric cipher), the share and the key from the solved puzzle and a proof. The proof is a digest of the vote v and $g(x_i)$: $h(v, g(x_i))$. The function g is known only to 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 an additional secret string of bytes or a symmetric cipher with a secret key. Note that the strong one-way hash function used with a secret argument has the same functionality as a digital signature but is much more efficient. To allow TA to verify the proof, the *mix* sends $E_{k_i^M}(g(x_i))$ to 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.

Voting. The voter sends the ballot $b_f = E_k(R, k_i^M, s_i, E_{x_i}(v_f), h(g(x_i), v_f))$ with a chosen vote v_f to A_C. The *counter* solves the ballot and extracts s_i and k_i^M. It uses the key k_i^M to decipher all puzzles, which it obtained from TTP in the initial phase. If the k_i^M is proper, one of the puzzles had been encrypted with it.

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

A_C extracts from its puzzle k_i^C, x_i and shares. Next, it 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 its puzzle) and sends it, along with x_i, to TA. TA uses x_i to find the appropriate key k_i^C and decrypts the secret. TA compares the extracted K_s with the one stored in the database. It also uses corresponding k_i^M to mark $g(x_i)$ published by the *mix* agent. If the two operations are successful, TA sends a reply to 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 to send the result as a 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. A_C can also convey x_i to the voter as a confirmation of the fact of casting the ballot (to fulfill 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 TA can verify all pairs of votes and proofs utilizing $g(x_i)$ values.

5.2 Formal Specification

The notation used in this specification is based on the one used for SVO logic [29]. An encrypted message m with a 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) $A_V \rightarrow A_C$: $\{R, k_i^M, s, \{v_f\}_{x_i}, h(g(x_i), v_f)\}_k$

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

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

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

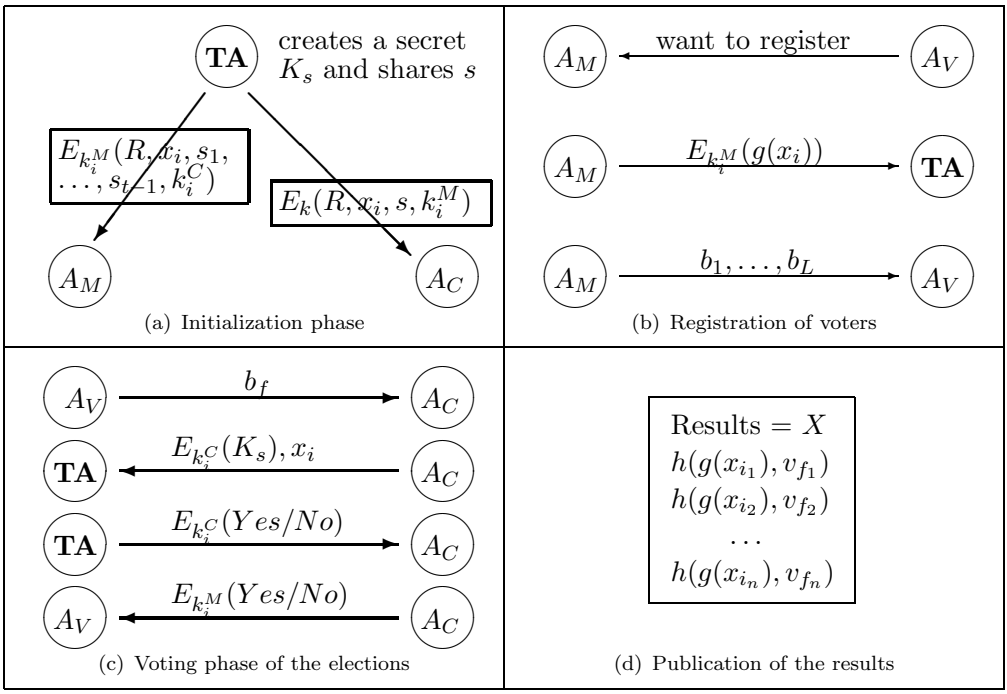

Fig. 3. Phases of the protocol

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

Registration phase. During the registration phase only the *mix* has to make

some computations: solve a puzzle from TA and create ballots for users. Solving a single puzzle is not computationally hard. As the tests indicate the time to solve a single puzzle is small, from less than a second to over a minute, depending entirely on the key's size. Note, that the actual size of a 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 introducing additional *mix* agent A_M. The *mix* also has to create a proof for each ballot: $h(g(x_i), v)$. We assume, that g is an efficient function (the best candidates are symmetric ciphers or hash functions), 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. Thus, 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 the time required to generate $g(x_i)$ and the actual hash (it 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 the size: $b_{size} = s_{size} + R_{size} + k^M_{isize} + E_{size}(v) + h_{size}$. It is easy to calculate that the total size of a 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 a 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 no 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 a 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 Remainder Theorem. The results of the tests show that this operation is rather fast. TA needs just to find appropriate keys in the database, decrypt the secret and validate it (simply compare it to the stored one). All those actions are rather fast. Also, the final operation of A_C is very efficient, since it is only encryption of the data replied to the 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. However, sensibly selected number of voters per *counter* can minimize the risk of delays.

6.2 Communication Effort

(i) Initialization phase: 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 the appropriate party with the speed *2 Mbit/s* is less then 50 milliseconds for systems with n raging from 50 to 500 for data

sent by TA to A_M and up to 5 sec for data transmitted to 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 the size of approximately 128 bits to TA and a set of ballots, less than 10KB to a user. Even with a limited bandwidth (e.g., *2 Mbit/s*) and a large number of users system can be efficient (sending the puzzles to 100000 users 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 a voter), so no communication overhead can be expected, especially that operations during this phase are rather fast and voter should not have wait long time for reply from the *counter*.

6.3 Security

Completeness. If the user casts a correct ballot then the *counter* is bound to find appropriate cryptogram and recreate a valid secret. Moreover, it is able to extract the identifier x_i, decrypt the vote and tally it.

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 the success are extremely low. Even if the adversary is a dishonest voter and he 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 it was tallied, but the *counter* can prove otherwise by publishing the proof, which is hard to forge without help of the *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/hers vote.

The ballots are encrypted with a symmetric cipher utilizing x_i as a secret key. 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 a selected vote.

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

the one actually casted by the voter. If his/her proof is not published, he/she can show his chosen ballot to *TA* and it can verify whether suitable data in the database was marked as used (or the confirmation stage can be added to the protocol, as described in Section 5.1). In the second case *TA* can verify the proof presented by the *counter* and settle the dispute. *TA* can also estimate the minimal and maximal number of voters taking part in the elections basing on the amount of $E_{k_i^M}(g(x_i))$ values created by A_M and the number of successful interactions with A_C.

Receipt-freeness. The scheme allows the voter to verify if his/hers vote was tallied by publishing the proofs $(h(g(x_i), v_f))$, but does not enable him proving to any third party what vote was casted. The voter can present the coercer a ballot $b_f = E_k(R, k_i^M, s_i, E_{x_i}(v_{f_1}), h_{f_2})$ for the published hash $h_{f_2} = h(g(x_i), v_{f_2})$. 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. Moreover, it is easy for the voter to create a false ballot with a selected vote and a published hash, since the coercer cannot verify the proof $h(g(x_i), v_f)$, because the function g is secret and known only to the *mix*, and the list of produced $g(x_i)$ values is known only to TA.

Robustness. The non-participating voter does not influence the course of elections. All appropriate authorities (*counter*, *mix* and *TA*) have to take part in 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 extend the application based on the agents. One of the methods is to create hierarchical structure, based on triples of agents: A_M, A_C and A_{TA} (an agent representing TA). Each triple is responsible for registering users in a certain region and for 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 Conclusions

To improve efficiency of the presented method some modifications can be introduced.

Registration phase (*mix's* actions) can be divided into two independent subphases: the first one, for receiving authentication data from 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.

The ballot for our scheme is practical and easy to create. Moreover, 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 a typical e-voting system than the simple one-way hash function but they require some computations on user's

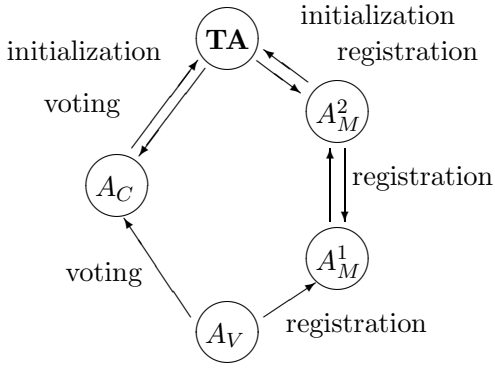

Fig. 4. Modified architecture with more distributed trust

side, which is a disadvantage, especially for mobile devices or cards.

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)** (2000), pp. 1431–1440.

[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. and J. Bloom, *A modular approach to key safeguarding*, IEEE Transactions on Information Theory **IT-29(2)** (1983), pp. 208–211.

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

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

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

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

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

[9] Chaum., D., *Secret-Ballot Receipts: True Voter-Verifiable Elections*, IEEE Security and Privacy **2** (2004), pp. 38–47.

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

[11] Damgard, I., J. Groth and G. Salomonsen, *The Theory and Implementation of an Electronic Voting System*, in: D. Gritzalis, editor, *Secure Electronic Voting*, 2003, pp. 77–100.

[12] Damgard, I. and M. Jurik, *A generalization, a simplification and some applications of Pailliers probabilistic public-key system*, in: *Proceedings of public key cryptography, PKC'01*, LNCS **1992**, 2002, pp. 119–136.

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

[14] Fujioka, A., T. Okamoto and K. Ohta, *A practical secret voting scheme for large scale elections*, in: *Advances in Cryptology – ASIACRYPT'92*, LNCS **718**, 1993, pp. 248–259.

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

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

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

[18] Juels, A. and M. Jakobsson, *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 Art of Computer Programming*, Addison-Wesley, NY, 1989.

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

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

[22] Merkle, R., *Secure Communications over Insecure Channels*, Communications of the ACM (1978), pp. 294–299.

[23] Ogata, W., K. Kurosawa, K. Sako and K. Takatani, *Fault-tolerant anonymous channel*, in: *Proceedings of the ICICS 97*, LNCS **1334**, 1997, pp. 440–444.

[24] Park, C., K. Itoh and K. Kurosawa, *Efficient anonymous channel and all/nothing election scheme*, in: *Advances in cryptology – EUROCRYPT'93*, LNCS **765**, 1994, pp. 248–259.

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

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

[27] Sampigethaya, K. and R. Poovendran, *A framework and taxonomy for comparison of electronic voting schemes*, Computers & Security **25** (2006), pp. 137–153.

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

[29] Syverson, P. and P. van Oorschot, *On unifying some cryptographic protocols*, IEEE Security and Privacy (1994), pp. 14–28.

[30] Zwierko, A. and Z. Kotulski, *A new protocol for group authentication providing partial anonymity*, in: *Proceedings 1st EuroNGI Conference on Next Generation Internet Networks – Traffic Engineering – NGI'05*, 2005, pp. 356–363.