Proposal of a new online voting system

Gerold Grünauer
geroldgruenauer@web.de

In this paper a new easy to use, secure and transparent online voting system is proposed. The new scheme can be easily used by small associations or organizations. The new scheme most notable protects the voter against common cheap and effective Trojan-horse / keylogger and phishing attacks by using an new concept of voting-TAN-lists.

1. Motivation

Elections are the essential part of every democratic society and organization. Hence it is very important to hold up as many elections as possible. Unfortunately, elections come with big administrative efforts and costs.

In order to circumvent the drawbacks of conventional "physical elections", a lot of people suggested the use of cheaper online voting systems.
Today a lot of alternative e-voting systems have been proposed. Some of them are already used. Unfortunately most of them do not even fulfil the most basic security requirements, whereas other systems are provably secure, but completely impractical. Furthermore a few e-voting scandals destroyed the peoples trust into these voting schemes.

As a result, we have the need for a new easy to use, practical, secure and transparent online voting scheme, that can not only convince experts but also citizen, lawyers, …

In this paper such a new easy to use, secure and transparent online voting system is proposed.
The new scheme is furthermore secured against very dangerous, effective and cheap keylogger and phishing attacks by using a new type of voting-TAN (transaction number). These attacks have normally due to insecure client computers extraordinarily high chances of success. However, when a user wishes to cast his ballot, he never tells his computer, which candidate he chooses, but instead enters the associated TAN for this candidate. As Trojan horses never can see the voters decision or any information needed to vote for another candidate, the attacks mentioned before are rendered useless.

2. Requirements

In order to be accepted by the voters or the law, an online voting system must be as secure as a conventional “physical” election.
Hence online elections have at least to meet the following requirements:

1. Only eligible voters are allowed to vote
2. Every voter shall only cast one vote
3. It must be impossible to alter anybodies vote
4. The voting procedure must be anonymous
5. The complete voting procedure must be transparent

Like any other electronical “mass product” that is used by a lot of inexperienced users, the new voting scheme must also be:

- *User-friendly*
- *Robust,* i.e. even in the case of malfunction or failure of a component, the system must still continue to work. (failure describes here also the case, that the components security has been broken, i.e. a “security – failure”)
- *“Hacker secure”*, powerful, cheap and well known “standard” attacks like DoS (Denial of Service), Trojan horses, social engineering, … shall not work
- *Transparent*, i.e. every voter (therefore all users and experts) can check the systems integrity without any troubles (i.e. they do not have to register as auditing personal, …)

3. Security model

Security is the essential part of the complete voting system and the only reason, why such a complicated cryptographic scheme has to be used at all. In the following section the underlying security assumptions of this scheme are discussed.

3.1. General assumptions

The new voting scheme is based on the assumption that a constant amount of the users involved are malicious regardless which role they play in the scheme. Hence a “trustworthy authority” – as required by a lot of cryptographic schemes – cannot exist. Consequently, the best type of protocol would be a “self enforcing” one, that cannot be fooled by any individual or group of malicious users.

Unfortunately most practical cryptographic schemes do not have this property but have the need for such “trustworthy authorities”.

This problem is in the proposed scheme approached by three general methods:

- Whenever possible design the algorithm, such that they are self-enforcing
- If absolute trustworthiness is required (see Anonymity), split up the affected agency into smaller entities, such that all single entities are required to cooperate in order to compromise the scheme / security aspect
- If it can be assumed, that a certain agency won’t be compromised in a certain way, implement a mechanism that won’t eliminate the specified security threat but only detect a possible security failure (however, in this case most times the election would be invalid)
- If it is not possible to eliminate a threat completely, the possible attack shall require more efforts as to be profitable.

3.2. Anonymity

Unlike data integrity and authentication (known users shall be authenticated by a server), anonymity is a problematic requirement of every voting system.
The voter has on the one hand to be legitimated as valid user (hence he cannot be anonymous). On the other hand he has to cast “unwatched” his ballot, offering the user a wide range of possibilities (casting fake ballots, …).

Hence the new scheme has not only to enforce the privacy of the voter but on the other hand has to enforce the integrity of the cast ballot. Furthermore the complete vote casting procedure must be practicable and robust.

In order to solve the “anonymity problem”, the proposed voting protocol uses a new type of publicly verifiable ballot mixing system.

4. The new scheme

The new voting system consists of three major steps:

- **System setup**
  In this step, the voting scheme is set up: The necessary election workers and certain system parameters are chosen. A verified list of eligible voters is created and each voter is provided with his voting credentials. Finally all users, who want to use the system, have to sign in.

- **Election**
  After the setup during a specified period, all users can cast their votes.
  The election workers must be available in case of unexpected problems.

- **Vote count and integrity check**
  After the election is closed, the votes can be counted and the integrity of all databases is checked.

For the system setup and vote count, several election workers are needed. These election workers must not be compromised (in terms of security) altogether. Otherwise the voters anonymity might be in danger.

4.1. Used algorithms and concepts

In order to perform the above described tasks, several well known and some new algorithms and concepts are needed. For each algorithm the notation and certain practical and theoretical remarks are addressed.

The used notation in this paper is always printed in italics. Each algorithm is considered as a function with certain input / output parameters. The defined “functions” (in terms of programming languages) can be “called” later in any new algorithm and in the voting procedures.

The mentioned functions must be present in every implementation of the voting system.
Important note: In the following descriptions the term User refers to both the user as person and the users software. In the appendix you will find a graphical overview of all algorithm where user and client software will be distinguished.

Encryption / Decryption and digital signatures

The voting system uses only public key cryptography. Therefore for all encryptions we need a public key, for all decryptions we need the corresponding private key.

Encrypted Data = Encrypt(Data, Public Key)
Data = Decrypt(Encrypted Data, Private Key)

Signature = Sign(Data, Private Key)
Boolean (true / false) = Verify(Data, Signature, Private Key)

Remarks
The encryption function must accept arbitrary input (for example, Data might be a String, or the output of a previous Call of the encryption function).
The encryption function must be randomised (i.e. you cannot reproduce the same output using the same input). This feature is required for the ballot mixing system.

The decryption function shall detect decryption errors. It is however not necessary to implement this feature, as this will not affect security.

The signature generation function Sign must accept arbitrary input. The Verify function checks a previously generated signature using the same Data as input, as the Sign function.

When using the RSA cryptosystem (with fixed relative small encryption exponent e, for example e = 65537 or e = 17), the size of each database entry can be shortened without affecting security and the computation time needed for the ballot mixing net can be minimized (transmitting a ballot requires 15 to 20 successively calls of the encryption function)

Secure Checksum / Hash

For signature generation / verification, the secured database and the ballot mixing protocol a secure checksum (Hash – Function) is needed:

Checksum = Hash(Data)

Remarks
Any collision-free and non-invertible cryptographically secure hash function can be used for the voting system.

TAN / TVN authentication

The client / server computer may be infected by various malware, including trojan horses, viruses, key-logger, …. In order to frustrate the hackers behind, a strong authentication for each security critical operation between server and client is needed. The TAN / TVN (**T**rans**a**ction **n**umber, **t**ransaction **v**alidation **n**umber) procedure provides an easy way to protect the client and the server form malware.

TAN-Authentication(TAN / TVN-list)

Algorithm
Description: The user wants to validate an action he is currently performing in cooperation with the server.

1. The server looks up the current TAN index, the user has to use for the authentication step. This index is send to the user.
2. The user(person) enters the requested TAN. The TAN is send to the server.
3. The server checks, whether Hash(TAN) exists in the users TAN table.
   If it doesn’t exist the server terminates the process with an adequate warning.
   Otherwise the currently performed action is finished and the corresponding TVN is send back to the user. The current TAN is invalidated by recording the entered user TAN in the users TAN table.
   The TVN may be for example computed by: TVN = Hash(TAN + ServerTANKey)

Remarks
The TAN index shall be fixed for a specific action. A voter has only to cast a ballot and to sign-in. Hence the user only needs two TAN / TVN pairs. One for each of the two actions.

It is vital that the user is clarified about the TAN procedure: The user shall never enter the TAN’s anywhere except in the voting application. If an invalid TVN is returned by the server, the user has to call an operator (because this cannot happen by means of technical failure but only due to security failure).

The server shall not know the TAN – values itself but only a hash of these. As a result, a compromised server cannot pretend that a specific user has voted, because he would have to know the users TAN in order to claim this (the entered TAN is saved during the invalidation). The server would have to work together with the voting administration in order to accomplish such an attack.

The TAN lists for each user must be generated outside the server (because of the before mentioned reason). Each user has to receive a TAN list prior the sign-in and voting period.

The just described TAN – authentication protocol should not be confused with the already mentioned Voting-TAN-list.

Secured Database

In order to store ballots and other system parameters, a publicly verifiable database is needed. It must be possible for every user to detect manipulations (deletions, insertions, modifications). In case of a manipulation the election officials have to decide what to do (in most cases, the election will become invalid).
Furthermore the voter gets a proof that the server added his vote to the database.
The following "function" requires the possibility to contact the server (for example over the internet).

Proof of database-insertion = Add To Database(Data)

<u>Algorithm</u>
Description: A user wants to have data inserted into the verifiable database located at the server. The user obtains during this procedure a "proof of database-insertion which can be used to check the database – integrity at any time.

1. The user sends the server his Login – credentials (UserID / PIN) if he didn't already
2. The database server checks the Login – Data and checks the users TAN table, whether the authentication TAN has already been entered (if so, the user already submitted the data, see step 6)
3. The user sends his data-block he wants to have inserted (for example his ballot), to the server.
4. The server checks the data-blocks syntax, … as far as possible and computes the "proof of insertion" certificate:
   ```
   Certificate = {  Cert_Block =
                    {
                        Chain – Hash all of previous database entries
                            $ChainHash_i = Hash(Entry[i] + ChainHash_{i-1})$,
                        Hash-Sum of the users datablock
                            Checksum = Hash(User-Data)
                        EntryID = i (ID of last inserted entry)
                    }
                    Cert_Signature = Sign(Cert_Block)
                 }
   ```
   The certificate is send to the user.
5. The user checks the certificates signature and the checksum of his own datablock. If the signature or the checksum is incorrect, the user has to call an operator (because the server or his own computer has been compromised). Otherwise he performs:
   TAN-Authentication(User's TAN-list)
6. By saving the users submitted TAN (this is done in the TAN-Authentication) the server can prove that the user already tried to put his entry into the database.

Remarks
In a real world application, the user already authenticated himself before the database insertion using his UserID / PIN. This step (1) can be omitted in this case.

As the experienced reader might see: Only the database entries before and the users new entry are protected by the certificate. Hence it may be desirable, that a user can obtain an integrity check certificate at any time. This certificate will (of course) only contain the ChainHash and not the Hash of the users data.

The database manipulation detection of course only works, if enough users (not all users are necessary though) and especially the last users check the database using their Check-Certificate. If, however, the election workers are required to obtain such a check-certificate at specified time intervals, the detection will work always.

The database – server cannot just make a voters-ballot / sign-in block vanish into thin air, because the voter gets an insertion-certificate when he submits the data. The server cannot fake the data-packet, because the voter can check the correctness of the database entry before he enters his TAN. Without the TAN however, the server cannot “vote / sign-in” for the user. If the user later checks the database (when it is publicly available) he can check his entry. The voter has therefore to be encouraged to save his check-certificate and to perform this validation.

Ballot mixing – Anonymous Database

Protecting the voters anonymity during the election is a vital task of the new voting system. In order to accomplish this task, a new publicly verifiable ballot – mix – net is used. A mix-net must have the following properties:

- All inputs must be mixed in a way, that without the help of all (or a treshold) of the involved “mixing” authorities, the connection between the inputs and the outputs cannot be recovered
- It must be impossible for the voting authorities to alter the output data (i.e. change the ballots). This property must be publicly verifiable.

The presented new solution for designing the “mix net” satisfies the above mentioned requirements. It will be, however, assumed, that the mixing authorities won’t try to alter the packets as it is not possible to detect which authority tried to manipulate (it is only possible to detect the manipulation. Furthermore, it will be assumed, that the submitted ballot – packets are formed in the way described later.
The mix – net consists of several procedures. The voters procedure to add the data and the procedures to mix the ballots.

Add To Anonymous Database(Data)

Algorithm
A user wants to insert his data into the anonymous database. He needs the public keys of all mixing authorities.

1. The user obtains all public keys (Public Key[0], ... Public Key[n] of the voting authorities.
2. The user encrypts sequentially his data-packet with all Public Keys
   c_1 = Encrypt(Data = c_0, Public Key[0])
   c_2 = Encrypt(c_1, Public Key[1])
   …
   c_{n+1} = Encrypt(c_n, Public Key[n])
3. The user computes the checksum c = $\sum_{i=0}^{n} Hash(c_i)\ Mod\ M$
4. The user submits the data-packet to the secured database:
   Add To Database(c_{n+1}, c)
5. The user saves the insertion – certificate in order to check the database at a later moment

Mix and Open the Packets

Algorithm
The authorities mix the ballots and shuffle them.

1. Sequential each mixing authority (starting from n to 0) has to perform:
   - Obtain all submitted data-packets from the server
   - For all submitted data-packets
     c_i = Decrypt(c_{i+1}, Private authority key[i])
     Note: This cancels out in the correct order all encryptions, the user applied
   - Shuffle all now decrypted packets and send them back to the server
2. The server obtained now all decrypted packets and the intermediate decrypted packages. The server computes the sum of all the hashes. This sum shall be equal to the sum of the user – submitted checksums.

Remarks
Alternative systems with the same properties might be used as well.

The last step can be repeated by each user, because the result of the mixing operation and the packet – database are publicly available.

The exact relationship between submitted packets and users can be reconstructed by two possible methods:

1. Each mixing authority reveals it's shuffle relationships. To secure the mix – net against this type of attack, we can increase the number of mixing authorities (in practice 4 to 8 trusted "mixers" may be sufficient)
2. The binary knapsack problem c[Specific User] = $c'_i * x_i$ is solved.
   This problem can be addressed by sufficient large parameters t.
   If we have 30 voters, we should have 15 to 20 mixing authorities.
   For these parameters, we have: 600 different c'_i values. 20 of these 600 values

have to be chosen to sum up to the users check-sum. This knapsack problem cannot be solved by currently known algorithms.

3. If some consecutive mixing authorities work together (i.e. reveal their shuffle relationships), they can make the problem easier, as they know a part of the correct x_i. This problem can be addressed by either using more distinct mixing authorities or (if not enough election workers are available) by arranging the existing mixing authorities in a circle. (Mixer 1 -> Mixer2 -> Mixer3 -> Mixer1-> …)

It is not possible for the mixing authorities to manipulate the packets. In case of a manipulation, the checksum would change. However, the submitted checksums are already protected by the ballot database. Hence it is necessary to fake a packet in such a way, that the checksum is not changed. This requires the hash – function to be broken (which is assumed to be not possible).

A malicious mixing authority might corrupt the packages in order to disturb the voting process. Although, this action will be detected at once after “mixing” the ballots, the author knows of no suitable way to identify the “corrupting entity”. It is, however, assumed that in practise (in small-scaled elections) this won’t happen. The problem can be addressed by splitting up each authorities private key, such that the “mixing” can be done by alternative mixing authorities.

A user can disturb the voting procedure by submitting fake checksums. This problem can be addressed by using a slightly modified “mixing procedure” – this method will be, however, described in another paper). It is assumed, that in small-scaled elections this won’t happen.

Because the data submitted to the anonymous voting database cannot be checked by the server (as it is encrypted multiple times), any data could be send. Especially fake TANs may be a threat. Hence it is important, that all Voting-TAN-Lists (as described in the next section) are affixed with a stamp (so the voter can claim he got this list from an official site). Furthermore, it is necessary, that the check-digits included in the Voting-TANs cannot be computed without machine help and designated (or manipulated) software. A user not trying (by examining the source code, ...) can therefore never enter an invalid TAN, as the check-digits are checked by the client software.

Voting TAN – lists

Trojan horses, spyware, keyloggers, faked online – voting sites (= phishing),... pose the greatest risk for modern online security systems. Such attacks can be accomplished relative easy and cheap by even non-expert users (by using trojan-horse “development kits”). As a lot of client computers might be insecure, a “man in the middle” (i.e. the hacker in case of a phishing site, stolen User ID / PIN, …) shall not be able to vote (or vote in another way the client might have) with the deduced voting credentials.

Furthermore, some user might experience internet problems just on the election day (the problem may be real or be faked by the user in order to gain any advantages, …). As a result a telephone “voting hotline” should be offered, where such users can cast their vote via telephone. As the operator and his supervisors can hear the voting credentials, the voters “decision” (whom he really voted for) must be protected.

These two desirable goals are achieved by using the new concept of “Voting Tan lists”. Every voter gets anonymous an additional TAN – List (don’t confuse them with the TAN’s mentioned previously).
The “Voting Tan List” contains a list of all candidates and for each candidate a corresponding TAN, you have to enter during election in order to vote for him.

In order to vote, the user has just to enter one of the corresponding candidate voting TANs. After the eletcion, the TANs are revealed and can be publicly checked.

The administration has now to reveal, what candidate is associated with each user-entered TAN. Instead of just announcing the outcome (the administration could announce anything) they have to reveal for each TAN a list of numbers (all numbers are shuffled together – hence you cannot say which number is connected to which TAN). One of these numbers is marked as candidate number, the others are just random.
Prior the election a (sum able) checksum is revealed each TAN, that can be computed with the numbers mentioned above. Hence all users can check, that the correct candidate numbers are revealed without knowing which candidate number belongs to which TAN.

The lists must be prepared during voting setup and send to the users prior to the election sign-in.

List generation

1. A administrative authority prepares n TAN - lists (n shall be greater than the number of voters) in the following manner:
   - For each list and each possible candidate, k different TAN – numbers are randomly chosen.
   - Each TAN number is extended by some check – digits (these digits are used by the client software to check, whether they have been entered correctly)
   - For each TAN – list, a random identification number TAN-ID is generated
   - Each complete list (candidates TANs and the TAN-ID) is printed, affixed with a stamp or "physical" signature and put into (similar) envelopes
2. For each candidate TAN, an additional voting numbers list is generated:

   TAN -VotList = { a_0 = (Random-Part | Candidate-number),
   a_1 = (Random-Part | 0),
   …
   a_t = (Random-Part | 0) }

   For each TAN-VotList a "checksum" is computed by: c = $\sum_{i=0}^{t} Hash(a_i)$

   The server obtains for each TAN number: The associated TAN-ID, the TAN-VotList checksum c) and Hash(TAN), but not the TAN numbers itself
   (This database has to be publicly available for testing purposes)
3. All envelopes are conventionally mixed by the election workers.
4. The mixed and closed envelopes are added to the voters credential letter (where he can find his User ID or PIN)

The physical mixing ensures, that it is unknown to the election workers, which voter got which voting TAN list.

At the end of the votes, each ballot will contain a candidates TAN. The vote revealing step must the be performed:

Vote-Revealing

1. For each TAN found in the ballots database, the administration looks up the associated voting number list and creates a large suffled list containing all voting numbers:
   Example:
   Looked up voting number lists: {a00, a01, a02, a03}
   {a10, a11, a12, a13}
   {a20, a21, a22, a23}
   The outcome could be then:
   {a21, a10, a13, a03, a22, a00, a20, a02, a11, …}
2. All users and the server can check, whether
   $$\sum_{Set\ of\ all\ found\ TANs} c_{Current\ TAN} = \sum_{Set\ of\ all\ revealed\ voting\ numbers} Hash(a_{Current\ Voting\ Number})$$
3. The server filters out all candidates and counts the votes
4. The result of the election can now be announced

4.2. The new voting system

The actual voting system can be now described:

1. System - Setup

1. The election administration creates a verified list of eligible voters.
2. The voting administration chooses several trustworthy “mixing authorities” required for the ballot mixing system.
3. A trustworthy entity creates for all voters the Voting-TAN-lists as decribed before.
4. Each user is provided with two TAN / TVN pairs (one for sign-in, one for casting the ballot) and a UserID / PIN pair. The UserID / PIN shall be send by email or a second letter.
   The enveloped and already shuffled Voting-TAN-List is included in one of the letters send to the user. The Voting-TAN-list shall not be send by email.
   Each eligible voter verifies that he received the necessary letters and signs-in:
   1. The voter enters in his client-software his UserID / PIN in order to login
   2. The voter enters the Voting-TAN-ID used to identify the corresponding TAN – List. This number is (optionally together with the Hash of a user chosen voting-password) send using the anonymous database.
      Add To Anonymous Database(TAN-ID, [Hash(Voting Password)])

   The database-insertion-certificate is stored for later checks.
   During this step the voter is asked the Sign-In TAN and gets back the TVN. If there is any problem, he calls at once the operator.
5. After the sign-in deadline, the anonymous sign-in database (containing the TAN-IDs and Hash values of the voting passwords (if used), is opened and shuffled by the mixing authorities using the *Mix and Open Database* procedure as define above.

If you don’t require such strict anonymity as postulated in this paper, you may omit the ballot mixing and trust only in the voting-tan-list. This is, however, not recommended.

For the case of unexpected events, a hotline shall be installed in order to give the voters the opportunity to resolve any problem.

If you use the optional additional voting-password (as defined in step 5), no group of administrative agencies knows all the data (except they can install a Trojan-horse on the clients computer) to be able to cast a ballot instead of the user.

In the case, that user credentials are believed to be compromised, the sign-in entry can be changed without problems prior to opening and mixing the sign-in database. The user is presented new voting credentials (some more Voting-TAN-lists as necessary may be computed therefore). After the sign-in deadline, the database cannot be changed, because otherwise the anonymity of the voter may be in danger.

Now the server has a list of used TAN-IDs and optional voting password hashes. New voting tan lists cannot be introduced anymore, as the database cannot be manipulated.

The voting password may be generated automatically by the client software and displayed to the user for write-off. The generated password should contain some check-digit and the voter should be asked a minute later to retype it (in order to help the user no forgetting the password). The TAN – letter shall contain a designated space for this purposed.

2. The election

1. Each voter logs in using his UserID / PIN and enters the Voting-TAN of the candidate he wishes to elect. Optionally (if already used at the sign-in) the user also has to enter his voting-password
2. The client software transmits using the anonymous database / ballot mixing system the data packet (Voting-TAN, [Voting-Password])
   Add To Anonymous Database(Voting-TAN, [Voting-Password])
   During this step the voter is asked to enter his Votecast-TAN. He checks the received TVN and stores the insertion-certificate for later checks.

Should there be any technical problems that could withhold the voter from casting his ballot, he shall call the problems hotline. The hotline-operator can cast the vote for the user without seeing it. By returning the TVN number, the voter can be sure, that his vote has been entered into the server.
Unfortunately, the voter does not get the insertion-certificate. The operators of the hotline and their supervisors can, however, be instructed to use the received certificate to validate the database after the election. Moreover, not every voter has to validate the database, because each certificate checks the integrity of all casted ballots before.

3. Vote count and integrity check

1. The mixing authorities open and mix the ballot database using
   Mix and Open Database
2. The "voting-tan-list" administration reveals all the voting-numbers as described above. The result of the election can be now computed and displayed to the voters.
3. The server, the election workers (and each user willing to do so) can now check the integrity of the involved databases.

For small-scaled elections it may be sufficient to just provide all databases form the server. The user can install the required software products in order to validate the data.

5. Summary

In this paper a new easy to use, practical and secure voting system for small to medium scaled elections has been presented. The new algorithm is furthermore especially secure against very popular Trojan-horses and malware attacks.

The new concept of anonymous, verifiable Voting-TAN-lists may be userful for other application than online voting systems, too.

As this scheme is pretty new, further cryptanalysis is still necessary. The author would be thankful, if you can send him any ideas, attacks, …