An electronic voting system supporting vote weights

Charlott Eliasson[1] and André Zúquete[2]

[1] Blekinge Institute of Technology, Karlskrona, Sweden
chel01@student.bth.se
[2] IEETA / University of Aveiro, Portugal
avz@det.ua.pt

Abstract. Typically each voter contributes with one vote for an election. But there are some elections where voters can have different weights associated with their vote. In this paper we provide a solution for allowing an arbitrary number of weights and weight values to be used in an electronic voting system. We chose REVS, Robust Electronic Voting System, a voting system designed to support Internet voting processes, as the start point for studying the introduction of vote weights. To the best of our knowledge, our modified version of REVS is the first electronic voting system supporting vote weights. Another novelty of the work presented in this paper is the use of sets of RSA key pairs with a common modulus per entity, for saving both generation time and space.

1 Introduction

Electronic voting protocols should respect some basic properties of elections, namely accuracy, democracy, privacy and verifiability [1]. One of those properties, democracy, states that "*each eligible voter is allowed to vote and to vote at most once*´´. This property is a normal requirement in most electoral processes but is far from being axiomatic. In fact, there are some scenarios where the votes from some persons have, or should have, a different weight than the other votes. For instance, some communities, such as the associates of a football club, can have a weight somehow proportional to duration of the membership; or a member from an administration board can have a different weight for solving draw situations. Thus, voting weights are a useful, real-life form of differentiating participants in voting processes. However, as of today we have no knowledge of electronic voting systems with support for weighted votes.

This paper will focus on the support for weighted votes a particular voting system — REVS (Robust Electronic Voting System [2]). REVS is a fault-tolerant electronic voting system designed for voting through the Internet. It uses replication as the basic mechanism to tolerate system failures in communications, servers and voters' applications. Furthermore, it also tolerates failures in the correct behavior of the several entities running the voting protocol: neither voters nor servers, until a certain level of collusion, can interfere with the correct behavior of the system without notice.

Supporting weighted votes means that, when voting, a voter's vote is worth w votes. To implement this service in REVS, several requirements were considered. The basic requirement was to minimize the modifications on the protocol of REVS, in order to reduce the probability of introducing new vulnerabilities in the final protocol. Other more specific requirements were the following:

- Scalability: support for an arbitrarily large set of available weights and support of arbitrarily high weights;
- Efficiency: the performance of the system should not be notably disturbed when extensively using different weights;
- Usability: for a voter it should be equal to vote the "traditional´´ way or using a weight attribute; and
- Anonymity: a weighted vote, or a set of w votes provided by a voter with weight w, should not provide any hint about the voter who cast it.

The final solution respects all these requirements except the last one, anonymity: in some particular cases, a voter can be linked to his vote. In fact, when there is a single voter allowed to cast a vote with a specific weight, or a single voter that actually used a specific weight, the voter can be link to his vote. This problem was not solved in the protocol presented in this paper because we didn't find a fairly easy solution that does not interfere too much with any of the other requirements. Furthermore, this anonymity problem already appears in paper-based voting systems, where voters with different weights use bulletins of different colors or cast their votes in different ballot boxes, one per each weight.

REVS uses RSA [3] keys and Chaum's algorithm [4] for blindly signing votes from authorized voters. For supporting weights we extended this behavior by using sets of RSA keys for the signing process. But, for saving time and space, we used sets of RSA keys sharing the same modulus. As far as we know, this was never used before.

Our solution for supporting weighted votes was implemented in the code of REVS, which is publicly available[3]. REVS is fully implemented in Java, which makes its deployment very easy.

This paper is structured as follows. Section 2 overviews REVS. Section 3 presents some approaches that were considered for adding weighted votes to REVS but abandoned in favor of the solution presented in Section 4. Section 5 discusses the security of the solution. Section 6 presents some details of its implementation. Finally, Section 7 draws some conclusions and forecasts future work. Since we have no knowledge of other electronic voting systems supporting vote weights, there is not one section dedicated to related work.

2 Overview of REVS

REVS is a blind-signature based voting system designed for secure and robust electronic voting [2, 5, 6]. The REVS architecture, depicted in Figure 1, includes a client application, an electoral Commissioner, and a set of electoral servers

[3] http://www.gsd.inesc-id.pt/~revs

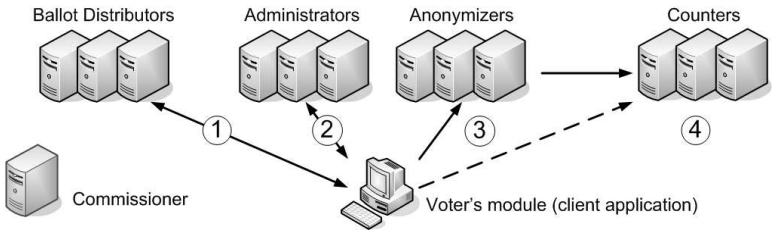

Fig. 1. Architecture of REVS

— Ballot Distributors, Administrators, Anonymizers and Counters. All electoral servers can be arbitrarily replicated for improving load balance, availability or for preventing collusion-based frauds. Valid votes can be cast repeatedly in several Counters, and even in the same Counter, without affecting the final tally.

REVS uses a blind-signature protocol that resulted from the evolution of three other protocols: FOO, EVOX and EVOX Managed Administrators [7–9]. Figure 1 also presents REVS' protocol steps, which are the following:

1 - Ballot distribution. The Voter contacts a Ballot Distributor to get a ballot for a given election. The Ballot Distributor returns him the requested ballot, the election's public key and the election's operational configuration (e.g. servers' keys and locations), all signed by the Commissioner. Voting bulletins are XML documents, signed by the Commissioner, providing a set of rules for presenting and verifying the voting options for voters.

2 - Ballot signing. After expressing his will on the ballot, the voter commits to the ballot with a random bit string and generates a digest of the committed ballot. Then the voter's module generates a random blinding factor, applies it to the digest and sends the result to a subset of the Administrators for signing.

An Administrator, after receiving a signing request, verifies if it had already signed a blind digest for the requesting voter. If not, it signs the vote and saves that signature; if it had signed before, it returns the previously saved data. After receiving the signature, the voter's module removes the blinding factor and verifies its correctness using the Administrator's public key. This process is repeated until a required number of t signatures is collected. The value of t must be higher than $N/2$, to prevent voters from getting more than one valid vote.

3 - Ballot Submission: The voter's module constructs the ballot submission package, joining the ballot, its signatures and the bit commitment. Then he submits this package to the Counters through the Anonymizers, encrypted with a hybrid cryptosystem using a random symmetric session key and the election's public key, concluding the voting protocol. A voter can submit the same package to any Counter as many times as he feels necessary to be sure that the ballot has reached his destination.

4 - Tallying Phase: After the end of the election the Commissioner discloses the election's private key. Then the counting process is performed by the set of Counters and involves the following steps: (i) decryption of submission packages

with the election's private key; (ii) verification that all required t signatures from the Administrators are present; (iii) removal of repeated votes, i.e., the ones with the same bit commitment; (iv) tallying the remaining votes from all Counters.

3 Evaluation of possible Solutions

The basic idea of this work is to give a weight to a vote (or to a voter) and, at some point in the system, multiply the vote by its weight. This can be done in a number of ways and many ideas of how to make the implementation have been considered. Before describing our solution, we start by describing the alternatives considered and the reasons for discarding them when facing the requirements.

3.1 Votes with an embedded weight

A possibility was to include weights in voting bulletins distributed to voters, which would then be copied into the votes sent to Counters. But this approach requires a strong assumption: the voters' application must be trusted not to forge weights. Since the voters' application may be tampered in some scenarios, namely when "voting anywhere" is considered, the voters' side cannot be trusted to give the correct input for the system when weights are considered.

The simple copy/paste of weights could be strengthened by adding a cleartext value of the weight when submitting a blinded vote digest for getting a signature from an Administrator. Then, the weight, checked and signed by all the required Administrators, could be added to the final vote submitted to Counters. A bit commitment value should also be added to the weight to prevent stolen, signed weights, to be used by other voters. The drawback of this approach is that protocol messages from voters to Administrators and from voters to counters would increase in size, namely would double in size. This collides with the requirement of keeping the performance of system close to the performance of the initial version of REVS.

3.2 Casting of votes accordingly to a weight

Another possibility is to multiply a vote by giving the voter w number of votes to deal with. A voter could only fill in one ballot but then the voter's application would run the voting protocol w times, using w different bit commitments.

This approach has many drawbacks. In terms of performance, it scales badly with the value w: (i) The voter would have to keep w different bit commitments; (ii) The voter would have to run the protocol w times; (iii) Administrators would have to keep w signed votes for the voter; and (iv) Counters would have to store and count w different votes for the voter.

Moreover, one of the mechanisms currently used by REVS to deal with fault tolerance — the need of t signatures from N Administrators, $N/2 < t \leq N$ — could allow voters to get more valid votes than the ones they are entitled to. For instance, with 3 Administrators and $t = 2$, a voter with $w = 2$ could get 3 valid votes. We can use well suited N and t values for an election for tackling this specific weight boost vulnerability, but at the end that would complicate the deployment of the system due to the use of weights.

This approach, however, has the advantage of providing anonymity for voters. Furthermore, it gives voters the ability of casting different votes among the set of votes allowed by their weight.

4 Our solution

The solution we chose for this work was to cast different signatures on votes for different weights. The Administrators are given different W signing keys, one for each possible weight, and the knowledge of the weight w for each voter. Furthermore, Administrators return to a voter the weight bound to its vote by the Administrator's signature.

The Counters are given different $N \times W$ validation public keys, one for each Administrator and possible weight. For each vote they use a set of N public keys, corresponding to the signing keys representing the same weight. When voters download a voting bulletin for a particular election they also get their weight and the public keys of all Administrators bound to the weight or, otherwise, all the $N \times W$ public keys used by Counters.

The knowledge of weights bound to keys does not provide information on specific voters; and the knowledge of a voter's weight is official knowledge. This way, except in special circumstances, privacy is ensured.

4.1 Changes in the protocol

The changes in the protocol are minor. For each election we need to define the number W of possible weights and their value. For each allowed weight there will be an associated asymmetric key pair per Administrator. The Commissioner establishes a mapping between each weight and each public key of each Administrator (hereafter referred to as the **WKpub table**). This table is sent to the Counters for the tally process.

The process of registering voters must be enriched to accommodate the specification of one weight per voter. The mapping between voters and weights must then be sent to Administrators; they need it for choosing the correct private key for signing a voter's vote according to his weight.

In REVS, voters get the public keys of all Administrators on the voting bulletin, for validating the signatures provided by the latter. When weights are considered, for distributing the right information to voters on voting bulletin two solutions are possible:

1. Distribute a generic bulletin to all the voters, a bulletin containing the WKpub table. This simplifies the task of the Distributor but the size of the WKpub table may be a problem for performance.
2. Distribute customized bulletins to voters, i.e., bulletins with the public keys of Administrators corresponding to the voters' weights. In this case, the Commissioner has to produce W different, signed bulletins and the Ballot Distributors must use the mapping between voters and weights for giving voters the right bulletins. This complicates the task of Ballot Distributors but keeps the current size of bulletins.

In both cases, the voter get its weight when it gets blind signatures from Administrators. In the first case that helps him to select the proper public keys from the WKpub table for validating the Administrators' signatures. In both cases, it allows the voting application to present to the voter the weight that is being considered for his vote.

If all the voting weights got from Administrators are not the same, the voter must complain for a configuration problem. Otherwise, it propagates its weight to the ballot submission package sent to Counters. The propagated weight is not mandatory for the final tally, it is only for accelerating the counting process; even if wrong, the vote can still be counted, though with more computational overhead for Counters.

At the end of an election, the votes submitted to Counters are validated. This means deciphering them with the election's private key (disclosed by the Commissioner) and checking the Administrators' signatures. A vote is valid if it is correctly signed by at least t Administrators and all of them used their private key for the same weight. Thus, a votes' weight can be directly inferred from the Administrator's signatures on it and if the vote is valid, then it counts as many times as its weight. Counters may use the weight in a received voting package as a hint for getting the right set of Administrators' public keys. But, in case of failure, Counters try other sets, corresponding to other weights.

4.2 Optimizations

The number of Administrator's asymmetric key pairs grows proportionally with the number W of weights. This as an impact on scalability, both in the generation of $W \times N$ key pairs, required by all Administrators, and in the WKpub table used by Counters and possibly distributed in bulletins. However, the production of multiple RSA signing keys has an opportunity to minimize this scalability issue. Namely, the process for creating W RSA keys for a single entity can create the modulus, n, once and latter use the same n for making all RSA keys.

The optimized generation process runs as follows. The first key pair is calculated with the original behavior of the RSA algorithm, where n is the modulus, e is the public exponent and d is the private exponent [3]:

1. Generate two large random primes, p and q, of approximately equal size such that their product $n = pq$ is of the required bit length, e.g. 1024 bits, and compute $n = p\,q$ and $\phi(n) = (p-1)\,(q-1)$.
2. Choose a (small) integer e, $1 < e < \phi(n)$, such that $\gcd(e, \phi(n)) = 1$ and compute the secret exponent d, $1 < d < \phi(n)$, such that $e\,d \equiv 1 \pmod{\phi(n)}$.
3. The public key is (e, n) and the private key is (d, n).

The other $W - 1$ key pairs with the same n are the computed as follows:

1. Choose a (small) integer e', $e < e' < \phi(n)$, such that $\gcd(e', \phi(n)) = 1$. For preventing the deduction of straightforward relationships between d values, e' should also be coprime to all other e values computed for the same n.
2. Compute the secret exponent d', such that $e'\,d' \equiv 1 \pmod{\phi(n)}$.
3. The public key is (e', n) and the private key is (d', n).

Using the same modulus n for all key pairs of each Administrator has several advantages. First, we save computational time and space when computing the key pairs — only two large prime values p and q need to be generated per Administrator. Second, we save memory and bandwidth for storing and transmitting the WKpub table.

This optimization cannot be extended to encompass all Administrators´ key pairs, because when we have several d values for the same modulus n, all d values must be known only by the key pair owner. Otherwise, the owner of one d value could discover all other d values using the common modulus attack [10, 11].

4.3 Computing blind signatures using different key pairs

The voter's application only gets the voter's weight in the replies of Administrators. But this raises a problem, because the Chaum's RSA-based blind signature scheme used in REVS requires the voter knowing in advance the key pairs that Administrators will use (thus, knowing in advance the voter's weight):

1. The voter gets a random k and computes k^{-1} such that $k \cdot k^{-1} \equiv 1 \pmod{n}$.
2. The voter blinds the vote digest $h(v)$ by computing $k^e \cdot h(v) \bmod n$, and sends the result, $h'(v)$, to the Administrator.
3. The Administrator computes $h'(v)^d \bmod n$ and sends the result to the voter.
4. The voter computes $k^{-1} \cdot h'(v)^d \bmod n$ and gets $h(v)^d \bmod n$.

As we can clearly see, in steps 1 and 2 the voter needs to know the public components e and n of an Administrator key pair for blinding the vote digest.

We handled this problem by changing the blind signature computations and not the protocols messages. Given the fact that all key pairs of an Administrator have the same modulus n, then the computations are done as follows:

1. The voter gets a random k and computes k^{-1} such that $k \cdot k^{-1} \equiv 1 \pmod{n}$.
2. The voter blinds $h(v)$ by computing $k^{\prod_i^W e_i} \cdot h(v) \bmod n$, where i represents a weight, e_i the public key of the Administrator for that weight and W the number of weights; the result, $h'(v)$, is sent to the Administrator.
3. The Administrator computes $h'(v)^{d_w} \bmod n$, where d_w is the private key for the weight w of the voter, and sends the result and w to the voter.
4. The voter computes $k^{-1} \cdot k^{-\prod_{i \neq w}^W e_i} \cdot h'(v)^{d_w} \bmod n$ and gets $h(v)^{d_w} \bmod n$, the intended result.

When we have a single weight ($W = 1$), it is easy to see that these calculations are exactly the same that were performed in the original REVS system.

5 Security evaluations

The security of REVS was already evaluated and discussed when it was proposed (c.f. [2, 5]). Therefore, here we will mainly discuss the security of the upgrades introduced in order to support vote weights.

The introduction of weight capabilities adds three extra requirements to the voting system: (i) it is not possible for a vote weight to be altered; (ii) it ensures

that eligible voters use their correct weight; and (iii) neither authorities nor anyone else can use vote weights to link any ballot to the voter who cast it.

Regarding the first requirement, a vote cannot be altered because that requires changing t signatures, just like in the original REVS. Since the weight is embedded in the signatures, changing a vote weight requires changing all the signatures on a vote. A set of t colluded Administrators can do it, but mainly in theory, since that requires access to all the copies of that vote stored in the Counters. Therefore, it requires a larger collusion, involving all t Administrators and an undetermined, arbitrarily large percentage of Counters.

Regarding the second requirement, vote weights are forced by Administrators and not arbitrarily chosen by voters. Thus, Administrators manage proper weights just like they manage the list of eligible voters. And single Administrators cannot cheat, because that would make them disagree between each other.

Regarding the third requirement, there are privacy issues. In fact, if a single voter is entitled to a specific weight, or if a single voter uses a specific weight, then anyone with access to the authorization databases used by Administrators can link the vote to the voter. Nevertheless, current paper-based voting processes do not handle this problem either. Unless a voter could break its weighted vote in many unitary votes, which may be tedious or infeasible for large weights, the voter must use a different bulletin or cast the vote in a differentiated ballot box, thus raising the same privacy problem. Concluding, our solution is not worse than existing paper-based solutions, mainly when large weights are considered. Solving this problem and keeping the system scalable remains an open issue.

Finally, we need to discuss the security implications of sharing a common modulus by all the RSA key pairs of each Administrator. There are some known problems in using the same modulus for different RSA key pairs [11, pag. 289] but in our case they do not apply:

- The private component of the keys pairs with the same modulus must be known only by a single entity. Otherwise, an entity knowing a private value d could compute other private values d' used by other entities (all of them sharing the same modulus n, of course) [10]. In our case, this is not a problem: each Administrator chooses an n and computes W different (e_w, d_w) pairs for it. At the end, all (e_w, n) pairs are published as the Administrator's public keys and the Administrator keeps privately all the d_w values.
- The same message m encrypted with two RSA public keys sharing the same modulus n can be decrypted without knowing the private keys [12]. Again, in our case this is not a problem because we do not use the RSA key pairs for confidentiality, but for signing votes. Thus, we do not encrypt with public components, we only encrypt with private components.

Concluding, in terms of security our solution is capable of enforcing a correct use of vote weights but presents some privacy issues that may appear when small sets of voters share the same weight. Furthermore, the shared modulus optimization for the RSA signing keys does not, as far as we can see, introduce new vulnerabilities in the voting protocol. Namely, all the known shared modulus attacks do not apply to our protocol.

6 Implementation

REVS is fully implemented in Java and uses MySQL databases to store configuration information used by REVS servers. These databases are also used to store information produced and/or gathered by servers, such as signatures provided by Administrators and voting submission packages sent to Counters.

In this section we briefly describe the changes required in REVS in order to support our solution for handling vote weights. The changes are minor, since the basic protocol was mainly left unchanged, and where all implemented.

Commissioner. At the Commissioner, new options to manage weights were added to the graphical interface. The number of weights W is entered first and thereafter the valid weights and the weight for each voter. The data of allowed weights is then to be sent to the Administrators, which required only a minor addition in handling the exchange of data between Commissioner and Administrator. There where also additions in the exchange of data from the Commissioner to the Administrators and to the Counters to include the WKpub table.

Administrators. The information of allowed weights from the Commissioner is used as a template to make the signing keys for each Administrator. An entirely new function was written for the making of optimized RSA key pairs, since no such function was available in Java libraries. The private component of each key pair is saved in a database table with the corresponding weight (**WKpri table**), which was added in the Administrators database.

When a vote arrives, the weight of the voter is checked in the voters table. The weight is then used to check, in the WKpri table, which signing key to use for the given weight and the vote is signed with it.

Ballot distributors. Two solutions where discussed for distributing information to voters, in Section 4.1. We do not consider the WKpub table to be of a large enough size to cause any performance problems for this work, so we chose the first solution; to attach the WKpub table to the bulletin and to use only one bulletin for all voters participating in an election.

The solution we chose leaves the Ballot Distributors without any knowledge of the voters' weights and keeps them fast and simple. It also provides less changes needed to the system.

Voter's module. The voter's module was changed to deal with the WKpub table, that is now part of the bulletin. The blind signing algorithm was also changed to the new one presented in Section 4.3, the weights replied by Administrators are checked (they must be all equal) and presented to the voter in the graphical interface. Finally, the vote weight is propagated in the submission package sent to Counters for accelerating counting procedures.

Counters. In the validation of each vote was added code to check the weight of the vote, forwarded by the voter in the submission package, and to get the corresponding public keys from the WKpub table. This validation includes the risk of the weight being altered by the voter. The counting of valid votes only needed the minor addition of counting the votes as many times as the weight says, instead of only counting them once.

7 Conclusions and future work

In this paper, we evaluated different possibilities to make a support for weighted votes in the electronic voting system REVS and described the best solution for the system. Namely, our solution relies on using sets of signing key pairs to represent the different weights allowed in an election. We also presented a performance enhancing solution for making the, sometimes large, set of RSA signing key pairs from the same modulus.

To the best of our knowledge, this is the first electronic voting system supporting votes weights. The same happens with the optimization used for the W RSA key pairs for each Administrator, we do not know of any system using it.

Considering our requirements — scalability, efficiency, usability and anonymity —, the first three of them were attained. Regarding anonymity, there is still a problem for the case of a single voter with a weighted vote. This is a fundamental problem, since it also exists in paper-based elections. We tried to find a solution to this problem without interfering with the basic characteristics of REVS but at the end we rejected all the ideas of how to deal with the problem in this work. Consequently, and in the context of REVS, our main issue for future work is to deal properly with this privacy issue.

References

1. L. Cranor and R. Cytron. Sensus: A security-conscious electronic polling system for the Internet. In *Proc. of the Hawaii Int. Conf. on System Sciences*, Wailea, Hawaii, USA, 1997.
2. Rui Joaquim, André Zúquete, and Paulo Ferreira. REVS – A Robust Electronic Voting System. *IADIS Int. Journal of WWW/Internet*, 1(2), December 2003.
3. R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Comm. of the ACM*, 21(2), February 1978.
4. David Chaum. Blind signature system. In *Advances in Cryptology – CRYPTO '83 Proc.*, pages 153–153, New York, USA, 1984. Plenum Press.
5. Ricardo Lebre, Rui Joaquim, André Zúquete, and Paulo Ferreira. Internet Voting: Improving Resistance to Malicious Servers. In *IADIS Int. Conf. Applied Computing 2004*, Lisboa, Portugal, March 2004.
6. Rui Joaquim. A fault tolerant voting system for the internet. Master's thesis, IST / UTL, February 2005.
7. Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *Advances in Cryptology – AUSCRYPT '92 Proc. (LNCS 718)*, Queensland, Australia, 1992. Springer-Verlag.
8. M. Herschberg. Secure Electronic Voting Using the World Wide Web. Master's thesis, MIT, 1997.
9. B. DuRette. Multiple Administrators for Electronic Voting. Bs.C thesis, 1999.
10. Gustavus J. Simmons. A "weak" privacy protocol using the RSA crypto algorithm. *Cryptologia*, 7(2):180–182, 1983.
11. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. 5th Printing.
12. J. M. DeLaurentis. A further weakness in the common modulus protocol for the RSA cryptoalgorithm. *Cryptologia*, 8(3), 1984.