

Enabling the Application of Open Systems like PCs for Online Voting

Melanie Volkamer¹, Ammar Alkassar², Ahmad-Reza Sadeghi³, Stefan Schulz³

¹ German Research Center for Artificial Intelligence volkamer@dfki.de

² Sirrix AG alkassar@sirrix.de

³ Ruhr-Universität Bochum {sadeghi, schulz}@crypto.ruhr-uni-bochum.de

Abstract. An increasing number of elections have been performed using Online Voting. Due to these experiences, the technical research topics have changed: While voting protocols have been well analysed in the past, now, the security and trustworthiness of the client platforms come to the fore. Malware could cause arbitrary damages regardless how secure the underlying voting scheme is. Moreover, corrupt voters can overcome the receipt-freeness of many protocols by manipulating their voting client (PC). Currently, the client side is mainly protected by organizational measures imposed on the voters, largely ignoring the issue of corrupt voters. However, since elections are the very basis of democracy, verifiable technical solutions are essential. Therefore, we examine the necessary properties of a trustworthy client which both protect the voters from malware and the voting system from corrupt voters. Our approach is based on Trusted Computing in combination with a secure operating system. We show that other existing voting protocols can be implemented on the top of our approach as well as a significantly simplified voting scheme by still retaining the required properties.

1 Introduction

Recent remote online elections show that Online Voting is not only theoretically discussed, but also applied in practise. Examples are the Estonian Local Government Council Election, the elections of the Gesellschaft für Informatik in Germany (GI), the online referendum in Switzerland, and the 2006 ACM SIG election. Additionally, countries like the Netherlands and Austria plan to introduce Online Voting for people living abroad for the next national elections⁴. Nevertheless, people are still cautious about the introduction of Online Voting and we are far away from an employment in large.

Online Voting System can be attacked in various ways: The attacker can either attack the end-user device, the communication or the voting servers. Let us assume that well established security functionalities are in place to prevent attacks to the communication and to the voting servers, because attacks to the communication are prevented by well known and analyzed voting protocols

⁴ Information about all these elections can be found in [8].

and the voting servers are protected from intruders over the network by using state-of-the-art security mechanisms installed and configured by administrators (firewalls, virus scanners, etc.). Moreover, physical access for intruders to the voting servers can be prevented using, e.g., access control and secret sharing mechanisms.

Our concern in this paper is the voter's PC or in general end-user devices. What kind of manipulation is possible by malware which is not detected by the voter or even malware which is installed on purpose by a corrupt voter? Do assumptions made by voting protocols about the client computing environment hold in practise? Obviously they do not, because the voter's devices in Online Voting scenarios are open systems like a PC or PDA. In general, voters are not able to protect themselves efficiently against malware. Likewise, we cannot assume that a malicious voter do not manipulate his device on purpose, since a platform owner has complete control over it.

In this paper we examine the security and trustworthiness of the end user device and discuss the application of Trusted Computing in combination with a secure operating system to support the required security and trustworthiness of the client. We argue that, in contrast to current situations Online Voting can be done much more secure and also effeciently using this technologies.

Our focus in rimarily are security issues whereas usability aspects, while certainly important, play a lesser role. A welcome side effect of applying Trusted Platforms for Online Voting is the possibility to simplify the voting protocol enormously to make it easier to understand and to verify. However, it is also possible to use existing protocols on the top of our approach, thus existing systems can still be used with minor changes.

The paper is organized as follows: In Section 2, we discuss common vulnerabilities, and define security requirements. Existing and related work is proposed and discussed in Section 3. We discuss the application of Trusted Computing in combination with a secure operating system to produce a secure and trustworthy voting client in Section 4. The implementation issues for a simplified voting protocol as well as TC as an extension to existing Online Voting System can be found in Section 5 and we conclude in Section 6.

2 Requirements and Vulnerabilities

The main technical requirements to an Online Voting System are the following two (see [10] and [13] for a more detailed list):

1. Votes must not be added, removed, or altered 'undetected'.
2. Only the voter is aware of his voting decision. Nobody else is able to link the voter to his vote and the voter has no possibility to prove his decision to a third party (receipt freeness).

Theses properties must hold for the communication, the voting client and the voting server. We will analyze only the voting clients: Currently, electronic voting devices are already used in polling stations because it can be verified that

these machines ensure both requirements. These devices are closed box systems which are only used for elections and which have no additional functionality. The security and trustworthiness of these devices is justified by their comparatively small complexity, which enables an evaluation of the whole device and not only the voting software, and the tamper-resistant design of these devices. Unfortunately, closed-box voting devices are out of the question for remote online elections because of enormous per-item costs. Thus, remote Online Voting has to work on deployed hardware and software on open systems like PCs and PDAs. In order to use these devices for voting we have to achieve closed-box level security on these open systems to fulfill the requirements.

Vulnerabilities. Currently, devices like PCs and PDAs are threatened mainly by three categories of attacks:

1. The voting software is prone to manipulation by malicious code on the voting device; e.g., viruses and Trojan horses.
2. The voting software is manipulated by a corrupted voter. This is possible because the device is usually completely under the control of the user/voter.
3. In addition, the attacker could disseminate manipulated voting software.

While the third group of attacks can be prevented by conventional means, e.g., by the application of software certificates, the first two attacks cannot be prevented by traditional means, yet. We will now have a closer look to these two types of attacks.

2.1 Malicious Code on Voting Devices

Malicious code, running on the voter's client lowers the security of any voting system to absurdity: Plenty of attacks are imaginable: (i) altering the voter's decision without the voter noticing it, (ii) preventing voters from the election, by throwing away the vote and making the voter believe that everything works correctly, e.g., by displaying faked messages, (iii) extracting the voter's decision and transmitting the plaintext vote to the attacker.

Common cryptographic means do not overcome any of these attacks, since malicious code interacts before the cryptographic operations are applied. The adversary may, for instance, eavesdrop on mouse or keyboard inputs and deduces the voter's decision. Thus, both requirements can be violated by malicious code on the voting device.

This is a strong problem because malicious code can be distributed easily and automatically, e.g., by exploiting security flaws of the end-user device or by sending infected emails to the voter. This can be done massively via viruses. Malicious code could also be put on the voter's device by developers of products running on many end user devices (e.g., Solitaire). Compared to postal voting, these attacks can be done automatically and in a scalable way with significant impact on the election result.

2.2 Corrupt Voter

Corrupt voters may alter the voting software on their device and/or their device in arbitrary ways: They can, e.g., log their mouse and keyboard events, any other I/O interfaces like the one to connect to the network. Moreover, they are able to store each intermediate computation step. Thus, having a probabilistic encryption function enc_P and a plaintext m as well as the corresponding cipher text c after computing $c = enc_P(m)$, the voter has logged all random parameters to be able to prove that c belongs to m . The most serious problem by doing so, is the generation of a proof for the voting decision which would violate the receipt freeness and thus, allow vote selling.

Receipt-freeness is a well known problem. While many receipt-free voting protocols already exist (see, e.g., [6] and [9]), most of them are based on assumptions⁵ with respect to some parameters (e.g., random numbers for probabilistic encryption) which have to be kept secret on the client side. This assumption does not hold in practise since platform owners have complete control over the voting device - the PC.

It is not enough to sell m , c and $proof(m, c)$. Someone buying ballots, the customer, might also want to know whether the shown c is the one that is cast and counted or one created it in order to cheat. Therefore this proving-attack works quite well for any voting protocol working with a bulletin board where the encrypted vote can be somehow assigned to the voter. Therefore, a corrupt voter can sell his logged data (e.g., vote, parameters, and keys) and the customer of the vote verifies whether the corresponding vote is on the bulletin board or not. This attack can be supported by the application of voter verifiability. If the voter has the possibility to verify that his vote is stored and counted. This proof can also be used by the customer to check whether the voter behaved as promised.

Notice, it is quite a lot of effort for single voters to get the necessary data out of the voting software just to sell a single vote. However, most customers will wish to buy a large number of votes anyway, and thus provide the corrupt voters with ready-made tools (which conduct the necessary logging and reporting automatically), which they only have to install and use.

Summery. We can conclude, that any voting scheme has to consider the client part as an *untrusted* part of the voting system, which limits the usage of many simpler voting schemes.

3 Related Work

Online Voting projects like in Switzerland [1] or the one of the GI [5] have also noticed the client weakness. Thus, they give handouts to the voters which explain voters how to improve the trustworthiness of their PCs. While this approach can

⁵ Receipt free protocols which are not based on these assumptions are, e.g., [7] and [16]

reduce the risks created by malware, many users will likely not be able to follow these instructions, and the technique is useless against corrupt voters.

Otten has also addressed the client problems and thus, proposed in [3] a special voting operating system based on Knoppix. Here, voters have to boot their PC from CD in order to vote. This approach does not solve the corrupt voter problem but it addresses the threats caused by malware.

Other work like [7] and [16] proposed the application of an observer, e.g., a smartcard. By doing so they overcome the attacks of a corrupt voter mostly but a smartcard does not interact directly with the voting server but over the end-user device and thus the open system. Malware on the end-user device can mount a man-in-the-middle attack and misuse the card, e.g., by sending a wrong vote to the card in order to encrypt and sign it or the PC displays the wrong ballot.

Already in 2002, Avin Rubin discussed the security considerations for remote electronic voting in public elections [14] and find several flaws caused by malware within the application of PCs as end-user devices for Online Voting. He already pointed out that hardware support to enable a trusted path between the user and the election server is necessary to overcome the threats.

4 Application of Trusted Platforms in Online Voting

In this section we shortly discuss a security architecture that can be used for secure online voting using open systems like PCs. This is in contrast to today's used methods.

The current research is looking for a *Trusted Platform* which includes the desirable properties of open platforms (e.g., PC's and PDA's) and of closed platforms (voting device, smartcards). Thus, it is not limited in its functionality like closed platforms but it is able to prove to a third party, that it is well behaved, like closed platforms. A trusted platform is based on *Trusted Computing* hardware and on a *secure operating system*.

Trusted Computing (TC). TC refers to a technology advocated, developed and promoted by the Trusted Computing Group (TCG). The core component of TC is the Trusted Platform Module (TPM), a secure and tamper-evident module (often a single chip) integrated into the platform. TPMs are thought of as not being costly, to the point of easily becoming a standard part of off-the-shelf PC platforms. About 50 million PCs with TPM chips have been shipped, and predictions for 2010 range around the 250 million mark⁶. They provide several essential functionalities:

- *Useful Cryptographic Primitives* like a secure random number generator in hardware.
- *Restricted Usage Keys* which can only be accessed in certain limited ways, such as being accessible only in one particular platform state, and not being migratable out of the TPM.

⁶ See https://www.trustedcomputinggroup.org/news/newsletter/2005/2005_July

- *(Remote) Attestation* to remotely authenticate (the configuration of) a platform. For this purpose, a trusted boot creating a chain of trust is essential. The trusted boot logs the boot sequence starting with the Core Root of Trust Measurement (CRTM) in protected registers of the TPM (called Platform Configuration Register - PCR). These PCR values are signed by a key called Attestation Identity Keys (AIK). The AIK is generated by the TPM and the public key is either certified by a Trusted Third Party or without by using the protocol developed by Brickell, Camenisch and Chen in [2]. In the second case no information about the TPM or its owner is given away.
- *Sealing* to cryptographically bind data to a specific platform, a specific platform configuration and the identifier of the invoking application. Sealing may be facilitated using restricted keys. Therefore, a key pair is generated and the corresponding secret key is bound to the current platform, platform configuration and application. Now, a remote instance can encrypt data including a demanded configuration using the corresponding public key. The TPM can decrypt the cipher but releases the data only if the current platform configuration matches the demanded one.

While the TPM itself is a passive component, the primitives it provides can, with secure operating systems support, be used to significantly improve security on end-user devices.

Secure Operating System. The operating system controls the information flow above the hardware layer and thus has access to all kind of data - including security critical data. Thus, a secure operating system is essential to protect security-critical applications from each other and from malicious code using security properties like *process isolation* and a *trusted path*.

Trusted Platform. A trusted platform is based on a secure operating system and the TPM. A security platform consists of a minimalized security kernel - the secure operating system. Any conventional operating system as well as security critical applications like the voting application can be executed on the base of the secure operating system. An example for such an architecture is the PERSEUS security architecture [11] (compare Figure 1 for a simplified application).

4.1 Secure and Trustworthy Voting Clients

In this section, we show which parts of a trusted platform can be used to protect against the identified attacks from Section 2.

Malicious Code on the Clients. The process isolation of the secure operating system is essential to prevent malicious code from accessing security-critical applications on the end-user devices in general. While malicious software might be on the device, it cannot influence or interact with the voting software. In addition, data displayed on the screen and data from the mouse and the keyboard can be neither read nor altered because of the trusted path between user

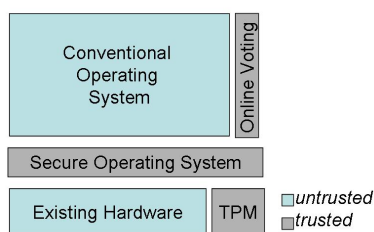


Fig. 1. Example for a trusted platform

and application. Thus it is not possible that a faked message is displayed on the screen to fool the voter into accepting a faked vote casting process. Moreover, it is not possible to eavesdrop the communication between the user and his system in order to deduce the voter's decision.

To ensure that incoming messages from the voting server are not read by malware or in general other applications, the voting server can seal the message and thereby bind it to the correct voting application on the proper platform.

To prevent malicious code to read or alter data from a previous voting phase, this data can also be encrypted based on sealing. Thus, only the authentic voting software running on the proper system can read the data.

Corrupt Voter. To prevent voters from manipulating the voting device or the voting software remote attestation is essential. Thereby, the whole system configuration, including the operating system and the voting software, is measured and proven to the voting server. The server only accepts a message and communicates with the voter if the voter's voting software is authentic and runs on a trusted platform. The voter can still install malicious software on any virtual machine but here he cannot vote because the voting server will not accept his ballot because of the wrong values in the PCRs (remote attestation can be achieved by sealing the messages to the configuration of the voting application and underlying TCB). Those malware cannot influence or eavesdrop the voting software running in the voting virtual machine because of the process isolation.

Dissemination of Manipulated Voting Software. In general, the voter downloads the voting software (the whole voting virtual machine). Currently, some additional software computes the voting software's hash value and the voter verifies the value or the voter checks whether the voting software is correctly signed. However, currently, malware running on the platform can manipulate this security check and the display, as well. This cannot happen on the trusted platform because of the trusted path.

In summary, a trusted platform overcomes the identified open problems of Online Voting and it provides more security for the last attack where partial

solutions already exist. Thus, a trusted platform on the voting clients and server is a solid foundation for a secure and trustworthy Online Voting System.

5 Possible Implementations

5.1 A Simple Voting Scheme

Currently, the voting protocols are quite complex, because the voter has to be identified and at the same time the election secrecy must be ensured. Protocols are based on almost all kinds of cryptographic primitives like, e.g., homomorphic encryption (e.g., [15]) and blind signature (e.g., [12, 4]). In general, only few voters understand most of the proposed voting protocols but for the average voter it is a mystic black box. This is one reason why research introduced voter verifiability but this makes protocols even more complicated. The application of trusted platforms on the client and the server side allows for simplifying the protocol to encryption and signing because we only need to prevent attacks on the network because we trust that the client and server voting software is verified and certified. Moreover, this supersedes voter verifiability.

A protocol (with a simplified notation) would, e.g., be the following one: There are two phases. In the first one the right to vote is checked by one voting server ($server_1$) and in the second one the vote is cast and therefore sent to a second voting server ($server_2$).

Registration Phase. It starts with a client and server side remote attestation including an exchange of public sealing keys:

$$(1) \text{ Voter} \rightarrow \text{Server}_1 : \text{sign}_{AIK}^{\text{Voter}} (PCR_{S_{\text{Voter}}}, PK_{\text{Sealing}}^{\text{Voter}})$$

Notation: This message denotes the remote attestation protocol, where at the end the server knows the platform configuration of the voter (because of $PCR_{S_{\text{Voter}}}$) and the public sealing key $PK_{\text{Sealing}}^{\text{Voter}}$ it has to use to communicate with the voter and only with the voter.

$$(2) \text{ Server}_1 \rightarrow \text{Voter} : \text{sign}_{AIK}^{\text{Server}_1} (PCR_{S_{\text{Server}_1}}, PK_{\text{Sealing}}^{\text{Server}_1})$$

Now, the main voting protocol starts:

$$(3) \text{ Voter} \rightarrow \text{Server}_1 : \text{sign}_{AIK}^{\text{Voter}} (enc_{\text{Sealing}}^{\text{Server}_1} (ID, PIN))$$

The voting server checks by verifying the signature, whether the voter works with the authentic voting software on the trusted platform. The server itself can only encrypt the message if the authentic server side voting software runs on the trusted platform. Now the voting server checks whether the voter is an eligible voter (using ID and PIN code) and replays with

$$(4) \text{ Server}_1 \rightarrow \text{Voter} : \text{sign}_{AIK}^{\text{Server}_1} (enc_{\text{Sealing}}^{\text{Voter}} (ID, ok))$$

The voter's voting software checks by verifying the signature, whether the answer was sent from the authentic voting software running on a trusted platform. In addition, only the authentic client side voting software running on a trusted platform is able to decrypt this incoming message. Only if there is an "ok" sent in step (4), the voting software displays the ballot. Thus, the Registration Phase is finished.

Vote Casting Phase. Knowing the voter has voting-rights the voting software displays the ballot. Now the voter can choose candidates and acknowledge his decision. Then, the voting software first informs $Server_1$ that the voter has cast a ballot.

$$(5) \text{ Voter} \rightarrow \text{Server}_1 : \text{sign}_{AIK}^{\text{Voter}} (\text{enc}_{Sealing}^{\text{Server}_1} (ID, "voted"))$$

The voting server checks whether the authentic voting software is still running on the trusted platform. Now the voting server ticks off the corresponding voter in the electoral register and replays with

$$(6) \text{ Server}_1 \rightarrow \text{Voter} : \text{sign}_{AIK}^{\text{Server}_1} (\text{enc}_{Sealing}^{\text{Voter}} (ID, "flagged"))$$

Now, it starts again with a client and server side remote attestation including an exchange of public sealing keys (different keys as above). In this phase the remote attestation is based on the [2] protocol. Thus, neither the servers nor any network sniffer⁷ does know which voter or even which TPM sends the encrypted vote because no voter or even TPM information are available.

$$(7) \text{ Voter} \rightarrow \text{Server}_2 : \text{sign}_{AIK'}^{\text{Voter}} (PCR_{sVoter}, PK_{Sealing'}^{\text{Voter}})$$

Notation: This message denotes the remote attestation protocol, where at the end the server knows the platform configuration of the voter (because of PCR_{sVoter}) and the public sealing key $PK_{Sealing'}^{\text{Voter}}$ it has to use to communicate with the voter and only with the voter.

$$(8) \text{ Server}_2 \rightarrow \text{Voter} : \text{sign}_{AIK}^{\text{Server}_2} (PCR_{sServer_2}, PK_{Sealing}^{\text{Server}_2})$$

Now, the main voting protocol starts again:

$$(9) \text{ Voter} \rightarrow \text{Server}_2 : \text{sign}_{AIK'}^{\text{Voter}} (\text{enc}_{Sealing}^{\text{Server}_2} (vote))$$

The server accepts a vote-message if it is generated and signed by the authentic voting software. Consider, the server does not need any other mechanism to verify authentic votes, because it trusts that an authentic voting software running on a trusted platform only sends a vote-message if the voter has the right to vote. Thus, the essential property that only authorized voters can cast a ballot is enforced mainly by the client voting software. Therefore it is important to evaluate and certify the software. In addition, the server voting software has to

⁷ For the network sniffer this does only holds if the voter has different IP addresses in both phases. Otherwise he can use the IP to link all protocol messages to one specific voter.

ensure that the ballot message is sent from a proper system running the authentic client voting software (thus, the steps before took place). In addition it has to be ensured that replay ballot messages are detected.

Finally, the server sends a confirmation:

$$(10) \text{ Server}_2 \rightarrow \text{Voter} : \text{sign}_{AIK}^{\text{Server}_2} (\text{enc}_{\text{Sealing}'}^{\text{Voter}} (ok))$$

The voter's voting software verifies the message and displays the voter that his vote is stored successful.

At the end of the Election Day the votes stored on the voting server are decrypted and counted. The encrypted and decrypted votes can be made public. Thus, everybody can compute the result on his own.

5.2 Analysis

We will now see, whether the proposed protocol fulfils the requirements defined in chapter 2 for the client, the server and the communication. Assuming the applied client and server side software is verified, it is enough to show, that the requirements hold for the communication and that a voter cannot cast more than one vote. Obviously, each voter can only cast one vote, because after choosing candidates and acknowledging his choice, the voter lose his right to vote and Server_1 does not accept a request from this voter anymore. The election secrecy requirement is ensured on the network because all messages are encrypted and corresponding decryption keys are kept secret by the voting client and server respectively. Moreover, it is not possible to create new protocol messages because it is not possible to either fake a signature or manipulate a voting client in order to fake signatures. It is also not possible to replay an old message, because the voting servers and client servers check the freshness of the messages and the applied keys respectively.

Future work is to have a closer look to such simplified protocols especially with respect to system break downs and a verification of such a protocol.

5.3 Application to Existing Systems

In this section, we examine systems based on SSL and a web-browser, such as Polyas (used for the GI elections) and the system used in the Swiss elections. These protocols first establish an SSL connection to the voting server (or to two successive servers, a voter registry and a ballot server). We assume each party has an active TPM chip.

Wrt. the current point in research we could distribut a CD to boot a secure platform.

This CD contains a GUI capable of providing a trusted path, a web browser and a secure VPN module. All communication is tunneled via IPSec, with all non-IPSec traffic rejected by the voting server without consideration. Before the actual voting protocol (and in fact, SSL key exchange) is executed, both sides attest one another.

Of course, limiting voters to the use of one CD-based solution and using attestation has the side effect of locking out voters using platforms without a TPM. Also, some voters might feel their system to be “secure enough”, and be uncomfortable with having to reboot from the CD. Neither should be easily dismissed, but would merit further consideration.

6 Conclusion

We illustrated that the voter’s device is still a weak point of any Online Voting System based on open platforms. Attacking the client enables the breaking of the election secrecy as well as the manipulation of the election outcome by altering votes on the voter’s device. Moreover, we showed that receipt-free voting protocols and voter verifiability does not really help to solve these problems. Nevertheless, we believe that Online Voting will increasingly applied in future because of the increasing mobility of the voters. To increase the security and to turn a multi-purpose open system, like the voter’s PC is one, into a secure and trustworthy system, we propose to use an appropriate security architecture based on a security kernel and Trusted Computing. Even if we know that currently there are still open problems with Trusted Computing and a lot of work to do with secure operating systems, we believe that Online Voting should not be applied at least in large for any parliamentary elections without a trustworthy computing platform.

Moreover, by using the functionality of trustworthy platforms, we can simplify the voting protocols while also being able to use any other security critical application (like homebanking).

References

1. Nadja Braun and Daniel Brändli. Swiss e-voting pilot projects: Evaluation, situation analysis and how to proceed. In Krimmer [8].
2. Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145, New York, NY, USA, 2004. ACM Press.
3. Dieter Otten. Mehr Demokratie durch Internetwahlen? Vortrag gehalten im Nixdorf Forum in Paderborn 15-2-2006, 2005.
4. Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251, London, UK, 1993. Springer-Verlag.
5. Rüdiger Grimm, Robert Krimmer, Nils Meißner, Kai Reinhard, Melanie Volkamer, and Marcel Weinand. Security requirements for non-political internet voting. In Krimmer [8].
6. Martin Hirt and Kazuo Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology - EUROCRYPT 00*, volume 1807 of *Lecture Notes in Computer Science*, pages 539–556. Springer-Verlag, 2000.

7. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 61–70, New York, NY, USA, 2005. ACM Press.
8. Robert Krimmer, editor. *Electronic Voting 2006, Austria, Proceedings*, volume 86 of *LNI*. GI, 2006.
9. Byoungcheon Lee, Colin Boyd, Ed Dawson, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. Providing receipt-freeness in mixnet-based voting protocols. In *ICISC*, volume 2971 of *Lecture Notes in Computer Science*, pages 245–258. Springer-Verlag, 2003.
10. Council of Europe. Legal, operational and technical standards for e-voting. recommendation rec(2004)11 adopted by the committee of ministers of the council of europe and explanatory memorandum. *Council of Europe, Straßburg*, 2004.
11. Birgit Pfitzmann, James Riordan, Christian Stubble, Michael Waidner, and Arnd Weber. The PERSEUS system architecture. In Dirk Fox, Marit Köhntopp, and Andreas Pfitzmann, editors, *VIS 2001, Sicherheit in komplexen IT-Infrastrukturen*, pages 1–18. Vieweg Verlag, 2001.
12. Alexander Prosser and Robert Müller-Török. E-Democracy: Eine neue Qualität im demokratischen Entscheidungsprozess. *Wirtschaftsinformatik*, 6:545–556, 2002.
13. Physikalisch-Technische Bundesanstalt Braunschweig/Berlin PTB. Online Voting Systems for Nonparliamentary Elections - Catalogue of Requirements. http://www.berlin.ptb.de/8/85/LB8_5_2004_1AnfKat.pdf retrieved on 15-2-2006, 8.5.2004.
14. Aviel D. Rubin. Security considerations for remote electronic voting. *Commun. ACM*, 45(12):39–44, 2002.
15. Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. *Lecture Notes in Computer Science*, 1666:148–164, 1999.
16. Joern Schweisgut. Coercion-resistant electronic elections with observer. In Krimmer [8], pages 171–177.