Information Security in a Campus Computing Environment

by

Mark Robert Will

B.Sc., University of Victoria, 1990

A Thesis Submitted in Partial Fulfilment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

[redacted]

Dr. E. Manning, Supervisor (Department of Computer Science)

[redacted]

Dr. G. Shoja, Departmental Member (Department of Computer Science)

[redacted]

Dr. K. Li, Outside Member (Department of Electrical and Computer Engineering)

[redacted]

Dr. A. Tweedale, External Examiner (Software Development, UVic)

University of Victoria

Supervisor: Dr. Eric Manning

ABSTRACT

Universities must constantly seek ways to upgrade their information systems in order to provide effective and efficient services while holding costs down and protecting the flow of information. The protection of information is a critical issue that becomes increasingly difficult as computers and networks proliferate and the applications that run on them become more sophisticated and complex.

There are areas in the University of Victoria where modern computer technology could improve information security. We describe several of these areas, then select one, campus elections, for detailed study. The main goals of the detailed study are to reveal how the conventional voting systems of the University are insecure and to illustrate how electronic voting systems can improve on their deficiencies. Several example voting systems, including two developed by the author, are analysed and compared. A functional prototype of one of the systems, produced by the author, is also described.

Examiners:

[redacted]

Dr. E. Manning, Supervisor (Department of Computer Science)

[redacted]

Dr. G. Shoja, Departmental Member (Department of Computer Science)

[redacted]

Dr. K. Li, Outside Member (Department of Electrical and Computer Engineering)

[redacted]

Dr. A. Tweedale, External Examiner (Software Development, UVic)

Table of Contents

List of tables

List of figures

Acknowledgements

The author wishes to thank Dr. Eric Manning for his guidance and suggestions, members of the supervisory committee for their participation, and the staff in the Department of Computer Science for their assistance. Special thanks go to Janet, Wilhelmina, Peter D., Fanny, and Peter W.F., for their ideas, encouragement, and unwavering support.

Glossary

account domain

The range of applicability of a set of user accounts (for example, a UNIX server and the resources that its accounts can access) is the account domain of those accounts.

anonymous channel

A logical communications path between two stations is an anonymous channel if it prevents identification of the source station (particularly by the destination station).

all-or-nothing distribution of secrets (ANDOS)

ANDOS is a protocol with which a holder of secrets can share the secrets with clients without any client knowing which secrets any other client has received **and** without the holder of the secrets knowing which secrets the clients have requested.

asymmetric key algorithm

An asymmetric key algorithm is an encryption algorithm that uses different keys for encryption and decryption. This is also commonly called a *public key* algorithm.

attack

An attack, in the context of system security, is an attempt to contravene security.

authentication

In general, authentication is the process of identifying oneself to another party from whom a service is desired. Authentication of voters is required before they cast their ballots so that the election authorities can verify that each voter is eligible and, if necessary, registered to vote in the election.

authorisation

Authorisation is a condition of access privileges that a user has in a system.

ballot

A ballot is the physical medium that contains and represents a vote (decision).

ballot form

A ballot form is the object that voters use to indicate their choices.

blind signature

A blind signature is the process of signing a document without knowing its contents. Often an additional stipulation is made that the signer is still able to verify the validity of the document, despite not being able to read its contents.

brute force attack

A brute force attack is the simplest way to recover a key used with a cryptographic algorithm to encrypt a message; the attacker tries decrypting the message with different keys, usually with a known portion of the original message to compare the results to, until the key is found.

campus computing environment

A campus computing environment is a configuration of computer systems that comprise a variety of hardware and software such that the individual components are often incompatible and their administration is organised into multiple distinct domains.

central facility

A central facility is a term used to refer to an agent that performs the roles of collector, validator, tallier, and registrar of ballots.

certification authority

A certification authority is an entity that provides irrefutable proof of authenticity of one party for another, given that the two parties trust the authority, by means of signed certificates.

cleartext

See *plaintext*.

collector

A collector is the person(s) or object responsible for collecting ballots from voters.

election

An election is the process by which members of a group reach a decision or set of decisions that will affect them as a whole. A decision could involve a single action, a law or condition, or the selection of a representative who will make decisions during a term.

electorate

The electorate is the set of voters who are eligible to participate in an election.

eligible voter

Voters must be eligible to vote before they can vote in an election. This is a status that places voters in a societal electorate, and is normally determined once. For example, when faculty members join the Department of Computer Science at the University of Victoria, they become eligible to vote in faculty elections, department elections, and some university general elections.

encryption

Encryption is the disguising of information by algorithmically manipulating its representation so that the original data can only be recovered by the possessor of the correct algorithm and a key.

fraud

Fraud is the intentional manipulation of the results of an election by tampering or by deliberate miscounting of votes.

hardened system

A computer system where especially stringent measures have been taken to avoid attack is said to be hardened. Such measures are normally applied to important servers and include limitations on, or the outright removal of, services or features, and sometimes the actual rewriting of operating system components. For example, hardening a UNIX system might include remove of all "r" services (*rsh*, *rlogin*, *etc.*), refusal of *write* privilege for some users, and a customised security system.

mail-in voting system

In a mail-in voting system, ballot forms are mailed to voters who complete and return the forms, each within an unmarked inner envelope and a self-addressed outer envelope, to the collection facility by mail.

message digest function

A message digest function takes a message of any length and computes a unique string of arbitrary length, from which the original message cannot be determined. The function must be such that the possibility of any two messages *hashing* into the same string is extremely remote (ideally 1 in 2^n where n is the length in bits of the fixed string), regardless of how similar the messages are.

one-way hash function

See *message digest function.*

plaintext

Plaintext is data that has not been encrypted.

polling booth

A polling booth is a physical station where ballots are cast (completed and deposited into a collection receptacle).

polling booth voting system

In a polling booth voting system, voters complete and submit ballot forms at designated *polling stations*, usually by privately placing the forms into bins. Authorisation is normally done at the polling station but registration, if any, is done beforehand.

public key algorithm

See *asymmetric key algorithm.*

registration

Eligible voters for an election are identified in the registration process. This process confirms the eligibility of voters and allows the election administration to produce a list of registered voters who are eligible to vote.

registrar

A registrar is the person(s) or object responsible for registration.

secure channel

A logical communications path between two stations is a secure channel if it prevents viewing of its information content by an unauthorised third party. The term *secure channel* does not imply protection against attacks such as vandalism (e.g., destruction of data).

security

In Computer Science, security is defined as protection against unauthorised use of computer resources. In information science, it is protection of data from unauthorised viewing, alteration, or sabotage.

snooping

Snooping is a passive attack on voting system security, where information is read by a party who is not authorised to do so.

symmetric key algorithm

A symmetric key algorithm is an encryption algorithm that uses one key for both encryption and decryption. This is also commonly called a *private key* algorithm.

tallier

A tallier is the person(s) or object responsible for counting votes, distinguishing between spoiled and unspoiled ballots, and posting of the results of an election.

tampering

Tampering includes the acts of changing or removing valid votes in an election without the consent of the affected voters, or adding votes that are not associated with any participating voter. Tampering may be done to commit election fraud or simply for malicious reasons.

validation

Validation is the act of verifying and certifying a ballot in an election as belonging to an authorised or registered voter who has not previously voted or (if permitted) is intending to replace a previous vote.

validator

A validator is the person(s) or object responsible for validation in an election.

vote

A vote is the decision made by a participant in an election. It is usually indicated on a ballot which is subsequently submitted for counting.

voter

A voter is a participant in an election who casts a vote.

vote stuffing

Someone (typically a collector, validator, or tallier) who secretly adds invalid votes to a tally under the guise of non-existent, unauthorised, or non-participating voters is said to be *vote stuffing* or *stuffing the ballot box*.

1. Introduction

1.1 Public sentiment about computer security and cryptography

Cryptography can be used to improve the security of many common forms of business (e.g., credit card purchases) and communications (e.g., correspondence). However, it is a widely misunderstood and mistrusted discipline, so applications of cryptography are not readily accepted into the mainstream of society. The mistrust stems from numerous incidences of security breaches involving well-known algorithms and systems that rely on them. For example, the Data Encryption Standard (DES) symmetric key algorithm, used by many computer operating systems and electronic banking systems, was recently *cracked* [6].[1]

The lack of confidence in cryptographic implementations is mainly due to misinformation drawn from incidents where computer security has been compromised. Few of these incidents have actually resulted from inherent flaws in the algorithm(s) used. The cause is nearly always attributable to implementation faults (e.g., the keys used for encryption are not well-protected[2]), bad policies (e.g., failure to enforce hard-to-guess passwords for users) and practices (e.g., transmitting passwords *in the clear* for remote authentication), and poor design decisions (e.g., using *weak* keys for encryption algorithms[3]).

---

[1] Other notable victims of similar defeat include the RC-40 symmetric key algorithm [3] and RSA [35] public-key algorithm. Both of these occurred in previous years.

[2] Windows 95 passwords can be recovered from the system registry by other users.

[3] ...such as those satisfying the U.S. government's munitions export controls [18].

Cryptography is not well-understood by the general public but the field has produced many tools for computer security, especially cryptographic algorithms, that have been used for many years. These algorithms have withstood years of analysis, so they are evidently very difficult to compromise (hence, algorithms such as the DES symmetric key algorithm and the RSA asymmetric (public) key algorithm have been widely accepted in the computer industry). The most feasible way to defeat an algorithm that is devoid of logical flaws is by trying each key from that algorithm's finite set of possible keys (called its *key-space*) until the correct one is found; this is termed a *brute force attack*. Any cryptographic algorithm can be broken by a brute force attack, in theory.

Note that the only weakness of a cryptographic algorithm revealed by a successful brute force attack is that its key-space is too small. Since the size of the key-space is directly related to the length of the key, a brute force attack can be made arbitrarily difficult by increasing the length of the key. Although a successful brute force attack may still be theoretically possible, the processing power necessary to achieve it can be so great as to be beyond the means, or not worth the while, of a potential attacker. For example, if the gain from a successful attack is of less value than the investment necessary to pull it off (and the possible loss, if caught), the attack is not likely to be attempted. Therefore, an information security function can be as secure as deemed necessary by using a cryptographic algorithm that lacks flaws and uses sufficiently long keys, within practical limits.[4]

Fortunately, there are cryptographic algorithms and security tools that can be used within the computing infrastructure of a campus such as the University of Victoria's, and can provide adequate security, even considering that the affordability of processing power quadruples every three years.[5] It is important to understand that the reliability of cryptographic algorithms and security tools is not a significant risk; it is how they are

---

[4] The longer the key size, the slower the algorithm (all else being equal) and, sometimes, the more difficult it becomes to prove the infallibility of the algorithm.

[5] This is a corollary from Moore's law, which states (roughly) that processing power doubles every eighteen months. Gordon Moore was a co-founder of Intel Corporation.

implemented that leaves them vulnerable to compromise. With thorough knowledge of their characteristics and careful attention to their implementation, information specialists can expect to achieve highly secure systems using modern cryptographic technology.

1.2 Purpose of thesis

As business and society become increasingly dependent on information systems for service provision and competitive edge, security has become one of the most popular topics in computer technology, for reasons described below. Although there are effective tools and techniques for information security, their deployment and use are hampered by rampant growth in the industry and by a general lack of understanding of the issues by the user community. This trend creates opportunities for serious damage by hackers and limits the deployment of useful networked applications.

In this thesis, we identify and outline several information security concerns on university campuses. Our focus is on the issue of privacy but we pay attention to closely related issues as well, such as authentication and data integrity. Each security concern and its significance are described, followed by possible solutions. A solution for one of the problems, that of campus elections, is investigated further in the second part of the thesis. This investigation includes a prototype implementation of the solution.

1.3 Description of a campus computing environment

This thesis deals with computer security concerns on university campuses. As an example, we use the Gordon Head campus of the University of Victoria. This university is representative of research-intensive Canadian universities in terms of its size, student population, breadth of disciplines, and its technology infrastructure. Also, this university is structurally similar to other types of organisations such as businesses, governments, and other educational institutions of comparable size, in that it has a hierarchical organisation of groups and subgroups. It appears, however, that in the University these individual groups and subgroups (i.e., faculties, departments, etc.) often possess a

particularly high degree of autonomy. This autonomy is evidenced in part by the many disparate computing systems and distinct administrative domains found among the groups. We term this situation a *campus computing environment*.

A result of multiple disparate computing systems in the University is that there are security concerns and limitations in the computing environment as a whole. Each system has, to some extent, its own security policies because the technology is unique or the administration is autonomous, or both. This makes it difficult to ensure a uniformly high degree of computer security across the campus. Also, it is difficult for applications on individual systems to co-operatively function, in large part because of the incompatible security domains of the systems. This hampers the implementation of services that must span several distinct computing systems.

Part I: Current problems related to security

2. Overview

As society and the role that information technology plays within it change, so does our attention to issues of security. In addition to basic concerns like preventing snooping and avoiding damage caused by malicious hackers, there is another compelling reason to pay stricter heed to security issues today at the University of Victoria: the *British Columbia Freedom of Information/Protection of Privacy Act* (FOIPOP). The purpose of the Freedom of Information component is to provide British Columbians with statutory rights of access to appropriate public information held by the provincial government. The Protection of Privacy component provides the obverse; protection against inappropriate dissemination of information that is considered private. A challenge arising from the FOIPOP act is to comply with one component without contravening the other (e.g., refusal to disclose information in order to satisfy the *Protection of Privacy* component could violate the *Freedom of Information* component).

Since most information covered by FOIPOP is electronically stored and accessed, information technology and security is an important topic related to the Act. The *Protection of Privacy* component of FOIPOP is of primary interest, since we are principally concerned with security measures and not directly with service enabling, which is an issue more closely associated with the *Freedom of Information* component. There have already been highly visible applications of the *Protection of Privacy Act* on the campus, such as the restriction on professors from publicly posting grades and other student information using full student ID's as identification. A good overview of the act can be obtained from the British Columbia Freedom of Information and Privacy Association [*Rankin '95*].

2.1 University security: FOIPOP, single sign-on, networking

As information technology becomes increasingly sophisticated, powerful, ubiquitous, affordable, and familiar, its role in society will inevitably continue to expand. As a result, many of the conventional practices that are accepted today may soon become undesirable or even intolerable.

For example, electronic mail is currently in widespread use at the university and, although it is highly susceptible to various levels of espionage, users are generally not concerned. Most users of e-mail rarely require privacy or are doubtful that someone would bother to monitor their communications. Many are even unaware that e-mail is so vulnerable to snooping. Yet there are already good reasons why secured e-mail communications are desirable, as evidenced by the many secure e-mail systems offered now. For instance, sensitive information such as passwords for new or reset computer accounts are often provided by e-mail because of its convenience and personal information such as credit card numbers is sometimes sent by e-mail. Also, e-mail would probably be used by more people and for more purposes if it was generally believed to be secure from unauthorised viewing and tampering.

In general, networking with computers presents security problems when the network extends beyond an area that can be completely and adequately controlled by the party in need of security. The communication links will probably be exposed to a potential attacker at one or more locations, usually on a computer which is legitimately part of that network. Chapter 3 (Insecure network communications) covers this issue.. Also, when the information systems infrastructure of a large organisation consists of disparate technologies, such as in a *campus computing environment*, management of the multiple security domains becomes difficult and can present a security problem. This is covered in chapter 4 (Multiple non-inter-operable account domains).

There are other applications of information systems that suffer from security problems, and some do not necessarily involve computers (yet) in their primary

functions. For example, elections on campus are run manually; computers are used to store and process the results. There are many security concerns with campus elections, including protection of voter anonymity, prevention of fraud by voters and election administrators, and others (see Chapter 5 --- Insecure campus voting systems).

To further investigate these issues and how they affect us, we must formalise our definition of this multifaceted subject. Among the definitions provided by the *Merriam-Webster* dictionary for security are:

1. the quality or state of being secure: as (a) freedom from danger (b) freedom from fear or anxiety.
2. measures taken to guard against espionage or sabotage, crime, attack, or escape.

Notice that part (a) of the first definition describes a true condition of safety, part (b) describes a perception of safety, and the second definition refers to the process of achieving what the first definition implies. In Computer Science, security is generally considered to mean the protection of computer resources from unauthorised access. In Information Science, security is the protection of data against unauthorised viewing, alteration, and obliteration. We consider the main security concepts for information systems to be:

- **authentication**: how a user is identified to the system
- **authorisation**: how a user's access to system resources is controlled
- **secrecy**: protecting of messages from unauthorised viewing
- **integrity**: maintaining the correct content of a message, or the detection of any alteration to it.
- **certification**: proof of the origin or ownership of a message
- **detection**: the means for discovering unwarranted use of resources
- **auditing**: means for gathering information that may be used to investigate unwarranted attack on a system
- **non-repudiation**: preventing a person or entity from denying having sent a message or having committed a transaction

The first five concepts are used to *prevent* attacks on a system.[6] Authentication and authorisation are most important as they establish who can access a system and what levels of access they have on that system. The next three, secrecy, integrity, and certification, are concerned with safeguarding communications and are especially important in networked systems, where messages originate from remote hosts and travel along channels that the system cannot control access to. The sixth concept, detection, obviously refers to a means of detecting attacks *as they occur*, while the seventh, auditing, is used to discover or investigate attacks that *have already occurred.* Non-repudiation appears to be technically similar to certification, but the two serve different purposes. Certification provides the receiver of a message with *assurance* of who its originator was, while non-repudiation provides the receiver (or anyone) with *proof* that its originator **sent** the message; the latter is used for legal binding of people to transactions.

Consider how these concepts might apply when the university provides information according to the FOIPOP Act. A person requesting information covered by the act must first authenticate herself to a clerk using some form of identification, such as a valid driver's license. This is a form of certification by the user of her identity by a third party that the clerk's office trusts (i.e., the Motor Vehicles Branch). A signed declaration of identity and acknowledgement of the consequences of fraud might also be requested to discourage cheating. At the same time, the person requesting the information will probably want assurance that the clerk is trustworthy. The presence of the clerk in the appropriate office usually suffices as certification, although an ID badge or some kind of recognisable apparel may also be used.

When the requester's identity is known, the clerk must determine if the requester is authorised to view the information. Then, the requester can be presented with the desired information. Secrecy is a critical issue at this point since any unauthorised viewing of the information by a third party would constitute a direct contravention of the

[6] Although the last three can also be considered to serve preventative roles, since knowledge of their employment in a computer network might dissuade would-be attackers.

FOIPOP Act. This is generally the responsibility of the requester, but the manner in which the information is presented is important (for example, the information may be provided discreetly on a piece of paper, rather than being shouted across a room or displayed for all to see). Another important issue is how to make the transaction non-repudiating; that is, to ensure that the requester cannot deny having seen the requested information (and therefore is bound to any monetary charge for the service and any legal ramifications from receiving it). Again, a signature usually serves this purpose.

Audit trails and detection techniques are not directly part of the information transaction but instead are ongoing background functions. Audit trails of transactions performed at the university office help to identify violators of protocol (e.g., impersonators, snoops, thieves). This is a passive physical strategy, that may utilise cameras and other surveillance devices to record each transaction. Detection is a dynamic interpretative strategy where the system must determine if an intrusion has occurred (e.g., a security guard watches for suspicious behaviour through a monitor).

2.2 Summary

We see that there are many growing security issues with electronic communications at the University of Victoria, as information system become more deeply and widely used. Also, the introduction B.C. *Freedom of Information/Protection of Privacy Act* has formalised many everyday business functions and raised the attention to security concerns around these functions. Interest in security matters at the university is growing as a result of innovation **and** of legislation.

3. Insecure network communications

3.1 Description of the Problem

The proliferation of computer networks over the last decade has exposed the vulnerabilities of modern computer systems and has simultaneously compounded the difficulties in defending against those vulnerabilities. At the University, systems are continually threatened by network attack because most, from mainframes and midrange servers down to personal workstations, are now Internet-attached and are therefore accessible from innumerable points within and without the campus. Hackers, crackers[7], and viruses are the most common intentional threats that are faced by even the most *hardened* system. Many measures are taken to detect, avoid, and prevent such threats, at the expense of effort, usability, system performance, and overall costs, because the potential damage from attacks is often far more expensive.

A networked system can be attacked directly at its point of connection to the network, including its network adapter hardware and the software that communicates through the hardware. *Worms* and *viruses* are examples of how systems can be attacked from networks. Another type of attack on networked systems is to tap into their communication channels for purposes of snooping, gathering information to gain access to systems, or simply vandalising the transmitted data. For example, a malicious user on a personal computer LAN could run a network sniffing program to acquire the secrets of other users on the LAN, including access information for their accounts (see Figure 1).

---

[7] The difference between a *hacker* and a *cracker* is that the former seeks mainly to defeat system defences, such as login steps, while the latter aims to unravel protected (e.g., encrypted) information.

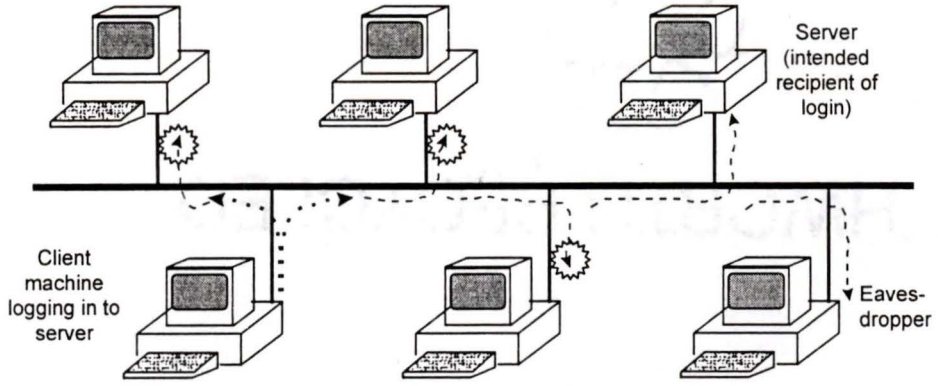


Figure 1. Security threat when communicating in cleartext

The obvious solution to attacks at network connection points is to design the connections more carefully. Unfortunately, this solution is not so readily achieved with software because there are many network connections (e.g., FTP, Telnet, http, etc.) and these connections are deployed on many different platforms, exist in multiple release states, and usually involve complex software and communications protocols. As a result, they are prone to subtle yet potentially dangerous flaws in their code. Poor administrative practices, such as the failure to follow or enforce strict guidelines for careful use of the network, can also expose a networked system.

Protecting the data traffic on a network is also simple in principle: Take access to the physical media away from would-be attackers. As with preventing attacks on network connection points, this is easier stated than done. One possibility is to limit network access to a select trusted group. Unfortunately, this would limit the range and usefulness of the network, which may not be acceptable. For instance, if a member of the trusted group wanted to send an e-mail message to someone not in the group, she would have to switch to a different network. This configuration is plausible for some highly sensitive business communications (e.g., within the University's administration) but not for

services that are intended for general access, such laboratory or library facilities for the general student body.

It is often surprisingly easy to mount a network attack. An attacker could use monitoring software (or hardware) to view the communications on a LAN or across the Internet. In a normal LAN (e.g., *Ethernet* or *Token Ring*), for example, all messages arrive at each workstation and it is up to the intended recipient(s) to keep the messages, and up to all others to discard them. Monitoring tools simply bypass these conventions to examine **all** messages. Monitoring is a difficult attack to detect since it leaves no trace; your colleague could be covertly reading your passwords for remote authenticated services from another workstation. Another attack is to impersonate the intended destination of a user's communications in order to gather information on that user. This is less subtle, and thus more detectable, than monitoring but is still easy to do since, in most networks (i.e., the Internet), the identification of a message's source (the workstation or the user, or both) is carried within the message's packet and is readily forged.

The World-Wide Web has exacerbated the dangers of network communications, as evidenced by the regular reports of successful penetration of systems by Web-originated attacks. Even if an attack does not directly involve a flaw in the Web itself, it may use the Web as a means of access to rapidly and covertly exploit a local system weakness. For example, it was recently revealed that the Microsoft NT 4.0 operating system could be breached through its Internet Explorer Web browser if a particular PowerPoint document was retrieved in a regular Web access. If the browser was configured to automatically launch PowerPoint whenever a PowerPoint document was retrieved, a subversive macro in this particular document would execute.

The Web causes problems for network security because:

1. it provides yet another conduit for attack from external sources
2. its popularity has created numerous *targets* for attack
3. it attracts many neophyte users who do not understand system security

4. rampant growth in use of Web technology and high demand from users for services impair administrators' ability to impose adequate levels of protection over a domain.

In general, a campus computing environment is exposed to many types of network attacks. As a result, most security procedures outside of very special-purpose highly sensitive applications are of a diagnostic nature as opposed to preventative. The complexity of dealing with this issue and financial constraints are the greatest handicaps to overcoming this problem. Any unwillingness on the part of the various departments and other groups in the University to co-operate to implement a centrally-developed solution would also pose a problem.

3.2 Suggested Solution

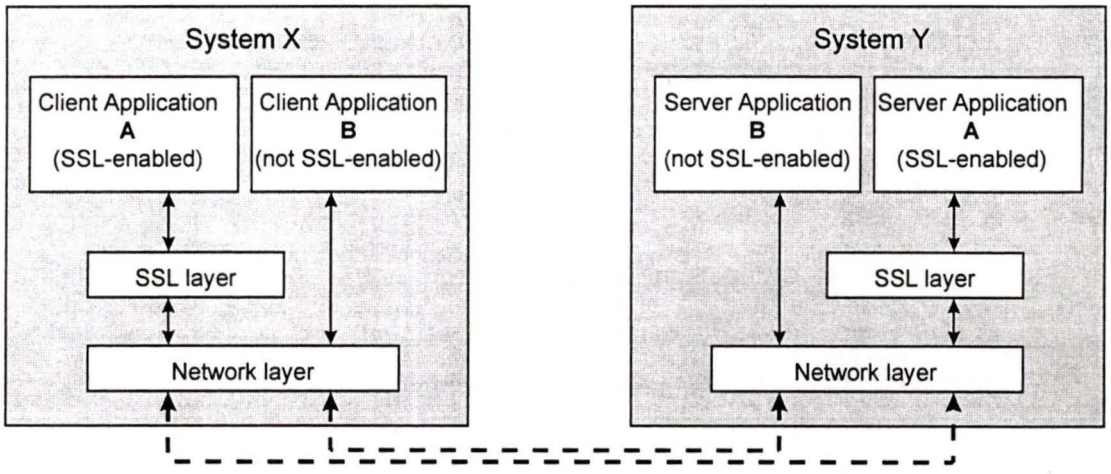
Any computer network in which sensitive information travels over communication channels that are potentially accessible by untrusted people needs network security. One effective technique for achieving network security is the application of cryptography.[8] In general, cryptography is the science of manipulating information to disguise its true content from all but its intended users. It is a well-established science and has many applications, yet it is not in widespread use today at the University and beyond.

Cryptography can reduce the potential damage to networked computer systems in several ways: by disguising transmitted information from unauthorised disclosure, by preventing forgery, and by guaranteeing to each party in a conversation the identity of the other(s). *Public key* technology, along with many recent related advances in algorithms and protocols, has spawned a plethora of promising cryptosystems that can do all of this.

One very attainable solution in the short term is secure Web communications based on encryption, such as with the popular Secure Sockets Layer (SSL) protocol (see Figure 2). For example, when users of the university's NetLink service access the NetLink

[8] It should be noted that cryptography can *improve* network security but is not a complete solution ([17]).

Web server, they do so over a regular non-secured connection. Since account information, including passwords, is often passed over these connections, users are exposed to snooping that could compromise their privacy. A simple and readily-available adjustment could make part or all of the NetLink Web server's offerings require a secure link. Since nearly all Web browsers in use today support SSL, and those that do not can be upgraded for little or no cost, it is reasonable to assume that only in rare circumstances will any user who can access the Web be unable to access a secured Web page. It is also reasonable to assume that this additional security will cause no inconvenience to users, since SSL works transparently from the user's perspective and causes little overhead or performance degradation.[9]



The Secure Sockets Layer handles the negotiation and establishment of a secure channel between client and server, including the identification of the server to the client and, optionally, of the client to the server. Its few communications with the client application(s) consist mainly of messages and, perhaps, simple information requests.

Figure 2. Simple schematic of the SSL protocol

To establish secure Web communications, we minimally need a verifiable certificate for the server, which is a certificate of identity that is owned by the server and is digitally

[9] This analysis comes from significant personal experience. The slowest part of an SSL-protected conversation is the initial connection which requires a *handshake* phase and certificate validation, and this time grows rapidly with the strength of the certificate encryption. See *http://www.netscape.com* for details on the SSL protocol.

signed by a *certification authority* that the user trusts. If the user can verify the signature of the certification authority, the user can confidently open a secure channel with the host. For the host to know the identity of the user, however, either the user must provide a similarly verifiable certificate or must perform another kind of authentication step. In the case of the NetLink service, this authentication is performed using a conventional login procedure where the user enters an account name and password. The user certificate concept is preferable, though, because it eliminates the need for entering an account name and password and because it can easily be applied to other services on other systems without the need to establish another authentication service and a new account for each user.

The following are suggested approaches to improve our systems' convertibility to secure channels:

1. Eliminate old, difficult to convert services and features.
2. Adopt newer, adaptable and securable services.
3. Wait for improved technology to retrofit existing features.
4. Develop our own protection tools where appropriate.

A similar solution to secure Web protocols that originated before the introduction of the World-Wide Web is *Kerberos*. Developed at M.I.T. as part of the *Project Athena* distributed system project [27], Kerberos uses private key cryptography to provide security features such as authentication, data privacy, and data integrity for remote communications. Kerberos is widely used in the industry and has had numerous revisions, so it is considered to be a relatively mature network security tool. M.I.T. is working on many enhancements to Kerberos including use of public key cryptography and certificates, smart card support, and commercial database integration [24]. Also, many vendors have augmented their implementations of Kerberos to provide additional capabilities, the most notable being the Open Software Foundation's Distributed Computing Environment (DCE).

Another solution, still in development, is a new version of the Internet Protocol [2] with a security specification that includes encryption of data packets. It appears that the

protocol cannot be immediately deployed in its full form (i.e., with all of its security features) so it would likely require a migratory period during which existing network infrastructure is brought up to the necessary level. Perhaps, in the meantime, this *Secure IP* could be partially deployed (e.g., with data traffic encryption only between network routers) without the need for significant upgrades.

4. Multiple non-inter-operable account domains

4.1 Description of the Problem

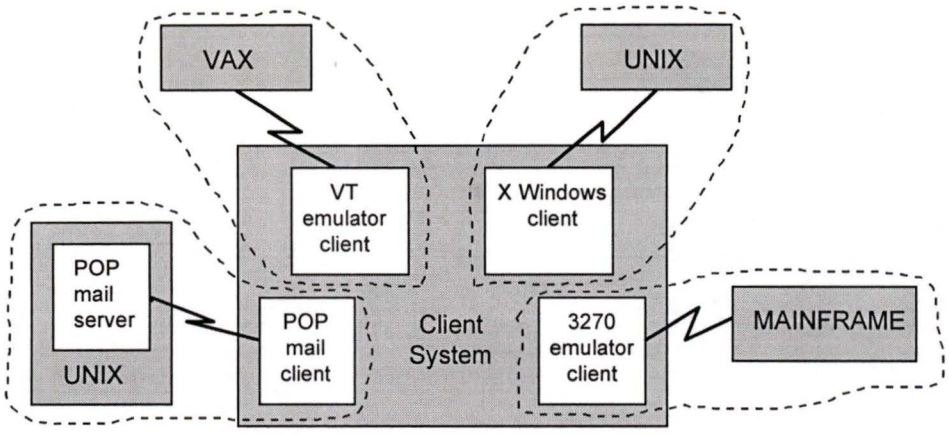
Definition: account domain

An account domain is the range of computer and peripheral resources that a set of system accounts can access. Each account may actually have access to a subset of these resources but the domain specifies the limits of what the set of accounts can collectively do.

Any large[10] organisation that has relied upon computer technology for a long[11] time will probably have many different types of computers and, consequently, users of its information systems will probably have multiple computer accounts. The University of Victoria is one such organisation. Beginning decades ago with a mainframe system to service its administration needs and to provide computing for its scientists, the university acquired more equipment as it grew, taking advantage of new capabilities and aggressive competitive tactics from emerging computer vendors. The university accumulated a broad range of equipment, from mid-range stations running UNIX operating systems to personal computers with Macintosh and DOS/Windows. At the same time, it was infeasible to dispose of the larger legacy systems, particularly the mainframe, because of the enormous investment in them and the complexities of migrating their applications to other platforms.

[10] Our notion of "large" in this context is relative, of course, and is simultaneously dependent on the "population" of the organisation and the prevalence of computer machinery within it. Most universities and governments would fit this description.

[11] Here, a long time in the computer age is approximately 20 years or more. The point is that over that time there has been considerable technological change in the industry.



- The dashed lines delineate the separate logical security domains (from the user's perspective).
- The thick lines indicate the paths of communications during client-server connections (jagged lines indicate remote transmissions).

Figure 3. Multiple services with distinct security domains

As a result, the University's present computing infrastructure is very diverse. There are many systems on disparate computing architectures and configurations across the campus. Since each system was developed with the intention of being a complete computing solution (whatever that particular solution happened to be) instead of merely a component of a larger framework, the individual security subsystems were not designed to interoperate with those of other systems. Instead, each possessed its own isolated security domain and these security domains had to be managed separately. Any attempt to develop applications that communicated between dissimilar systems over a network required the establishment and maintenance of cross-system security policy. As a result, almost every department that provides a computing service has its own account domain, and some, like the Department of Computer Science, have several. Furthermore, many students, faculty, and staff at the University of Victoria have multiple accounts on distinct systems (see Figure 3).

This is an inconvenience for the user because several accounts must be maintained, meaning several passwords must be remembered and occasionally changed. It also makes administration of the University's systems, as a whole, inefficient since the need

to co-ordinate security across many different computer systems requires "considerable duplication of effort." [39]. Most importantly (from our perspective), it is a security concern for both users and for system administrators since the practices of users with multiple accounts (with respect to those accounts) may undermine the level of protection offered to them by each individual system. For example, in order to keep track of so many accounts a user may write each account name and its password on a single piece of paper. If this paper was misplaced, the security of each of the user's accounts would be jeopardised. In addition, the multiplicity of accounts would complicate any effort by the user to protect all of those accounts in time by either closing them or changing their passwords.[12]

Another related problem is that a user might try to maintain the same password for all domains. In this case, the level of protection the user has on all systems is determined by the password limitations of the weakest system in the group. One of those systems, for example, might be used for terminal sessions (such as a mainframe or UNIX system) and set a limit of 8 characters for the password. To make matters worse, for practical reasons users are further limited to only the printable characters of a standard keyboard, of which there are about 95. Even if users chose random strings from this set, it would still reduce the number of possible keys a cracker would have to try by a factor of 3000. The worst problem, though, is that users tend to choose easily memorised passwords which are words, names, simple variations on names or words, and simple conjunctions. These factors can combine to make all of a user's accounts and services susceptible to password-guessing attacks.

The greatest problems with multiple accounts reside with the custodians of the systems. Administration becomes a much larger and complex issue than it should be, with administrators either needing to directly maintain many different systems or needing to co-ordinate with other administrators to assure proper privileges to users and

[12] Partly for this reason, a number of businesses offer an account management service to customers of credit and charge cards, including the immediate freezing of all of a customer's accounts in the event that security is compromised (i.e., because of a misplaced billfold).

to achieve interoperability across the individual systems. For example, a person joining the university (as a faculty member, staff member, or student) may require accounts in several systems under separate administrative jurisdictions. When that person leaves (or transfers within) the university, these accounts have to be *cleaned up* individually. It also becomes complex to control which services users are authorised to access in the various domains. Without a centralised authentication and directory service, it is unclear how to precisely determine what a user can and cannot do. This complexity creates confusion which is at the heart of many attacks.

4.2 Suggested Solutions

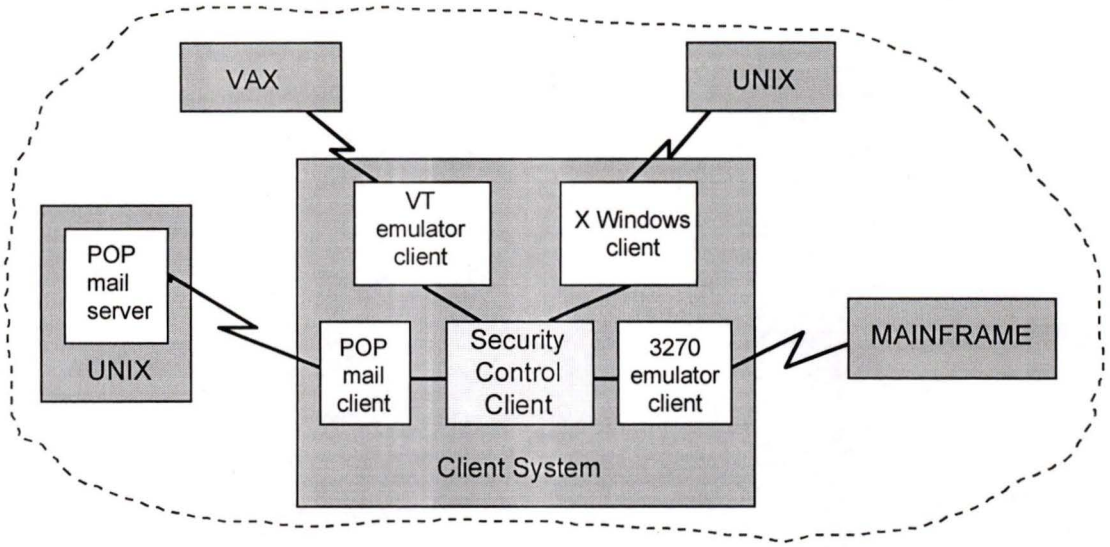
There are many products and strategies that are touted to achieve single sign-on, and these generally fall into two categories; client-based enablers and server-based enablers. This simple categorisation reflects the location of the mechanism that hides the details of multiple account domains from their users. Either solution class has its particular strengths and weaknesses but their common goal is to give users the illusion of having only one account domain for all services that they access, even if those services exist in multiple account domains.

Note that sometimes concern is raised that single sign-on, by creating a single means of access to all of a user's authorised services, actually decreases security. This is a topic that merits discussion; a brief description of the concern is provided in *Appendix A - A misconception about single sign-on*, on page 117.

4.2.1 Workstation-based single sign-on

Workstation-based single sign-on enablers modify the behaviour of the client applications that access secure network services. Often referred to as *scripting*, this technique allows the user to enter one ID-password pair through a local application which handles, in the background, the actual authentication process for each service the user accesses (see Figure 4). It is conceptually straightforward, relatively easy to implement, and so some organisations with large IT installations have developed their

own scripted authentication implementations, achieving millions of dollars in savings in administration costs and user access time [23].



- The dashed line delineates the logical security domain (from the user's perspective).
- The thick lines indicate the paths of communications during client-server connections (jagged lines indicate remote transmissions).

Figure 4. Workstation-based single sign-on

To illustrate, consider the following example: a user's typical day might include accessing services (e.g., applications, disk space, printers) on a LAN from a personal computer, using a POP mail service, and working on a remote UNIX system (via Telnet, FTP, X-windows, etc.). Each of these has its own authentication service, including an accounts database and a security interface protocol. Normally, the user must authenticate to a LAN server to get various software services and network access, then sign on to the mail service either once or repeatedly to pick up electronic mail, and then sign on to the UNIX system to open a terminal session.

With a workstation-based single sign-on enabler, the user needs only to sign on once, effectively to a local application. That application might immediately sign on to the LAN and retain the user's account name and password. Thereafter, whenever the user wishes to access the mail service or the UNIX system, the local application will automate the login procedure without the user's intervention (e.g., when a Telnet session

is started, the first thing the user sees is a terminal emulation screen and the shell command prompt).

Establishing this type of automation is usually straightforward because all modifications are performed on the client systems, specifically as to how users interact with their client applications: there is no change to the communications between client applications and server applications. Also, the task of adapting a single sign-on enabler to properly control the client application software, which is the most challenging step, can be simplified by taking advantage of the automation features present in some applications. For example, at least one popular terminal emulator can be configured to script the entire remote server login process, reducing the user's effort to the launching of the application. Many *File Transfer Protocol* (FTP) and electronic mail applications share this property.

For those applications that do not support scripted login, the single sign-on enabler would have to take temporary control of the application at the user interface or at the network interface. In the former case, the enabler simulates the user's actions by responding to certain conditions (e.g., the displaying of a **login** prompt) by inputting the appropriate character sequence (e.g., an account name)[13] without filtering any information. The latter case involves similar monitoring except that it occurs at the interface between the application and the network services of the operating system. This allows communications to be intercepted (and filtered) before reaching the user interface. For example, a user launching a terminal emulator would never see the normal account name and password prompts but instead would first see what usually follows *after* a successful login (see Figure 5). The network interception idea is much more difficult to implement but makes the intervention of the single sign-on enabler (as well as the entire login process) transparent to the user.

---

[13] This is sometimes termed *screen-scraping* because the enabler monitors what is being displayed on the screen, or a particular window, looking for certain strings of text to react to.

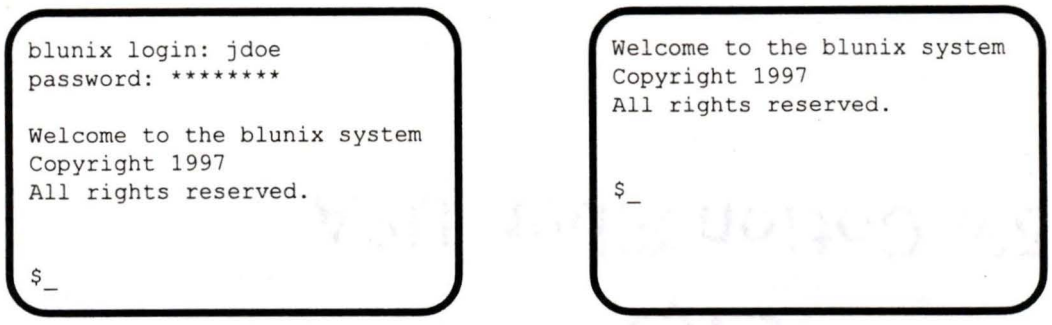


Figure 5. Intercepted emulation compared to normal emulation

So far we have seen how a workstation-based single sign-on enabler can reduce the number of logins by a user for multiple network services to one. This does not, however, reduce the number of accounts that must be managed by the user (e.g., for periodically changing passwords). In order to do so, the enabler must automate management of the individual accounts. The goal of automated account management is to require only one repetition of any management procedure by the user, such as a single periodic password change. There are two ways that account management can be automated: all accounts can all be set up similarly (e.g., the same account name and the same password) and controlled the same way, or the information for each account can be stored locally in a configuration file and the accounts can be controlled individually.

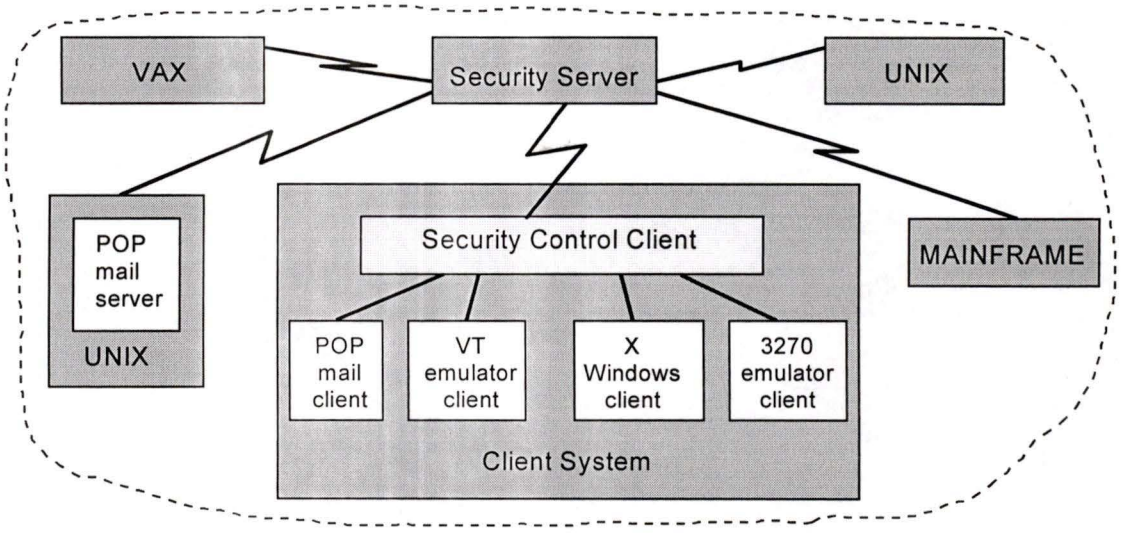
It may be difficult or impossible to establish common account information because of differing security policies on the various systems involved. For example, each account domain might have its own account naming policy, its own restrictions on the length and lexical content of passwords, and its own schedule for password expiration. A common account information approach would have to simultaneously satisfy each of the account domains' rules.[14] The alternative of dissimilar account information could

[14] A situation could arise where no compatibility existed between two or more authentication systems for a critical rule. For example, an older scheme might limit passwords to a maximum of 7 or 8 characters in length while a new scheme might set a minimum password length to a number greater than 8 characters.

also cause problems if, for example, a user changed her password to a value that was not acceptable for some account domains. The enabler would have to determine how to choose new passwords independently under these circumstances.

4.2.2 Server-based single sign-on

The main drawback to workstation-based single sign-on is that it requires the installation, configuration, and maintenance of special software on all the clients in a networked computer system. This imposes a great amount of labour and responsibility on the system administrators and therefore also increases the risk of errors. Another technique for achieving single sign-on is to centralise the authentication to all services on



- The dashed line delineates the logical security domain (from all users' and servers' perspectives).
- The thick lines indicate the paths of communications during client-server connections (jagged lines indicate remote transmissions).

Figure 6. Server-based single sign-on

one or more network servers. In this scheme, the *authentication servers* act as *brokers* between the client applications and the server applications. Clients (and servers) have local agents that handle negotiations with the authentication servers. Figure 6 illustrates the server-based single sign-on concept.

When a client wishes to access a service within the environment, it must present valid credentials from an authentication server, which could be in the form of a certificate or a ticket that is subsequently presented to any participating service that the client wishes to access. A certificate provides proof to the server of the client's identity, so that the server can consult with an authentication server to determine the access rights for the client. A ticket is more direct in that it immediately demonstrates access rights that were previously validated by the authentication server; tickets require clients to initially sign on to an authentication server. In either case, not only do all services *appear* to exist in the same logical authentication domain, but they in fact *do* exist in the same logical authentication domain.

The main advantage of server-based single sign-on is that administration of the authentication software is centralised; modifications do not have to be propagated to the client systems and users do not have to be aware of administration of the single sign-on enabler. Even if the authentication service is distributed over multiple computer systems to improve performance and availability, it is still easier to maintain than attempting to reconfigure hundreds or thousands of client systems. New services can be added without users needing to modify any authentication software, and possibly without even having to adjust their systems' configurations.

The server-based solution ostensibly redefines the interfaces of applications in a networked system. Services on remote systems no longer require the overt identification step that is typical of networked applications (e.g., the login and password prompts). Instead, they rely on services from the underlying operating system to manage their authentication credentials or capabilities, even if the client and server are on different machines. This low-level location-independent security management is critical to masking the physical separation of the individual components of a networked system, which is an basic feature that distinguishes distributed systems from simple networked systems. Therefore, it is not surprising that discussions about single sign-on usually include distributed system technology as well. In fact, the Distributed Computing Environment (DCE), which is probably the most mature commercial distributed system

environment available today, is the single most frequently mentioned technology in articles about single sign-on. A good general article on the virtues of DCE from the perspective of a university can be found in [31].

Of course other solutions exist in the potentially lucrative middleware[15] market. Two are distributed objects and World-Wide Web-based networking systems. Distributed object systems either provide a comprehensive security model similar to that of DCE or specify a general infrastructure for single sign-on. Web-based single sign-on derives from the power and popularity of the World-Wide Web, namely platform independence at the application level and standardised authentication (and data privacy) mechanisms. The potential for using the Web as an interface and secure transport for networked applications and communications is enormous. Many Web solutions use X.509 [8] certificates for mutual authentication between client and server, and Web products already perform this authentication transparently to the user with minimal prior configuration. Although no dominant Web-based solution has emerged yet, there are many contenders with very promising products.

The challenge for server-based single sign-on is how to migrate existing applications and services to the central authentication service. This is more than just a technical issue because there are so many solutions already and none is clearly dominant or universally adaptable to other solutions. Since the decision to select a particular server-based single sign-on solution involves a substantial investment over time in terms of product development and acquisition, it may soon be impossible to reverse or migrate the information systems infrastructure without huge costs. This technology will blossom when widely accepted and well-conceived standards for authentication on computer networks are established. Nevertheless, at least one major information technology research organisation [1] expects the deployment of server-based solutions to surpass that of workstation-based solutions by the end of 1998.

---

[15] We define middleware as the software that resides between business applications on end-users' systems and the data manipulation applications on the systems where the data is stored. This software has responsibilities such as security, flow control, transaction processing, and directory services.

5. Insecure campus voting systems

5.1 Introduction

At the University of Victoria, all departmental, faculty and many student body elections[16] are administered by the university and are processed through the office of the University Secretary. Approximately forty of these elections are held each year including term administrative positions, committee memberships, student political positions, and referenda. The elections are operated using either of two voting protocols: the *polling booth* protocol and the *mail-in* protocol.

Students at the university are most familiar with the *polling booth* voting protocol, where participants (voters) indicate their votes on a ballot form in the privacy of a *polling booth*[17] and deposit it directly into a bin (see Figure 7). This is the same protocol used for political elections in Canada. It is most often used for elections in large, diverse populations because of its security qualities and because it scales well (i.e., its resource demands and costs increase at a reasonable rate as the size of the electorate increases).

The general procedure for a polling booth election includes three major steps: registration, voting, and tallying. In the registration step, someone is designated as the *registrar*; this person determines who is eligible to vote in the election and compiles a *voters list*. Other organisational tasks, such as setting dates, arranging locations, and verifying rules are done in the registration step. In the voting step, voters go to a

[16] Some student organisations, most notably the University of Victoria Student Society (UVSS), function independently and thus run their own elections.

[17] In fact, the degree of privacy is usually minimal and controlled by the voter; for elections at the university, an actual polling booth is not used.

designated *polling station*, present identification to someone for verification of their eligibility to vote, indicate their choice(s) on a special form known as a *ballot*, and deposit the ballot into a bin. The person responsible for verifying a voter's eligibility is called the *validator*. In the tallying step, the votes collected in the bin are counted by another person called the *tallier*. After completing the count, the tallier posts the results.

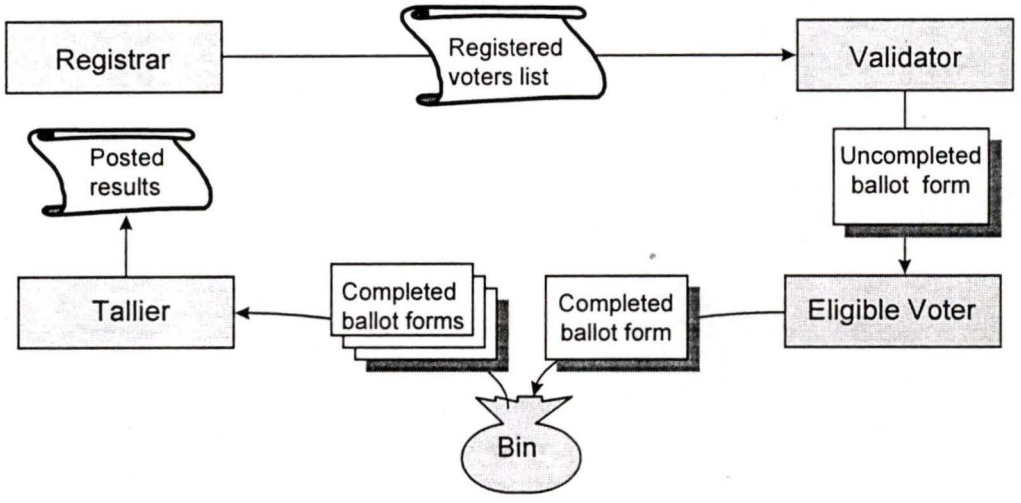
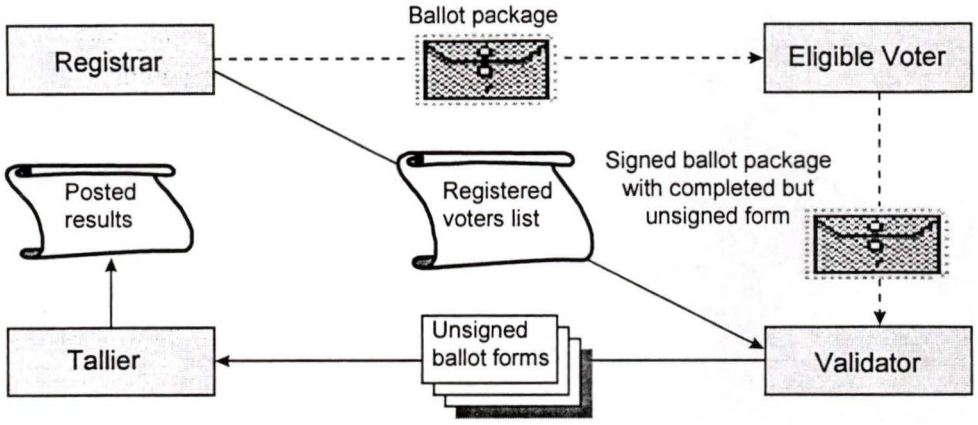


Figure 7. Polling booth voting system

Another type of voting system, called a *mail-in* voting protocol, is typically used for faculty and departmental elections because it is simple and inexpensive to run and is very convenient to the voter and the administration (see Figure 8). The mail-in protocol is suitable for elections involving small or geographically dispersed populations where the overhead of polling stations would be relatively prohibitive. However, it can be used for almost any election, as long as security demands are not too high.

The mail-in voting protocol has the same major steps that the polling booth protocol has, and the registration and tallying steps are essentially the same; the main differences lie in the voting step. In this protocol, ballots are completed and sealed inside blank envelopes; those envelopes are then sealed inside larger envelopes bearing the voter's signature, which are subsequently mailed to the validator. The validator confirms the

registration and authorisation of voters from the information on the outer envelopes, then removes the inner envelopes[18] and passes them unordered to the tallier.



Dashed lines indicate which transfers use the mail service. Communications not involving the voter do not necessarily use the mail service because the registrar, the validator, and the tallier might all be in the same physical location.

Figure 8. Mail-in voting protocol

At this point we need to specify the terminology we will use in describing voting protocols. These terms will apply to both conventional and unconventional (i.e., electronic) voting systems wherever practicable.

We will use the term **election** when referring to the act of making one or more decisions by democratic choice. This will include elections of persons for titles or positions as well as referenda. One who participates in an election to submit a choice is a **voter** and the choice that the voter makes is a **vote**. The actual physical medium that contains and therefore effectively represents a vote is the **ballot**. The process of submitting a choice is referred to as **voting** in the logical sense and **casting a ballot** in the physical sense. Often these two terms will appear to be used interchangeably; it should be clear from the context if either, or both, applies. Ballots are cast during the **polling period** which begins at the **opening of the polls** and ends at the **closing of the polls**.

[18] Hence the term *double-envelope* system.

The objects (people, devices, programs, etc.) which carry out the necessary work for an election are the **administrators**, and may include a **registrar**, a **validator**, a **collector**, and a **tallier**, as well as organisers. Prior to an election, the set of **eligible voters**, which includes everyone who has the right to vote in the election, is determined (or confirmed and updated if previously established) by the administrators. Eligible voters then declare their intent to vote by registering with through the registrar. At the time of the election, registered voters confirm their registration with the validator just prior to voting. The collector gathers the submitted ballots and presents them to a tallier who calculates the final results.[19]

The entire ensemble of processes, objects, and tools involved in an election is a **voting protocol**. An implementation of a voting protocol, including administrative processes, is called a **voting system**. We usually make references to voting protocols because we are concerned with algorithmic and procedural details, but sometimes it is unclear as to which term should apply within the context; in this case, the term *voting protocol* is preferred to the term *voting system*.

In order to assess the merits of voting protocols, we need a set of criteria to measure or compare them with. Logically, we should first consider the high level goals of the protocols and resolve our criteria in view of these goals. An example of a (very) high level goal for a voting protocol is *to encourage the awareness, acceptance, and support of democracy throughout society*. A good voting protocol is necessary, if not sufficient, to achieve this kind of goal. Still, a more tangible high level goal is needed to convince people of the importance of finding a better voting protocol. One such goal is *to enable and encourage direct democracy*.

Direct democracy is a system in which all decisions of consequence for a society are made by all[20] of its people collectively through elections. In practice most decisions are

---

[19] The collector is usually so closely associated with the validator that they are one in the same (normally referred to as the validator). Also, in some cases, the role of the registrar or the role of the tallier are merged with the validator.

[20] For various reasons many members of a society are deemed to be "unfit" to make decisions on some or all issues affecting that society[20]. Infants, youths, invalids, and convicts are among such persons.

made by elected representatives of a society; direct democracy has historically been an unrealistic ideal because of the following reasons [25] [29]:

1. Voting systems that are *completely* immune to fraud, misrepresentation, and coercion appear to be difficult (and perhaps impossible) to achieve.
2. Populous and geographically diverse societies tend to make numerous decisions and are complex. The sheer volume of votes due to the number of elections and the size of the electorate make direct democracy expensive and cumbersome.
3. For many reasons, voluntary participation in elections, particularly those that are large-scale, often involves only a fraction of the eligible population and exhibits a skewed representation of the overall demographics.
4. Conventional voting systems, especially in larger societies, use proportionately large amounts of resources such as paper, energy, and human & machine time, so they tend to be expensive and environmentally taxing.
5. Many believe that the people of a society are incapable of collectively making decisions in the best interests of that society because they are inadequately educated and informed, and because they do not fully comprehend their democratic responsibility [29].

For these reasons, most democratic societies in the past arranged to elect a number of representatives to decide which issues should be voted on and to carry out the vote.[21] Usually the voting was done by the representatives themselves, and sometimes decisions were made without a vote at all. This form of *representative democracy* had been generally accepted and, to some extent, necessary because of the aforementioned challenges to direct democracy. Unfortunately, the reality is that governments frequently make inappropriate decisions for society based on exaggerated influences from within

[21] There are other significant reasons, such as the control and influence of society by certain members of wealth and position. They have historically exerted a formidable resistance to the adoption of a purer form of democracy and may continue to do so well into the future.

and from special interest lobby groups [20]. This problem also extends to business and academic institutions.

In the modern era we have an abundance of communications media, including radio, television, telephony, computer networks, and others. As time progresses these media are rapidly becoming more widely available, more powerful, more functional, and less expensive. Perhaps most importantly, and as a direct result of the previous five points, sophisticated communications media are becoming widely used and accepted by the general public. We may be able to exploit these technologies to improve the democratic process by making voting systems faster, more accurate, more convenient, more flexible, and less expensive, eventually eliminating the need for representative democracy.[22]

Designing a voting protocol to support the goal of direct democracy is challenging because when we optimise one important quality of elections it is usually at the expense of another important quality. For example, the preservation of voters' anonymity complicates the prevention of fraud by both voters and election administrators because the former requires detachment of voters from their votes while the latter requires voters to be able to track their votes and, if necessary, prove their association with it. Further, it is challenging to achieve either of the above qualities while making a voting system easier to use.

The qualities mentioned above are examples of *ideal*[23] qualities of voting protocols. There are two subsets of ideal qualities: the *functional* and the *aesthetic*. Functional qualities are those characteristics that directly support democratic ideals while aesthetic qualities determine the appeal of a voting system. Functional qualities are essentially security issues, and are listed in Table 1 in order of their importance to voters.

---

[22] This does not necessarily mean that elected government officials (i.e., politicians) would be eliminated, even though the Democratech Party of British Columbia stated so in their mandate in 1992; rather, they would likely serve solely as agents of the people.

[23] It is important to stress the word *ideal*, since most of these qualities are provably less than 100% unattainable.

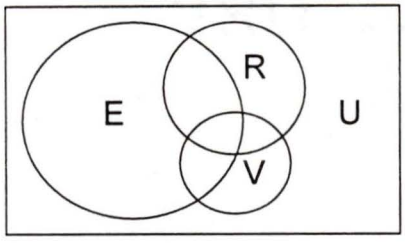
Table 1 Functional qualities of a voting protocol

F1:**Anonymity**. No one can determine how anyone else voted.
F2:**Validation**. Voters must be eligible, registered, and authorised to vote.
F3:**No multiple voting**. Voters can vote once at most.
F4:**Tamper resistance**. No one can get away with tampering with the votes or the voting system.
F5:**Accuracy**. Votes must be tallied accurately.
F6:**No direct influence of the voter**. Voters cannot be bribed or coerced into voting in a particular way.
F7:**No indirect influence of the voter**. No results are posted until the polls have closed, and no one can vote after the closing.
F8:**Verifiability**. Voters can verify that their votes were tabulated correctly.

Qualities F1, F2, F3, and F4 are the most critical to the technical acceptability of a democratic voting protocol. F5 is less than critical since some margin for error is usually tolerable, especially in large scale votes, as long as it is deemed to be unintentional and insignificant. If the margin for error is too high, however, it becomes an attractive cover for election fraud. F6 is often stated as "Voters cannot prove they voted in a particular way," which is a necessary condition to dissuade bribery and coercion. It happens to be very difficult to achieve for protocols such as the mail-in voting protocol where voters mark their ballots in isolation from the election officials. F7 is important in that no one should have any information about how others voted since it might influence their decision. F8 is valuable for discouraging election fraud and is directly related to F4, except that it is detective as opposed to preventative. This quality is rarely attained in practice with a conventional voting protocol because it presents logistical and administrative challenges to be both effective and non-intrusive.

Figure 9 illustrates the requirement for authentication in a voting protocol. Within a population (U), E is the set of all eligible voters (i.e., can register to vote), R represents all registered voters (i.e., are authorised to vote), and V is the set of voters who actually voted (i.e., participating voters). This illustration accounts for the possibility that some voters are not registered or even eligible. We want to establish that $V \subseteq R \subseteq E$. In other words, only eligible voters may register to vote, and only registered voters can vote.

Much of this restriction depends on proper administrative procedures, but the effectiveness of those procedures is greatly enhanced by a well-designed protocol.



U : entire population that the election is applicable to
E : those eligible for the election
R : those registered for the election
V : those who vote in the election

Figure 9. Voting status distribution

Aesthetic qualities are characteristics that make voting systems easier and more pleasant to establish, use, and administer. They are listed in Table 2 in order of their likely appeal to voters. **A1**, **A2**, **A3**, **A4**, and **A5** enhance voting for the voter; this is a crucial goal for a voting system to be successful. **A4**, **A5**, and **A6** also appeal to those active in seeking democratic reform by making elections more versatile, less time consuming, and more affordable.

Table 2. Aesthetic qualities of a voting system

A1:Convenience. Voters need not travel great distances or incur significant expense to participate in an election.
A2:Availability. Votes can be cast at any time up to the closing of the poll.
A3:Simplicity. The process of voting is easy to understand by all voters.
A4:Flexibility. The voting system supports different types of elections, and voters can change their votes before the polls close.
A5:Performance. An election or referendum can be rapidly deployed, carried out, enumerated, tabulated, and reported.
A6:Economy. The voting system costs very little to run.

5.2 Description of the problem

Voting systems at the University of Victoria are known to be susceptible to some measure of fraud. The strongest concerns come from those who participate in faculty and departmental elections, but there undoubtedly are detractors among the student

population. Many students, however, do not demonstrate concern about this potential for fraud, probably because they feel that the elections are not important enough or they are too detached from the issues and the process. To the latter effect, the consistently poor turnout at student elections[24] may actually be seen in large part as a form of *silent protest* against a lack of desirable candidates, a poorly viewed political system, or a suspect voting system. It is non-trivial to determine how far-reaching the effects of a tamper-prone voting system may be.

Functional Problems

Problems with the University's voting systems exist mainly because of problems with the voting protocols that they are based on. Of the problems with the polling booth and mail-in voting protocols, many are functional. Furthermore, both protocols fail to satisfy many of the functional criteria to some extent.

Problem: anonymity is threatened

Participants in faculty and departmental elections are much more attuned to the issues of those elections. As a result, some faculty members have complained about the inadequate security of the mail-in voting system. This is not surprising considering that the mail-in protocol depends heavily on the trustworthiness of the election administrators, leaving opportunities for snooping, tampering, and fraud. Since all of the ballot information is tied to the voter's identity in the double-envelope submission, the inner unmarked envelope containing the ballot must be permanently separated from the outer signed envelope before its contents are viewed. This requires that:

1. the ballots and inner envelopes sent to the voters are unmarked and generic
2. the double-envelopes remain inviolate until received by the validator
3. the validator verifies the identities of the voters, removes the inner envelopes, and gives the inner envelopes to the tallier in no particular order
4. the tallier examines the ballots in the envelopes it receives from the validator and counts the results independently.

[24] For example, in elections by the students to the Board of Governors and to the Senate in 1996 and 1997, fewer than 10% of the eligible voting population (student enrolment for the year) cast ballots.

In fact, each of these requirements represents a possible point of compromise of the voter's anonymity which is highly dependent on human integrity, care, and attention. The inner envelopes and the ballot forms could be traced to the voter by means of discreet markings or a discernible uniqueness in their physical makeup. When the double-envelope ballot packages are sent to the validator, the voter must rely on the delivery system to be secure and honest. After receiving the ballot packages and verifying the voters' identities, the validator could examine the contents of the inner envelopes to find out how voters voted. If necessary, it could also conspire with the tallier to achieve the same by using markings or an unscrambled ordering of the inner envelopes.

Voters can meet the first requirement by examining their ballot packages closely for markings and, if possible, comparing them to publicly posted samples to ensure that theirs are not unique. The other requirements demand the trust of the delivery system, the validator, and the tallier, or the presence of a trusted individual at each of those stages.

The polling booth protocol provides better protection of voter anonymity than the mail-in protocol because the responsibility of separating votes from voters' identities is given to the voters instead of the election administration. Voters complete ballot forms and submit them into an unordered bin from where they cannot be traced back to the voter. There is still a threat of ballot marking and there is also a possibility of deception concerning how the bin is managed, but most of the concern for anonymity is alleviated.

Problem: tampering

There are other security threats that can occur after the ballots have been collected. These are applicable to both systems since the protocols are essentially the same from the point of collection onward. These threats are generalised as *election fraud* since they involve tampering by the election administration. They include manipulation of submitted votes, such as deletion, alteration, or deliberate spoiling, and *stuffing the ballot box*, where fraudulent votes are added to the count (see Appendix B - Stuffing the ballot box - on page 120). Since the votes are completely separated from the voters who cast

them after they have been collected, there is no way for voters to verify that their votes were counted unaltered in the final tally. There is also nothing to stop the validator or the tallier from adding votes, except that they cannot do so to the extent that the final election results enumerate beyond the size of the electorate, assuming that this number is known to others and can be verified.

Another form of tampering peculiar to the mail-in protocol can occur during the delivery of the ballot to the validator. Specifically, the collection, processing, and delivery of the mail are prone to interception and subsequent removal, replacement, or modification of completed ballots.

The usual safeguards against tampering are to have trustworthy administrators, to have several administrators in place to watch over one another during each step of the vote processing, and to have trusted external parties to scrutinise the entire process; these apply similarly to the delivery system. These safeguards are not only resource-demanding but are also highly dependent on the trust of the individuals involved and therefore are not convincing to many voters.

Problem: inaccurate counting

Miscounting is possible in any voting system that relies on humans to tally the votes. It is a more significant concern in large-scale elections because the rate of human error (i.e., number of errors per counted vote) tends to increase with the size of the election. Although it is related to election fraud, counting errors are assumed to be unintentional and are minimised by strict preventative measures (greater care and redundancies in the counting process) and tolerance of any estimated error that is considered to be unavoidable. This is important not only because an inaccuracy in the count reduces the acceptability of the results but also because it provides a shelter for fraud.

Problem: multiple voting

There is also a possibility of *voter fraud*, in which voters attempt to tamper with the system protocol. The most common form of voter fraud is for voters to attempt to submit multiple ballots, especially by impersonating another eligible voter. The mail-in and polling booth voting protocols attempt to prevent this by their respective

authentication and registration processes. The mail-in system relies on signatures and envelope seals while the polling booth system relies on physical identification.

In the case of the mail-in protocol, a malicious voter might (i) forge the signature of another voter and submit a fraudulent ballot or (ii) intercept the voter's submission, open the outer envelope, replace the inner envelope with one containing a fraudulent ballot, reseal the outer envelope, and submit it. The first case attack would be detected if multiple votes signed by the same voter were received. However, if the valid ballots of the affected voters were first intercepted or if it was known to the malicious voter that they would abstain (e.g., deceased), then there would be no clear evidence of multiple voting.

In the case of the polling booth protocol, there is less opportunity for voter fraud because of the requirement that the voter be physically present at the validator when casting a ballot. It is still dependent on the accuracy of the identification check, though. Voters **can** attempt to identify themselves as other voters but will probably need to do so at different polling booths. Since this is difficult to do in high volume and since polling booth elections are usually used for large electoral populations, any damage by voter fraud is probably negligible. For the mail-in protocol, however, electoral populations are usually small and decision margins can be several votes or less, so voter fraud is a significant issue.

At this point we note an important consideration in designing a voting protocol: while tampering has a tangible effect on an election and therefore can theoretically be detected, an *invasion of privacy* does not necessarily affect the outcome and so, depending on what is done with the information, it may be impossible to detect. By invasion of privacy, we mean the discovery of election-specific information about the voter (i.e., how the voter voted) without the voter's consent. Only a strategy of prevention will be useful against this kind of attack since there seems to be no reliable means of detection. This imposes a challenge on designers because goals such as fraud prevention impose severe constraints on satisfying other voting protocol goals, particularly preservation of anonymity.

Problem: voter influence

Yet another problem that is peculiar to the mail-in protocol and other protocols that permit location-independent ballot completion is that of bribery or coercion. Since voters complete their ballots in complete privacy, there is no simple way to guarantee that no voter was bribed or coerced by a third party into voting a particular way. Ideally, it should be provable that a voter could not be coerced or bribed. In general, this means that voters should have **no** means of proving that they voted in a certain way, whether that proof is some kind of certificate or their being observed.

Problem: results are unverifiable

Finally, in addition to their being prone to various forms of fraud and tampering, the protocols used by the University's voting systems provide no means for voters to verify that their votes were counted and unmodified in the final tally. This problem is directly related to the problems of tampering and inaccuracy in that it allows them to occur.

Aesthetic problems

The security issues discussed above affect the acceptability of the two voting protocols, especially the mail-in protocol. Since the security of the mail-in protocol is dependent on the trustworthiness of so many elements, it is best suited for small-scale elections. However, its convenience is especially valuable for geographically diverse electorates and for this reason the Teamsters' union in the U.S. uses the mail-in protocol for its nation-wide elections [15]. Unfortunately, there are still some aesthetic problems with the mail-in protocol and many with the polling booth system that negatively affect their appeal and effectiveness.

Problem: inconvenience

From the voter's perspective, a polling booth system is very inconvenient. The voter must travel to a polling station, must remember to bring proper identification, and must be prepared to wait in line to vote. The voter must also have determined her vote in advance or risk making a hastened decision because of limited time at the polls. A mail-in system is more convenient in comparison because the voter can complete the ballot virtually anywhere and at any time between receipt of the ballot package and the closing

of the polls. However, the voter must be present at a certain addressable location or must travel to a predetermined location to receive the ballot package.

Problem: poor availability

Another problem with the polling booth protocol is that voters must vote when the polling booths are available. Since it is costly to provide the facilities for the polling stations and to staff them, they are not open for long. Voters must plan a trip to the polls from a rigid schedule. The mail-in protocol solves the availability problem by allowing voters to complete their ballots at any time from receipt of their ballot packages up until a sufficient amount of time before the polls are closed to allow for mail delivery.

Problem: inflexibility

Although both the polling booth voting protocol and the mail-in voting protocol are probably suitable for many types of elections (e.g., referenda, approval elections [5], etc.), neither allows users to retract their votes once submitted. This is because, once submitted, the vote is completely separated from the voter. There is no way for a voter to prove to the election officials which vote should be deleted (or removed from the tally) in favour of the new vote.

Problem: poor performance and high cost

Both the mail-in protocol and the polling booth protocol suffer from poor performance[25] and high expense[26], which restrict their use. The most notable effect of these prevailing conditions is as a reason, or at least an excuse, for having elected representatives make the majority of our decisions, as opposed to letting the electorate make the decisions by direct democracy.

---

[25] Compared to an ideal of near instantaneous response from the submission of a vote to the receipt of the vote and from the counting of the votes to the posting of the results.

[26] Compared to a zero cost ideal, or at least one with negligible variable costs for non-promotional materials and minimal administrative costs. Fixed expenses do not affect the frequency of use.

5.3 Suggested solution

Since both incumbent voting systems suffer many shortcomings, there is opportunity for a system based on an alternative protocol to replace either or both. Its only requirement is to improve on some of the shortcomings without compromising any important strengths. Many novel, alternative voting protocols have been conceived but have failed to replace the conventional protocols because they do not effect a net improvement. For example, most contemporary alternatives are based on some kind of modern technology and the cost, availability, and unfamiliarity of the technology has hampered their adoption.

The main constraints in using computers as the basis for a voting protocol are that they are not yet commonplace throughout society, are not always simple to use, and are not yet highly accepted by the general populace. However, each of these problems is rapidly diminishing with time and, already in certain sectors and organisations, computers are prevalent, well-understood, and generally accepted. The University of Victoria, for example, has an especially high density of computers, making it an ideal proving ground for computer-based voting systems which may become feasible throughout society.

Computer-based voting is not new, but it is rarely used in practice for several reasons, particularly because the necessary technology has been too expensive, immature, and esoteric. To some extent, computers have been used in the electoral process since the 1960s, mainly for counting, tabulation, and storing of results. For example, some elections used so-called *voting machines*, where voters filled out *punch card*[27] ballot forms and either *fed* them into the voting machine directly or placed them into a bin which clerks could later feed into the machines.

Although the use of computer systems to enhance elections is not new, the use of cryptographic techniques to enhance their functionality is. Public key cryptography,

[27] In older systems, a true punch card was used, but since the 1970's, at least, special devices were available to read simple entries made with a pencil, such as a darkened circle.

developed by Diffie and Hellman in 1975 [19], made highly secure fully electronic elections viable; throughout the 1980s a number of papers appeared in cryptographic journals proposing new, secure voting protocols. The best of these protocols make security breaches such as vote manipulation and invasion of privacy as difficult as defeating the cryptographic algorithms that they are based on.

Electronic voting protocols have other attractions. For instance, spoiled ballot forms and tallying errors will be virtually eliminated if computers control the processing. Furthermore, in addition to improved security using modern cryptographic techniques, electronic voting protocols can achieve better performance, convenience, flexibility, and availability by using the powers of computers and networks. Registration, authorisation, and verification of voters can be nearly instantaneous, as can updates to the lists of eligible and registered voters. The reduction in human effort would also lead to significant cost savings. Results might be more quickly available.

Another attractive promise of computer-based voting is that new possibilities that were previously impossible or impractical might now be attainable. For instance, depending on the protocol used, the voter can verify that her vote was counted correctly and can either resubmit her vote or contest the tabulation of it. Opportunities for ballot box stuffing and vote tampering by some or all parties involved might be eliminated. Perhaps the most appealing possibility for proponents of direct democracy and, hopefully, the general public[28], is that, with enhanced performance, convenience, and cost-efficiency, elections can be held much more frequently, thereby enhancing democracy.

However, there are many technical and practical challenges to finding the perfect voting protocol. Since the security, and therefore the effectiveness, of a computer-based voting protocol is largely in its logic[29], the algorithms must be resistant to attack.

---

[28] ... in fact, almost anyone but politicians, who would recognise this as power being wrested from them.

[29] Electronic voting systems do rely on physical security, such as the protection of equipment from unauthorised access or damage. Some electronic voting systems even require voters to be present at a ballot collecting station for authentication, as in the polling booth system [4].

Furthermore, the implementation of the algorithms (the software) must be trustworthy. This implies certification by a trusted authority or availability of the source code for scrutiny. It also means physical protection of the protocol, such as limited accessibility. Generally the communication links between stations cannot be guaranteed secure, so encryption of the data or the channel is necessary. Finally, the lack of physical security measures at the origins of votes (i.e., the voters' computers) must be addressed because it makes voter coercion possible.

6. Other areas with security risks

There are many other areas in a campus computing environment where information security is at risk and can be substantially improved with modern cryptographic techniques and technology. One such area is electronic payment schemes, using magnetic stripe vending cards for services ranging from photocopying to cafeterias. Despite their susceptibility to fraud, magnetic stripe vending cards continue to be widely used because they are relatively inexpensive and convenient. The threat of fraud limits their use to services of low importance[30] and to small value (less than $10) transactions. Nevertheless, there have been many reports of attacks on vending cards because, like cash, they are relatively easy to forge and do not require proof of rightful ownership for use (i.e., authentication), other than possession. For example, students at the University of Calgary discovered several years ago how to copy the cash value of one magnetic stripe card to another using an iron.[31]

Smart cards are more secure than magnetic stripe cards because they store information in tamper-resistant components and use cryptography to make it difficult to forge. They can be used in off-line transactions [14] and are therefore suitable for the small value transactions that magnetic stripe cards and cash are currently used for. Additional capabilities of smart cards include on-line processing (for larger purchases), as well as storage of personal information and various forms of identification, including digital images, that can only be retrieved by someone who knows the correct password.

---

[30] One important application of magnetic stripe cards is for access to electronically secured buildings and rooms.

[31] Interestingly, however, the University initially switched to smart cards to prevent the forgery, but found that the smart cards were very troublesome to administer (presumably because the technology was still immature and the costs were relatively high) so they switched back to magnetic stripe cards, albeit with a more forgery-resistant solution.

Another area of security risk is electronic mail. Not only does the plaintext nature of electronic mail systems used by most people in most organisations leave them highly susceptible to snooping, vandalism, and slander, but with executable attachments mail systems can potentially spread viruses and other malicious software. Many ideas on how to resolve these problems have been conceived and put into practice, but few are universally deployed and imposed. Electronic mail is a powerful communications tool that should be used only with standardised security features.

Part II: Design and analysis of a new electronic voting system

7. Overview

7.1 Context

We have seen that the University of Victoria's voting systems are not automated and that the two models, the *polling booth* system and the *mail-in* system, suffer from security flaws, operational concerns, and other undesirable characteristics. The main threats to security involve the anonymity of the voter and the accountability of the election administrators, but multiple voting, espionage, vote buying, and unauthorised voting are also possible. There are operational concerns, including proper counting and tabulation of votes. Aesthetically undesirable characteristics include inconvenience, complexity, slow response, inflexibility, and costly operation. These conditions negatively affect voters' satisfaction with the systems, the versatility of the systems, and the frequency with which elections can be held.

7.2 Motivation for electronic voting at the University of Victoria

The aesthetic and functional shortcomings of the University of Victoria's present voting systems provide incentive to study alternatives. The value of an improved system will be highest among those who participate in faculty and departmental elections since, as discussed previously, these voters consider the elections to be much more significant than do, for example, students who may participate in student body elections. We prefer, then, to concentrate on faculty and departmental elections to prove the worth of electronic voting systems, and look at replacements for the mail-in system that is

currently used for these elections. Further discussion of the polling booth system will be limited to some high-level comparative references as appropriate.

This focus on the mail-in voting system frees us from having to seek a general purpose solution for the two different conventional systems. It is also logical, in a way, since nearly every electronic voting system allows voters to complete and submit ballots from remote locations without relying on physical presence to ascertain the legitimacy of the vote. This is similar to the mail-in system but dissimilar to the polling booth system, so electronic voting systems can be more directly compared to the mail-in system..

There is further incentive for pursuing an electronic voting solution to replace the conventional mail-in system at the University; here, desktop computer equipment is common and user savvy with the technology is high, especially in technology-oriented groups such as the Faculty of Engineering. We use the Department of Computer Science, where all faculty members have network-connected computers, as a model for studying the merits and the concerns of a computer-based voting system.

7.3 Contributions

We present several electronic protocols, two of which were devised by the author. We describe the protocols, identify their merits and drawbacks, compare them, and then make recommendations about which protocols to select. One of the two new protocols, the *Improved Two-Agent* protocol, is an enhancement of a fairly simple electronic voting protocol that is intended to replace the mail-in voting system. It is structurally very similar to the mail-in system but, using computer and encryption technology, improves on several aspects of the mail-in system. The other new protocol, the *SAVE* protocol, is decidedly dissimilar to the mail-in system. It achieves substantial improvements in security and, for small electorates, is also aesthetically superior.

8. Formal description of voting protocols: notation

Before comparing the various electronic voting protocols to each other and to the mail-in protocol in place at the University of Victoria, we need to formalise how the protocols are presented in order to permit concrete comparisons. We define a notation to achieve this and to keep the descriptions terse. Table 3 lists the operations used in this notation.

Table 3. Operations used in the voting system notation

Operation	Explanation	Example
identity	Gives the identity of the person associated with the parameter; the parameter can be of many different types	*identity*(x)
knows	The left operand, which is an election participant, knows the right operand.	x *knows* y
vote	Completes a blank ballot, which is the argument.	$vote_i$(x)
blind	Disguise the contents of the argument so that they can be operated on but cannot be understood.	*blind*(x)
→ ("gets")	The left operand, which is a data object, is sent to the right operand, which is an election participant.	*x* → *y*

The following notes describe the rules of the notation:

1. Each election has (i) an *electorate* (ii) an *election administration*, (iii) a *ballot*, (iv) a *protocol*, and (v) a set of *protocol-enabling tools*.
2. The electorate is the group of registered voters and is represented by the symbol E. The electorate has size |E| but, for convenience, the size of the electorate can also be stated as n. Note that some voting protocols may allow registered voters to abstain

from voting. In this case, E does not necessarily represent the group of *participating* voters as well, except after the set of participating voters has been determined.

3. Registered (or participating) voters are represented in general by the symbol V_i; specifically, the notation V_i is meant to represent the permanent identification (e.g., name) of some registered voter Therefore:

$$E = \{V_i: i = 0..n\}$$

4. The election administration may consist of any of (i) a registrar, represented by the symbol R, (ii) a validator, represented by the symbol A, (iii) a collector, represented by the symbol C, and (iv) a tallier, represented by the symbol T. Sometimes a single *central agent* handles the roles of several of these agents.
5. The ballot is a data item which has a physical representation (a paper form) in conventional voting systems and a logical representation (a bit pattern) in electronic voting systems; it is represented by the symbol B when not completed, and by the symbol B_i when completed by voter V_i.

$$\text{e.g., } B_i = vote_i(B)$$

6. Protocol-enabling tools include sealing tools (i.e., physical envelopes, ciphers, etc.), represented by the symbols P for the (normally) public component and S for the correspondingly secret component, a message digest tool, represented by the symbol D, where applicable, as well as any special media used for point-to-point communications. The sealing tools include subscripts that indicate their ownership.

$$\text{e.g., } P_T \text{ represents the public sealing tool of the tallier}$$

7. Operations performed by protocol-enabling tools appear as the symbolic representation of the enabling tool, followed by the data it acts on, in parentheses

$$\text{e.g., data X sealed with the validator's public sealing tool appears as } P_A(X).$$

8. Combinations of data may be treated as a single object by enclosing them in angle ('<', '>') braces. This is helpful when several individual data items must be sealed or extracted, signed or verified, because these operations take one argument only.

e.g., $P_A(<X_1, X_2, X_3>)$ is an example of a message created by concatenating the messages X_1, X_2, and X_3, then encrypting them with the validator's public key

9. Negation (where applicable) is denoted by ~~strikethrough~~ and normally applies to functions as appropriate (e.g., ~~*knows*~~)

Notes about cryptography:

Unless otherwise stated, all public key ciphers used are capable of encryption and decryption as well as signing and verifying signatures. The ciphers consist of two algorithms: one used for *encryption* and the other used for *decryption*. Keys are generated in pairs: an *encryption key*, used with the encryption algorithm, and its *companion* key, a *decryption* key, used with the decryption algorithm. When used for signatures, the keys and their corresponding algorithms are applied in the reverse order, so the *decryption* key becomes the *signing* key and the *encryption* key becomes the *verification* key. In either case, the processing done with one key and algorithm is un-done with its companion key and algorithm.

In practice, one key (normally the encryption key) is divulged and is thus referred to as the *public* key, while the other key is not disclosed and is referred to as the *secret* key (and, therefore, it is important that the private key cannot be easily deduced from the public key). This arrangement is very useful: if the encryption key is made public then it can be used by anyone to encrypt messages destined for the owner of the key or to verify a signature made by that person; however, only the owner of the public key knows the (secret) companion key which is necessary to decrypt or sign the messages. The two keys together are often referred to as a *public-secret* key pair.

Public key ciphers are assumed to be commutative, so if a message is processed more than once using the same cipher but different keys, it can be recovered by processing it the same number of times with the same cipher using the companion keys, *in any order*. For example, if a message is encrypted with the encryption key of one key pair and is signed with the decryption key of another key pair, it can be recovered by

decrypting it with the decryption key of the first key pair and then verifying (and removing) the signature with the encryption key of the second key pair, and vice-versa.

e.g., $P_A(S_B(S_A(P_B(M) = S_B(P_A(S_A(P_B(M) = M$

for some message M and public-secret key pairs (P_A, S_A) and (P_B, S_B).

8.1 The mail-in voting protocol

Table 4 and Table 5 show the preconditions and the protocol for the mail-in voting protocol. This protocol involves a registrar, a validator, and a tallier, in addition to the electorate. The registrar maintains an eligible voters list and a registered voters list, which may be the same if registration is automatic (i.e., any eligible voter is automatically registered to vote). The registered voters list contains entries for each

Table 4. Mail-in voting protocol preconditions

registrar	has n small unmarked tallier's envelopes; P_T represents the action of sealing the envelopes (together with its contents) and S_T represents the action of opening the envelopes (extracting their contents). has n larger validator's envelopes; P_A represents the action of sealing the envelopes (together with its contents) and S_A represents the action of opening the envelopes (extracting their contents). has n blank ballots B. knows the names, addresses, and signatures of all voters knows the validator's address
voter	has a signature; it is applied to an envelope, is created with S_i, and is verified with P_i, where i refers to a particular voter in {1..n}. knows the validator's address
validator	knows the tallier's address

registered voter, with each entry consisting of the name of the voter and any other identifying information deemed important (e.g., social insurance number, employee number, etc.), and an example signature to verify the voter's signature against. The tallier maintains a tally, represented by the term *tally*, which is a collection of ballots B_i and is initially empty.

Table 5. Mail-in voting protocol

Step	Actor	Description of action	Symbolic representation	Assertions
1	R	Send a voters list with the names and signatures of all voters to the validator	$\{V_i, P_i : \forall i \in \{1..n\}\} \rightarrow A$	A *knows* V_i
2	R	Create ballot packages consisting of a ballot, a validator's envelope and a tallier's envelope, then send one to each voter	$B, P_A, P_T \rightarrow V_i : i \in \{1..n\}$	
3	V_i	Complete the ballot	$B_i = vote_i(B)$	X *knows* B_i *iff* $X = V_i$ X *knows identity*(B_i) *iff* $X = V_i$
4	V_i	Seal the completed ballot in the unmarked tallier's envelope	$P_T(B_i)$	
5	V_i	Seal the tallier's envelope in the validator's envelope	$P_A(P_T(B_i))$	
6	V_i	Sign the validator's envelope, add name, and send the result to the validator	$S_i(P_A(P_T(B_i))), V_i \rightarrow A$	
7	A	Verify the envelope's signature, and update the voters list	$P_A(P_T(B_i))$ $= P_i(S_i(P_A(P_T(B_i))))$	A *knows identity*(V_i)
8	A	Extract tallier's envelope	$P_T(B_i) = S_A(P_A(P_T(B_i)))$	A *knows identity*$(P_T(B_i))$ A ~~*knows*~~ $P_T(B_i)$
9	A	Send tallier's envelope to tallier	$P_T(B_i) \rightarrow T$	
10	T	Extract completed ballot	$B_i = S_T(P_T(B_i))$	A ~~*knows*~~ *identity*(B_i) A *knows* B_i
11	T	Update tally	*tally* = *tally* + B_i	

The protocol begins with the registration phase and, as with most voting systems (especially for smaller voting populations), this phase is somewhat ancillary in that it usually does not have to be carried out in full formality for each election. For instance, at the University of Victoria, students and faculty members are pre-registered to vote by virtue of their status as a registered student or as an employee. The main role of the registrar is to ensure that the eligible and registered voters lists are up-to-date, that the latter is available to the validator, as described in step 1 of Table 5, and that each voter gets a ballot package (step 2). Note that the symbolic representation in step 2 appears to

treat P_A and P_T as physical objects instead of logical operations. The operation of opening a particular type of envelope is associated with the envelope that it applies to.

The balloting phase is next, where voters complete their ballot forms (step 3), seal them inside the unmarked tallier's envelopes (step 4), seal the unmarked envelopes inside the larger validator's envelopes (step 5), then sign the validator's envelopes and send them to the validator (step 6). At this point, because of the double envelope sealing, only the voters can read their ballots. A separate validating phase follows, where the validator confirms the signatures of the voters from the received large envelopes (step 7), opens the envelopes and removes the smaller unmarked envelopes (step 8), then sends the unmarked envelopes to the tallier (step 9).

The validator knows the identities of the voters from the larger signed envelopes but cannot know the votes they cast without opening the smaller unmarked envelopes. Assuming that the unmarked envelopes are sent unopened and unordered to the tallier with no identifying information, the validator can no longer connect votes to voters and the tallier cannot do so either. In this phase, the tallier proceeds to count the votes by opening the unmarked envelopes, removing the ballots (step 10), and adding them to the tally (step 11). Once it completes the count, the tallier publishes the results.

8.1.1 Attacks on the mail-in voting protocol

There are many possible attacks on the mail-in voting protocol. The ballot packages can be tampered with at any point during delivery to the voters from the registrar. They could, for example, be removed, altered, or replaced before the voters receive them. Removal should be easily detected if voters are made aware of upcoming elections and thus expect to receive a ballot package in time for each election. Alteration or replacement could be more difficult to detect. A plausible attack would be to set a phoney return address for the package, leading voters to send their completed ballots to someone other than the validator. If voters do not know the correct address and do not expect to receive confirmation from the validator that their ballots were received, they will not be aware of the treachery. Even if they expect to receive confirmation from the

validator, a fraudulent confirmation could be sent by the perpetrator to the phoney address.

Dangers also exist during the submission of completed ballot packages from voters to the validator. An obvious attack is for someone to intercept a package in the mail in order to learn the voter's choices or to prevent the package from reaching the validator. In the former case, the foul play would be extremely difficult to detect, assuming that the ballot package was carefully reconstructed. In the latter case, the attack would be most likely to succeed if the removed ballot was replaced with a forgery (with different selections, of course).

When the validator receives ballot packages, it could open, or attempt to somehow observe the contents of, the inner envelopes. Essentially, the validator can carry out any of the attacks that could occur during delivery of the ballot packages from the voters. Additionally, it can collude with the tallier by marking the inner envelopes or by ordering them in a particular way before giving them to the tallier. This type of subtle attack might be the only option if the validator and the tallier are under general observation. Finally, the tallier could also be malicious, by discarding or miscounting ballots.

There are techniques to minimise the opportunities for these attacks. In general they are (i) to keep voters informed of the details of each election, (ii) to secure all delivery systems (between registrar and voter, between voter and validator, and between validator and tallier), and (iii) to provide observers to carefully monitor all proceedings. Unfortunately, none of these techniques is an absolute solution; they do not provide absolute security against the attacks they defend against. Their effectiveness generally increases with the amount of resources applied to them, but this leads to increased costs as well. The fact that mail-in voting systems are still susceptible to attack in practice suggests that the need to keep costs down is extremely significant and reaches equilibrium with the need for security long before the system is made *absolutely* secure.

8.2 Electronic voting protocols

Now we examine in detail how electronic voting systems are superior to the conventional mail-in system used for elections at the University of Victoria and select one solution for demonstration. To this end, we first describe several electronic voting protocols, then take the best candidates and show how they improve on the conventional mail-in protocol, and then compare the chosen candidates to select the best system for our environment. Note that the order in which the voting protocols are presented merely reflects the order in which they were analysed. Each of these protocols was intended for deployment on computer systems (or perhaps special-purpose devices) but other devices, such as the common telephone, have been proposed for electronic voting systems. We do not consider these alternatives to be desirable because they have highly contentious security characteristics (see Appendix D - Voting by telephone - on page 121).

8.2.1 One-agent protocol

Perhaps the most straightforward electronic voting protocol which could replace the current conventional mail-in system is one which utilises a single central facility to handle all processing of votes, including registration, collection, validation, tallying, and tabulating. This is similar to the mail-in protocol where the registrar, the validator, and the tallier are the same person. In the electronic protocol, ballot forms are enhanced data structures or objects, a voter software application assists the human voter in completing

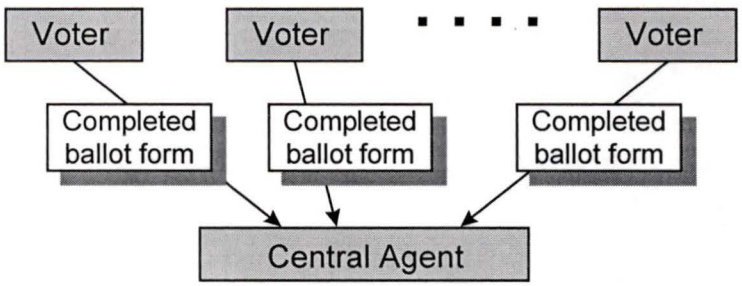


Figure 10. One-Agent voting system

the ballot, the registrar, validator, and tallier are software applications, the network functions as the mail (message) delivery system, and cryptographic functions are used for signatures and for sealing messages.

The basic procedure for this protocol is for voters to complete their ballot forms and send the completed ballot forms to the central facility where they are counted (see Figure 10). For privacy, voters should submit their ballots along a secure channel which is established using the public key of the central facility. To prevent unauthorised voting or multiple voting, voters should be required to identify themselves with a digital signature.

8.2.2 Two-agent protocol

The main problem with the one-agent protocol is that the central facility knows how everyone voted and therefore must be completely trusted. The reason is that the central facility is responsible for both validation and tallying[32], with the first procedure requiring the voter's identity and the second requiring knowledge of the voter's vote.

One approach to solving this problem is to distribute the responsibilities among two separate agents; a validator and a tallier. The simplest scheme has voters encrypt their ballots with the tallier's public key and send them, together with some form of

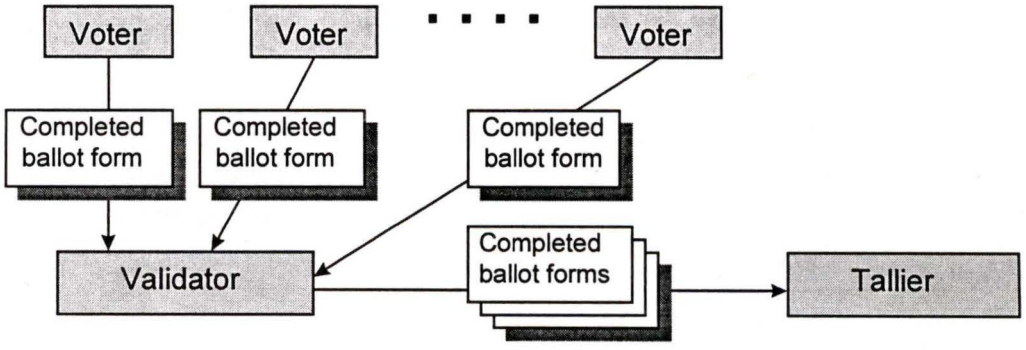


Figure 11. Two-agent voting protocol

[32] As well as other responsibilities such as registration.

identification, to the validator. The validator confirms the identity and the registration of the voter by verifying the voter's signature or by some other cryptographic technique. The validator then strips off any identifying information from the encrypted ballot and sends the ballot to the tallier. The tallier assumes that any vote sent from the validator is valid and therefore should be counted (see Figure 11). It can remove the encryption of the ballot with the tallier's public key.

8.2.3 Improved one-agent protocol

The main problem with the two-agent protocol is that the validator and tallier can collude to reveal how voters voted. Also, there is nothing to stop the tallier or, in some cases, the validator from adding their own votes to the count. Furthermore, there is no way for voters to confirm or correct the tabulation of their votes or anyone else's.

The two-agent protocol is susceptible to collusion because the validation and the tallying of a vote occur during a single transaction by the voter. Since validation requires voter identification and tallying requires ballot exposure, the two necessary components to link a voter with a vote are available during this single transaction. A solution is to require separate transactions by the voter for validation and submission of the vote, where the voter cannot be identified with **both** transactions. Since validation

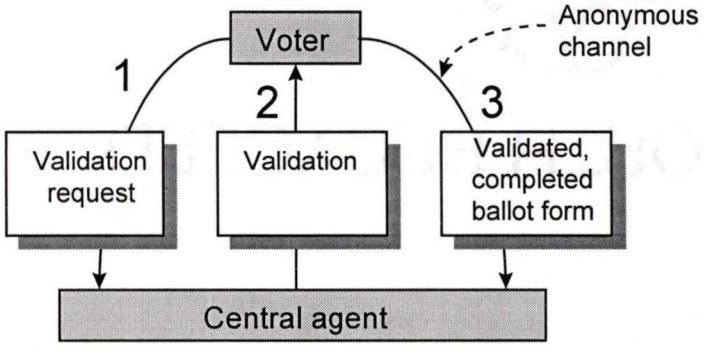


Figure 12. Improved one-agent voting protocol

requires identification of the voter, the submission transaction must be anonymous. The simplest[33] way to achieve anonymous vote submission is to provide voters with anonymous message delivery channels to the central agent. Figure 12 illustrates the basic ideas of the improved one-agent protocol which implements these ideas. The main challenge in this scheme is to provide a means for the central facility to validate a completed ballot that it can subsequently verify, without seeing the contents of the ballot. There are different technical approaches to solving this problem that Figure 12 does not illustrate in detail.

Nurmi, Salomaa, and Santean [30] designed a voting scheme that requires only a single facility for voter authentication and vote tabulation. It uses an *All-or-Nothing Disclosure of Secrets* (ANDOS) protocol [38] to distribute secret identification numbers from the central facility to registered voters without the central facility or any voter knowing who received which number.[34] Voters generate a public-secret key pair to be used only for the election, then anonymously send their identification numbers and votes (the latter encrypted with the secret key) to the central facility who subsequently publishes the encrypted vote as a verification of receipt. Upon verifying their receipts, voters anonymously send their identification numbers together with the public key to the central facility who proceeds to decrypt the votes. The central facility publishes the results for all to verify that their votes were counted correctly.

A more popular approach is to use a *blind signature* protocol [11], where the signer cannot read the contents of the message being signed. A physical analogy of this concept is the following: the form to be signed is covered with a sheet of carbon paper and placed inside an envelope, so that no one can read the contents of the form but, if the envelope is signed, the signature is transferred onto the form by the carbon paper. With cryptography, the client first *blinds* the message to be signed by encrypting it, then sends it to the signer. The signer then digitally signs the encrypted message and returns it to

---

[33] … although not necessarily the easiest …

[34] These numbers should be posted in advance and should be digitally signed by the central facility to prevent election fraud.

the client. Finally, the client removes the *blinding factor* by decrypting the message, then verifies the signature with the public key of the signer. The result is equivalent to the signer having directly signed the original message, except that the signer was unable to read the message because it was *temporarily* encrypted (see Figure 13). Note that the blind signature process requires the blinding operation (i.e., the public-key algorithm) to have a commutative property.

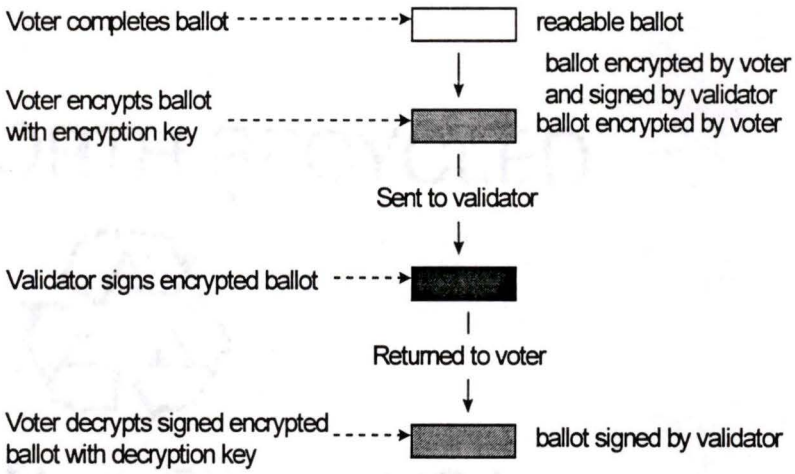


Figure 13. The blind signature process

Fujioka, Okamoto, and Ohta [21] designed a scheme that uses blind signatures and thereby does not require secret identification numbers. Each voter generates two public-secret key pairs: one that is used for the blinding of the ballot in the validation stage and another that is used for encrypting the signed ballot in the tallying stage. Voters blind their ballots with the encryption key of their first generated key pair and send them to the central facility together with identifying information. The central facility verifies the identification of the voters, signs their blinded ballots, and returns the signed blinded ballots to the voters. Each voter removes the blinding factor from the signed blinded ballot, encrypts the resulting signed ballot with the encryption key of the second key

pair, and sends the encrypted signed ballot to the central facility together with a secret identification number along an anonymous channel. The central facility signs the signed encrypted ballots and posts them together in a publicly-viewable list. After the voters have opportunity to view the list and verify the presence of their entries, each voter anonymously sends the decryption key from the second key pairs, together with an indication of which encrypted signed ballot it decrypts, to the central facility who decrypts the votes and tallies the results.

Note that the purpose of the two separate steps in the tallying stage is so that voters do not reveal their votes to the central facility until the central facility has given evidence (i.e., the publicly-viewable list of voters' messages signed by the facility) that it has properly received the vote messages. The extra step does not seem necessary. It should be sufficient for voters to send their validated (signed) ballot messages directly to the central facility along an anonymous channel. The signature of the validator on the ballot should suffice as proof of the voter's decision, should the need for a challenge of the tally results arise (see Appendix C - Challenging a miscounted vote - on page 120).

8.2.3.1 The Sensus protocol

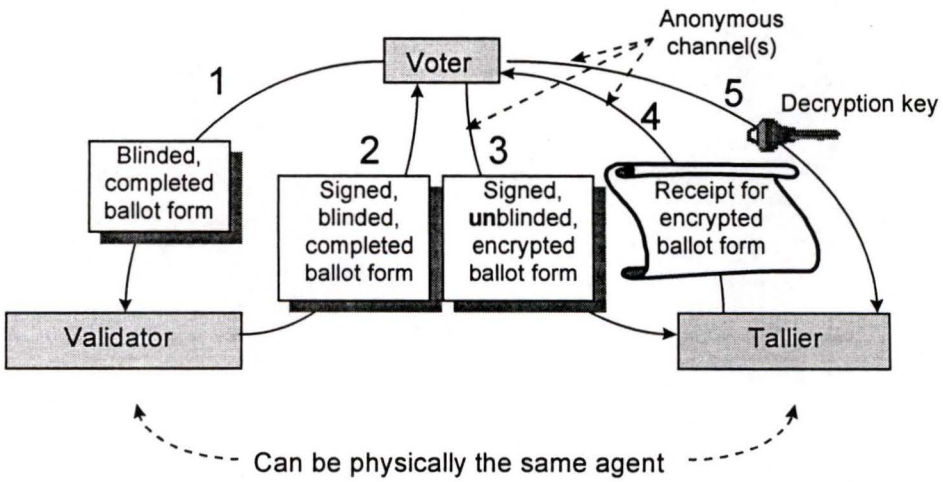


Figure 14. Sensus voting protocol

Cranor and Cytron [15] extended the Fujioka, Okamoto, and Ohta protocol and implemented it as a fully functional electronic voting system. Called **Sensus**, this system's protocol satisfies all functional qualities for a voting system, given a set of assumptions. Figure 14 illustrates the Sensus voting protocol. It is discussed because it is an excellent example of an workable electronic voting protocol and yet, like all such protocols, still has some potential problems that must be guarded against to ensure its security.

Sensus improves on the Fujioka, Okamoto, and Ohta scheme because it specifies that the tallier must send a receipt to the voter immediately after the voter submits a certified encrypted ballot to it. This frees voters in the Sensus protocol to immediately provide the tallier with the decryption key to extract the vote (under the Fujioka, Okamoto, and Ohta protocol, voters must wait for the tallier to post a list of the encrypted ballots after all the ballots have been submitted in order to satisfy themselves that their votes were properly received). There are also some cosmetic improvements to the voter interface.

Table 6 and Table 7 list the preconditions and the protocol of the Sensus voting system. The Sensus protocol description refers to an individual tallier and validator, instead of a single central facility, and recommends that they be deployed on distinct systems. In reality, however, the two agents can actually be on the same system (and the same software application) without any significant threat to security, just as in the basic improved one-agent protocol.

Table 6. Sensus voting protocol preconditions

registrar	knows the names, public keys, and network addresses of all voters
	knows the validator's network address
voter	has a public-secret key pair (Pi, SI)
	knows the validator's public key and network address
	knows the tallier's public key and network address
validator	has a public-secret key pair (Pv, SV)
tallier	has a public-secret key pair (Pt, ST)
	knows the validator's public key

Table 7. Sensus voting protocol

Step	Actor	Description of action	Symbolic representation	Assertions
1	R	Acquire each voter's public key	P_i: $\forall$ i $\in$ {1..n}	R *knows identity*(P_i) : $\forall$ i $\in$ {1..n}
2	R	Generate a unique ID number for each voter	I_i: $\forall$ i $\in$ {1..n}	R *knows identity*(P_i) : $\forall$ i $\in$ {1..n}
3	R	Send the ID number to each voter along with the ballot form	$<I_i, B> \rightarrow V_i : \forall i \in \{1..n\}$	
4	R	Send a voters list with the names, ID numbers, and public keys of all voters to the validator	$\{<V_i, I_i, P_i> : \forall i \in \{1..n\}\} \rightarrow A$	A *knows identity*(P_i)
5	V_i	Complete the ballot form	$B_i = vote_i(B)$	X *knows* B_i *iff* X == V_i X *knows identity*(B_i) *iff* X == V_i
6	V_i	Generate an election-specific key pair (not shown) for the blinding process and another (shown) for submitting the ballot to the tallier	P_i^*, S_i^*	X *knows* P_i^*, S_i^* *iff* X == V_i
7	V_i	**Seal** the completed ballot with the election-specific encryption key and digest the result.	$D_i = digest(S_i^*(B_i))$	
8	V_i	**Blind** the digest	$D_i^b = blind(D_i)$	
9	V_i	**Sign** a copy of the blinded digest and create a message composed of it, the unsigned blinded digest, and the voter's ID number	$<D_i^b I_i, S_i(D_i^b)>$	
10	V_i	**Seal** the message with the validator's public key and send the result to the validator	$P_A(<D_i^b, I_i, S_i(D_i^b)>) \rightarrow A$	
11	A	**Extract the** message contents	$D_i^b, I_i, S_i(D_i^b) = S_A(P_A(<D_i^b, I_i, S_i(D_i^b)>))$	A *knows identity*(I_i) A *knows* P_i A ~~*knows*~~ D_i
12	A	**Verify** the voter's signature on the blinded digest	$D_i^b = P_i(S_i(D_i^b))$	
13	A	Update a local copy of the voters list to indicate that the voter has voted		
14	A	**Sign** the blinded digest	$S_A(D_i^b)$	
15	A	**Seal** the signed, blinded digest with the voter's public key and send the result to the voter	$P_i(S_A(D_i^b)) \rightarrow V_i$	
16	V_i	**Extract** the message contents	$S_A(D_i^b) = S_i(P_i(S_A(D_i^b)))$	
17	V_i	**Remove** the blinding factor from the signed, blinded digest	$S_A(D_i) = unblind(S_A(D_i^b))$	
18	V_i	**Verify** the validator's signature on the digest	$D = P_A(S_A(D_i))$	

19	V_i	**Seal** the ballot with the election-specific secret key, then create a message comprised of it and the signed digest	$<S_A(D_i), S_i^*(B_i)>$	
20	V_i	**Seal** the message with the tallier's public key and send it to the tallier along an anonymous channel	$P_T(<S_A(D_i), S_i^*(B_i)>) \rightarrow T$	
21	T	**Extract** the message contents	$S_A(D_i), S_i^*(B_i)$ $= S_T(<S_A(D_i), S_i^*(B_i)>)$	T ~~*knows*~~ *identity*(M_3) T ~~*knows*~~ P_i^* T ~~*knows*~~ B_i
22	T	**Verify** the signed digest against the sealed ballot	$P_A(S_A(D_i)) =$ *digest*($S_i^*(B_i)$)	
23	T	**Sign** the sealed ballot	$S_T(S_i^*(B_i))$	
24	T	Generate a receipt number and add the receipt number and the sealed ballot to a hash table	K_i	
25	T	Return the receipt number and the signed, sealed ballot to the voter along the anonymous channel	K_i, $S_T(S_i^*(B_i)) \rightarrow V$	
26	V_i	**Verify** the tallier's signature on the signed, sealed ballot	$S_i^*(B_i) = P_T(S_T(S_i^*(B_i)))$	
27	V_i	Send the receipt number and the election-specific encryption key to the tallier along an anonymous channel	K_i, $P_i^* \rightarrow T$	
28	T	Consult the hash table using the receipt number to retrieve the sealed ballot		
29	T	**Extract** the ballot using the election-specific decryption key	$B_i = P_i^*(S_i^*(B_i))$	T *knows* B_i T ~~*knows*~~ *identity*(R#) T ~~*knows*~~ *identity*(P_i^*)
30	T	Update the vote tally and add the receipt number and election-specific encryption key to a log		

All actors in the Sensus protocol, including voters, the tallier, and the validator (and the registrar) have a public-secret key pair and the registrar knows all of the public keys. The actors all use the same public key cipher (e.g., RSA) and message digest function (e.g., MD5). The public key cipher is commutative and therefore can be used in a blind signature protocol. Voters each have access to an anonymous channel for communicating with the tallier.

The authors of the Sensus protocol list the following assumptions:

- Voters have access to an anonymous communications channel to the tallier over which the tallier is unable to learn the sender's identity and yet is still able to send a reply directly to the sender.
- Voters' computer systems are secure from eavesdropping or tampering.
- The tallier and the validator do not collude to expose the identity of the voter by timing their communications with the voter.
- All encryption algorithms are sufficiently strong to render their compromise unfeasible.

They could have listed more, such as:

- The tallier does not release results to anyone early.
- The tallier will correct any miscounted votes if challenged by the affected voters.
- Communication channels are error-free or at least adequately error-correcting.
- The public keys of all parties are correctly known by those who need to know them.
- The software is trustworthy (i.e., the code is bug-free, the protocol is properly implemented, and the software has no malicious function).
- Voters are honest and ethical (if bribery and coercion are concerns).

The registrar in the Sensus protocol produces a voters list consisting of entries for each voter that contain the name of the voter, an assigned voter id number, and the voter's public key. The registrar also makes an electronic ballot publicly available for voters to retrieve. As with any electronic voting protocol, the ballot can be a fully functional object that handles the user interface and input processing or it can be a simple data structure with rules for how it may be completed. By the end of step 4, each voter has an identity number for the election and the validator has a copy of the voters list. From this point onward, use Figure 15 as a visual reference for the cryptographic steps in the protocol.

Each voter completes the ballot in step 5, generates a public-secret key pair for ballot encryption and another for a blind signature in step 6, and encrypts the ballot with the encryption key of the ballot encryption key pair (steps A-C in Figure 15). Next, the

voter computes a digest from the encrypted ballot in step 7, blinds the digest using the encryption key of the other key pair in step 8, then signs a copy of the blinded digest in step 9 (steps D-F in Figure 15). Finally, the voter encrypts a message comprised of the signed blinded digest, the original blinded digest, and the voter's identification number, with the validator's public key, then sends it to the validator in step 10.

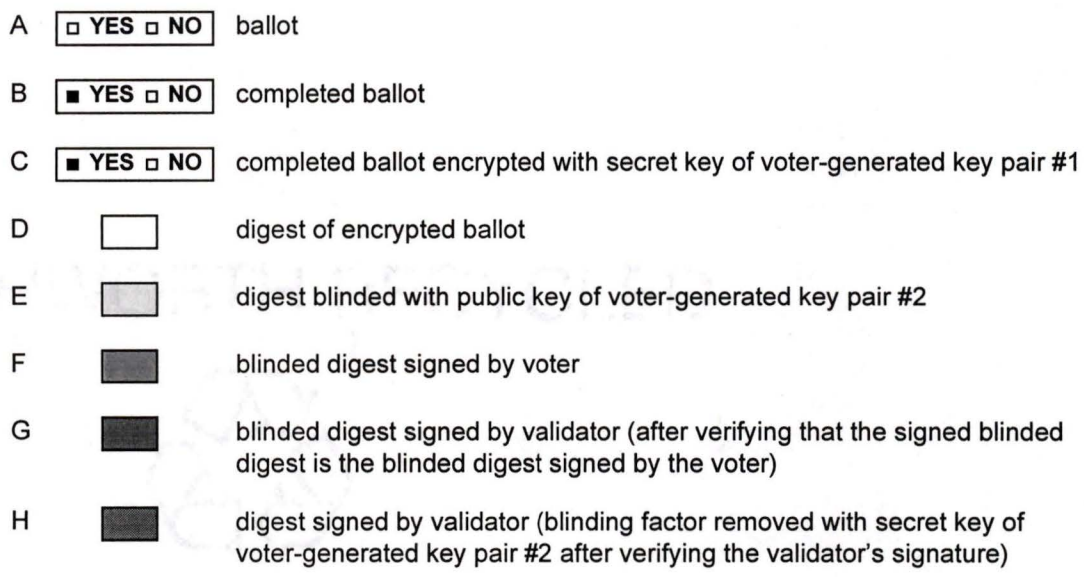


Figure 15 Sensus ballot encryption steps

The purpose of the signed copy made in step 9 is so that the validator can verify the voter's identity, as claimed by the included identification number. The validator removes the voter's signature from the blinded digest signed by the voter and compares the result to the blinded digest; they should be equal. This is done in step 12, after the validator decrypts the message from the voter (step 11). After verifying the voter's identity, it is important to note that the validator does not know anything about the original digest (or, for that matter, the ballot) because of the blinding factor applied to it by the voter. The validator then indicates that the voter has been validated (step 13), signs the blinded digest (step 14), encrypts it with the voter's public key, and returns it to the voter (step 15).

The voter extracts the message received from the validator (step 16) using her secret key and removes the blinding factor using the decryption key from the first election-specific key pair (step 17). The result is precisely the original digest but with the validator's signature (the voter can easily verify this by removing the validator's signature using the validator's public key and comparing the result with the original digest; they should be equal (step 18)). Step G in Figure 15 shows the blinded digest signed by the validator, and step H shows the signed digest with the blinding factor removed. The voter then encrypts the ballot with the encryption key of the second election-specific key pair and bundles it with the signed (by the validator) digest (step 19). Finally, the voter encrypts the resulting message with the tallier's public key, and sends the result to the tallier along an anonymous channel (step 20).

The tallier extracts the message received from the validator using its secret key (step 21), then verifies the encrypted (by the voter) ballot against the signed (by the validator) digest by digesting the encrypted ballot and removing the signature on the signed digest using the validator's public key; the results should be equal (step 22). Referring to Figure 15, the signed digest of step H sent by the voter to the tallier can be easily reduced to the digest in step D because the tallier knows the validator's public key. Since the voter also sent the encrypted ballot of step C, the tallier, who has the message digest function, can compute the digest of step D and compare it to the previously computed digest. The tallier signs the encrypted (by the voter) ballot (step 23), generates a receipt number (step 24), and sends the two back to the voter (step 25). It also stores a copy of the encrypted ballot with the receipt number as an index.

The voter receives the signed (by the tallier) encrypted ballot and the receipt number, verifies the tallier's signature (step 26), and replies along an anonymous channel with the receipt number and the key that will decrypt the encrypted ballot (step 27). The tallier retrieves the encrypted ballot using the receipt number as the index key (step 28), uses the decryption key to reveal the ballot (step 29), then updates the tally (step 30).

8.2.4 Improved two-agent protocol

The main challenge with the Sensus protocol is establishing an anonymous channel. The *Two-Agent* protocol avoids the need for an anonymous channel because voters are validated by the validator and do not communicate directly with the tallier; the validator does this on their behalf. We would like to retain this property while adding a means for voters to verify that their votes were counted correctly and to challenge any incorrect counts. Receipts are an important element added to the basic two-agent protocol that helps to achieve these features (see Figure 16).

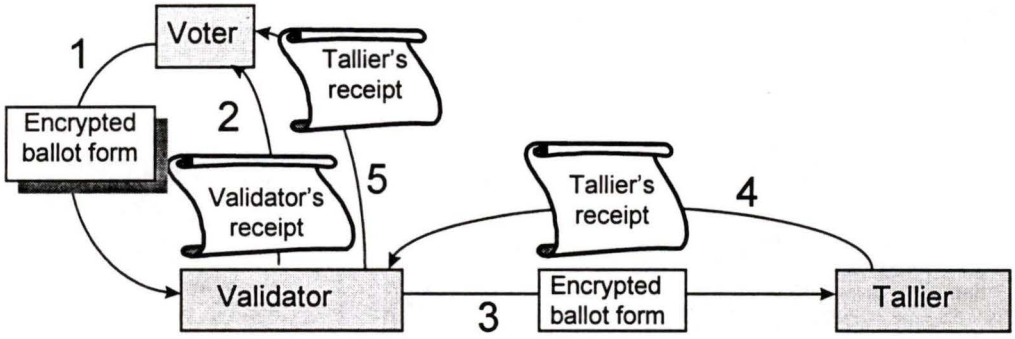


Figure 16. Improved two-agent voting protocol

Table 8 and Table 9 list the preconditions and the protocol of the improved two-agent voting system. As with the basic two-agent protocol, this protocol has a validator and a tallier, as well as a registrar, and the voter's ballot is submitted to the tallier

Table 8. Improved two-agent voting protocol preconditions

registrar:	knows the names, public keys, and network addresses of all voters
	knows the validator's network address
voter:	has public-secret key pair (P_I, S_I)
	knows the validator's public key and network address
	knows the tallier's public key and network address
validator:	has a public-secret key pair (P_V, S_V)
	knows the tallier's public key and network address
tallier:	has a public key pair (P_T, S_T)
	knows the validator's public key and network address

through the validator. The voters, the validator, and the tallier use one public-key cipher (algorithm) for signatures and for encryption. The registrar produces a voters list of all voters' names and their public keys. It also provides a ballot for voters to retrieve.

Table 9. Improved two-agent voting protocol

	Actor	Description of action	Symbolic representation	Assertions
1	R	Acquire each voter's public key	P_i: $\forall$ i $\in$ {1..n}	R *knows identity*(P_i) : $\forall$ i $\in$ {1..n}
2	R	Publish a voters list with the names and public keys of all voters	{<V_i, P_i> : $\forall$ i $\in$ {1..n}}	A *knows identity*(P_i)
3	V_i	Complete the ballot	B_i = *vote*$_i$(B)	X *knows* B_i *iff* X == V_i X *knows identity*(B_i) *iff* X == V_i
4	V_i	Generate an election-specific random ID number and an election-specific key pair.	I_i*, P_i*, S_i*	X *knows* I_i*, P_i*, S_i* *iff* X = V_i
5	V_i	Create a ballot message with the completed ballot, election-specific ID number, and election-specific public key	B_i^m = <B_i, I_i*, P_i*>	
6	V_i	**Seal** the ballot message with the tallier's public key, then combine with the voter's name	<V_i, $P_T(B_i^m)$>	
7	V_i	**Seal** the result with the validator's public key, then send it to the validator	P_A(<V_i, $P_T(B_i^m)$>) → A	
8	A	**Extract** the message contents	V_i, $P_T(B_i^m)$ = $S_A(P_A(<V_i, P_T(B_i^m)>))$	A *knows identity*(M_1) A ~~*knows*~~ S_T ∴ A ~~*knows*~~ B_m ∴ A ~~*knows*~~ B_i A ~~*knows*~~ I_i* A ~~*knows*~~ P_i*
9	A	Update the validator's log to indicate that the voter has voted		
10	A	**Sign** the contents of the voter's message to create a validation receipt	$K_{V(i)} = S_A(<V_i, P_T(B_i^m)>)$	optional
11	A	**Seal** the validation receipt with the voter's public key and send it to the voter	$P_i(K_{V(i)})$ → V_i	optional
12	A	**Sign** the sealed ballot message and send it to the tallier	$S_A(P_T(B_i^m))$ → T	

13	T	**Verify** the validator's signature	$P_T(B_i^m) = P_A(S_A(P_T(B_i^m)))$	
14	T	**Extract** the contents of the ballot message	B_i, I_i^*, $P_i^* = S_T(P_T(B_i^m))$	T *knows* B_i T ~~*knows*~~ *identity*(B_m) $\therefore$ T ~~*knows*~~ *identity*(B_i) T ~~*knows*~~ *identity*(I_i^*) T ~~*knows*~~ *identity*(P_i^*)
15	T	Update the tally and the tallier's log to indicate that a valid voter has voted		
16	T	**Sign** the ballot message	$S_T(B_i^m)$	
17	T	**Seal** the signed ballot message with the voter's election-specific public key and send it to the validator.	$P_i^*(S_T(B_i^m)) \rightarrow A$	
18	A	Update the log to indicate that the voter's vote was counted		
19	A	Send the message to the voter	$P_i^*(S_T(B_i^m)) \rightarrow V_i$	A ~~*knows*~~ P_i^* $\therefore$ A ~~*knows*~~ B_m $\therefore$ A ~~*knows*~~ B_i A ~~*knows*~~ I_i^* A ~~*knows*~~ P_i^*
20	V_i	**Extract** the contents of the validation receipt	$K_{V(i)} = S_i(P_i(K_{V(i)}))$	optional
21	V_i	**Verify** the validator's signature	V_i, $P_T(B_i^m) = P_A(K_{V(i)})$	optional
22	V_i	**Extract** the message contents	$S_T(B_i^m) = S_i^*(P_i^*(S_T(B_i^m)))$	
23	V_i	**Verify** the tallier's signature	$B_i^m = P_T(S_T(B_i^m))$	

It appears at a high level to be a relatively simple protocol: (i) a voter submits an encrypted ballot to the validator, (ii) the validator confirms the eligibility of the voter and forwards the ballot to the tallier, (iii) the tallier decrypts the ballot, updates the tally, and returns a signed copy of the encrypted ballot to the validator, and (iv) the validator forwards the tallier's receipt to the voter, possibly preceded by a receipt directly from the validator.

As usual, the registration stage involves the registrar building an updated voters list (step 1) and sending it to the validator. Actually, in this protocol, the registrar simply makes the voters list publicly available to all (step 2).

Each voter retrieves and completes a ballot (step 3) and generates an election-specific public-secret key pair and random identification number (step 4). The voter

bundles the ballot, the identification number, and the encryption key from the key pair into a message (step 5), then encrypts the message with the tallier's public key and bundles it with the voter's name (step 6). This message is encrypted with the validator's public key and is sent to the validator (step 7).

The validator decrypts each message received and verifies the voter's identity (step 8), then updates the log to indicate that the voter has voted (step 9). Next, the validator signs the decrypted message to create a validation receipt (step 10), then encrypts the receipt with the voter's public key and returns it to the voter (step 11). Note that the voter can begin to decrypt (step 20) and verify (step 21) the validation receipt as soon as it arrives. The validator signs the encrypted ballot and sends it to the tallier (step 12).

The tallier verifies and removes the signature of the validator from the message received (step 13) and extracts the ballot message created in step 5 using its secret key (step 14). The tallier assumes that the ballot message is good because of the validator's signature so it adds the ballot's vote to the tally (step 15). The tallier signs the ballot message to create a tally receipt (step 16), then encrypts it with the election-specific public key provided in the ballot message and returns it to the validator (step 17).

The validator simply indicates that the voter's vote was counted, assuming all went normally according to the tallier (step 18), and forwards the encrypted tally receipt to the voter (step 19). The voter can decrypt the tally receipt using the decryption key of the election-specific key pair (step 22) and verify the tallier's signature using the tallier's public key (step 23).

8.2.5 Zero-agent protocol

In any voting system, security is no better than the honesty of the entire electorate. For example, if all other voters conspire against one, they can determine how that voter voted simply by combining their votes and determining the difference from the posted results. Fortunately, the likelihood of this happening is extremely slim for at least two reasons, especially in elections with a large, diverse electorate. First, all of those voters would have to reveal their votes to each other and, second, each of the conspiring voters

would have to trust that all of the others are truthful about how they voted (unless, as with some electronic systems, they can prove that they voted in a particular way).

There is an interesting protocol [26] that is carried out entirely among the voters. We classify this as a *peer voting system* and refer to it as a zero-agent protocol since it requires no external agent to conduct the election; it could also be considered an *N-agent* protocol since all (*N*) voters are effectively agents in the process. The protocol shows that voters can protect their anonymity by encrypting their votes with other voters' public keys so that all voters must collaborate in order to reveal any single voter's vote. At the same time, the protocol guards against misuse such as multiple voting and unauthorised voting.

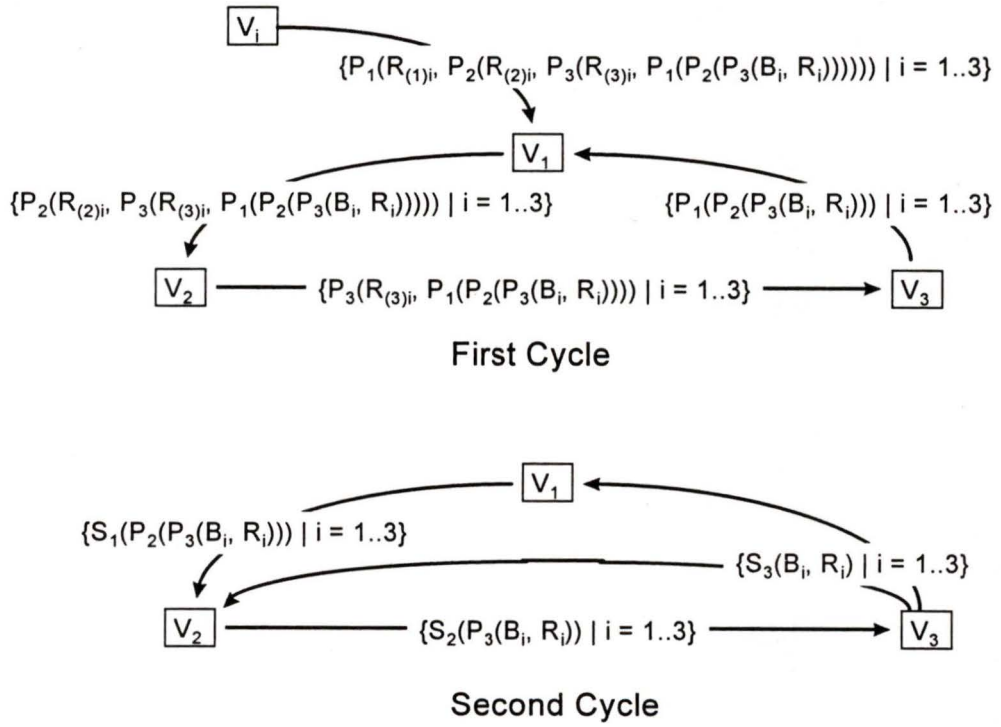


Figure 17. Zero-agent voting protocol

In the zero-agent protocol, each voter is responsible for acquiring a voters list which contains the names, public keys, and addresses of all participating voters (it is assumed that in practice a registrar would facilitate these administrative duties). Note that we

refer to *participating* voters instead of *eligible* or *registered* voters for the voters list. This is because the zero-agent protocol requires the participation of all voters who have expressed the intention to vote (i.e., no backing out).

Figure 17 shows the flow of the zero-agent voting protocol for a very simple example election involving only three voters. This figure contains more details about the message flow because the protocol is difficult to explain verbally. Note that there are two distinct stages in which sets of messages are processed and sent from one voter to another in a prescribed order; these stages are referred to as *cycles*. The first cycle is preceded by an initial set-up stage and the second cycle, which immediately follows the first cycle, is followed by a finish-up (tallying) stage. A description of the protocol follows, while a more thorough explanation of the security issues is provided in section 9.1.3.

In the set-up stage, voters agree upon a voter's list which indicates the participants, their public keys, and other information (e.g., their network addresses). Each voter V_i then votes, creating a ballot message B_i, generates and adds a random number R_i to the ballot message, and encrypts the result with **all** voters' public keys. Each voter then repeatedly generates and prepends a random number $R_{(x)i}$, then encrypts the result with voter V_x's public key, for $x=n..1$ (assuming an electorate of n voters). An example of the resulting message for the three-voter-electorate example in Figure 17 is shown at the top of that figure. As indicated in the figure, all voters send these initial messages to voter V_1(except, of course, voter V_1), from where the first cycle will begin.

The purpose of the first cycle, which is described as the *ballot scrambling* cycle, is to re-order the messages containing the ballots so that none of the voters can trace any but their own to any other voter. As seen in Figure 17, each voter decrypts each message and removes the outermost random number, then sends the resulting set of messages to the next voter. What is not shown in the figure is that each voter first verifies that her vote is among the set of messages and, after doing so and decrypting the messages, re-orders the resulting messages randomly. At the end of the ballot-scrambling cycle, voter V_1 has the set of messages ready for the second cycle.

Voter V_1 begins the second cycle by decrypting each message with her secret key, then signing and sending the results to voter V_2. Beginning with voter V_2, each voter verifies the signature of the previous voter, decrypts the messages with her secret key, then signs the results and sends them to the next voter. When the messages return to voter V_1, the second cycle is complete. All voters must verify that the message containing their ballot is intact. The purpose of the second cycle, referred to as the *validation* cycle, is to provide voters with confidence that their ballots are unaltered. Voters certify the messages containing the ballots at each point in the cycle. If a voter finds that her ballot is in any of the messages she receives during this cycle, she can terminate the election immediately. Otherwise, if her ballot is not among the final messages at the end of the cycle, she can prove this with the signed messages she received earlier from the previous voter by backtracking (re-encrypting) with the participation of each subsequent voter (and thereby also revealing the culprit).

In the final *tallying* stage, voter V_1signs all of the messages and sends them to each of the other voters. All voters can then verify the signature of voter V_1 and that their ballots are among the set of messages. Voters can then independently tally the votes, compare with other voters, and, presumably, reach a consensus on the results.

Table 10 and Table 11 provided detailed symbolic descriptions of the zero-agent protocol.

Table 10. Zero-agent voting protocol preconditions

voter V_i:	has public-secret key pair (P_i, S_i) has a copy of the voters list knows the public keys and network addresses of all eligible voters (i.e., has access to the eligible voters list)

Table 11. Zero-agent voting protocol

Step	Actor	Description of action	Symbolic representation	Assertions
1	V_i	Generate an election-specific public-secret key pair for blinding messages, an election-specific random number for the first cycle, and n random numbers for the second cycle.	$P_i^*, S_i^*, R_i, R_{(1)i}..R_{(n)i}$	X *knows* P_i^* *iff* X == V_i X *knows* S_i^* *iff* X == V_i X *knows* R_i *iff* X == V_i X *knows* $R_{(y)i}$ *iff* X = V_i (for y = 1..n)
2	V_i	Acquire and complete a ballot	$B_i = vote_i(B)$	X *knows* B_i *iff* X = V_i
3	V_i	Create a message of the completed ballot and the random number for the first cycle.	$<B_i, R_i>$	
4	V_i	**Seal** the result with all voters' public keys.	$M_{(2)i} = P_1(P_2(\ldots P_n(<B_i, R_i >) \ldots))$	
5	V_i	**Seal** the result, prepended with the second cycle random number for voter n, with voter n's public key.	$P_n(R_{(n)i}, M_{(2)i})$	
6	V_i	Repeat the previous step with the random number and public key of voters n-1 through 1, in that order, then send the result to voter 1.	$M_i = P_1(R_{(1)i}, P_2(R_{(2)i}, P_3(\ldots (P_n(R_{(n)i}, M_{(2)i})) \ldots))) \rightarrow V_1$	
7	V_1	Confirm that all messages have been received and that voter 1's message is among them.		V_1 *knows identity*(M_i) $\forall$ i $\in$ {1..n} V_1 *knows* M_i *iff* i = 1
8	V_1	**Extract** the messages encrypted with voter 1's public key and remove the outermost random number from each, then send the resulting messages in scrambled order to voter 2.	$M_i' = P_2(R_{(2)i}, P_3(R_{(3)i}, P_4(\ldots(P_n(R_{(n)i}, M_{(2)i}))\ldots))) \rightarrow V_2$	
9	V_2.. V_{n-1}	Repeat the steps of the previous voter, except with the next public key and voter	$M_i'^k = P_{k+1}(R_{(k+1)i}, P_{k+2}(R_{(k+2)i}, P_{k+3}(\ldots(P_n(R_{(n)i}, M_{(2)i}))\ldots))) \rightarrow V_{k+1}$ (k = 2..n-1)	V_x *knows* $M_i'^k$ *iff* x = i
10	V_n	**Extract** the messages encrypted with voter n-1's public key and remove the outermost random number from each, then send the messages in scrambled order to voter 1.	$M_{(2)i} \rightarrow V_1$	V_n *knows* $M_{(2)i}$ *iff* i = n V_x *knows identity*($M_{(2)i}$) *iff* x = i, $\forall$ i $\in$ {1..n}

11	V_1	**Extract** the messages encrypted with voter 1's public key and confirm that all messages have been received and that voter 1's message is among them	$S_1(M_{(2)i}) = P_2(P_3(\ldots P_n(<B_i, R_i>) \ldots))$	V_1 *knows identity*$(M_{(2)i})$ $\forall i \in \{1..n\}$ V_1 *knows* M_i *iff* $i = 1$
12	V_1	**Sign** the results and send the resulting messages to voter 2.	$M_{(2)i}' = S_1(P_2(P_3(\ldots P_n(<B_i, R_i>) \ldots)))$	
13	V_2	**Verify** the signatures of voter 1 and confirm that all messages have been received and that voter 2's message is among them	$P_1(M_{(2)i}') = P_2(P_3(\ldots P_n(<B_i, R_i>) \ldots))$	
14	V_2	**Extract** the messages encrypted with voter 2's public key	$S_2(M_{(2)i}') = P_3(P_4(\ldots P_n(<B_i, R_i>) \ldots))$	
15	V_2	**Sign** the results and send the resulting messages to voter 2.	$M_{(2)i}' = S_2(P_3(P_4(\ldots P_n(<B_i, R_i>) \ldots)))$	
16	$V_{3}..$ V_{n-1}	Voter 3 through voter n-1 repeat the previous three steps in turn, incremented appropriately	$M_{(2)i}'^{k} = S_k(P_{k+1}(P_{k+2}(\ldots (P_n(<B_i, R_i>)) \ldots))) \rightarrow V_{k+1}$ $(k = 3..n-1)$	V_x *knows* $M_i'^{k}$ *iff* $x = i$
17	V_n	**Verify** the signatures of voter n-1 and confirm that all messages have been received and that voter n's message is among them	$P_n(<B_i, R_i>)$	
18	V_n	**Extract** the messages encrypted with voter n's public key.	$<B_i, R_i>$	V_n *knows* $P_T(S_A(<B_i, I_i^*>))$ *iff* $i = n$ V_x *knows identity*$(P_T(S_A(<B_i, I_i^*>)))$ *iff* $x = i$, $\forall i \in \{1..n\}$
19	V_n	**Sign** the results and send the resulting messages to all other voters.	$S_n(<B_i, R_i>) \rightarrow V_x$ $(x = 1..n-1)$	
20	V_x	Tally the results independently		

This protocol involves a lot of computation and network transmissions, especially as the electorate grows. It may not be practical for student elections where the electorate is large (over 17,000[35] in 1996-97) and transient, but it is probably suitable for small elections, such as departmental elections, at the University of Victoria. Another protocol [10], also suited to smaller elections for basically the same reasons, could be considered.

[35] From the 1997 University of Victoria Calendar.

8.2.6 The SAVE protocol

The improved two-agent voting system and the Sensus voting system are suited to large-scale elections, and there are other protocols that claim the same suitability and other useful properties [12]. Unfortunately, these protocols require some assumptions based on trust to be fully secure. The zero-agent protocol is not burdened with these trust requirements but it does not scale well because of its large computational demands which grow rapidly with the electorate. It would be worthwhile to examine the various solutions that have been developed so far and attempt to draw from them to create an even better protocol.

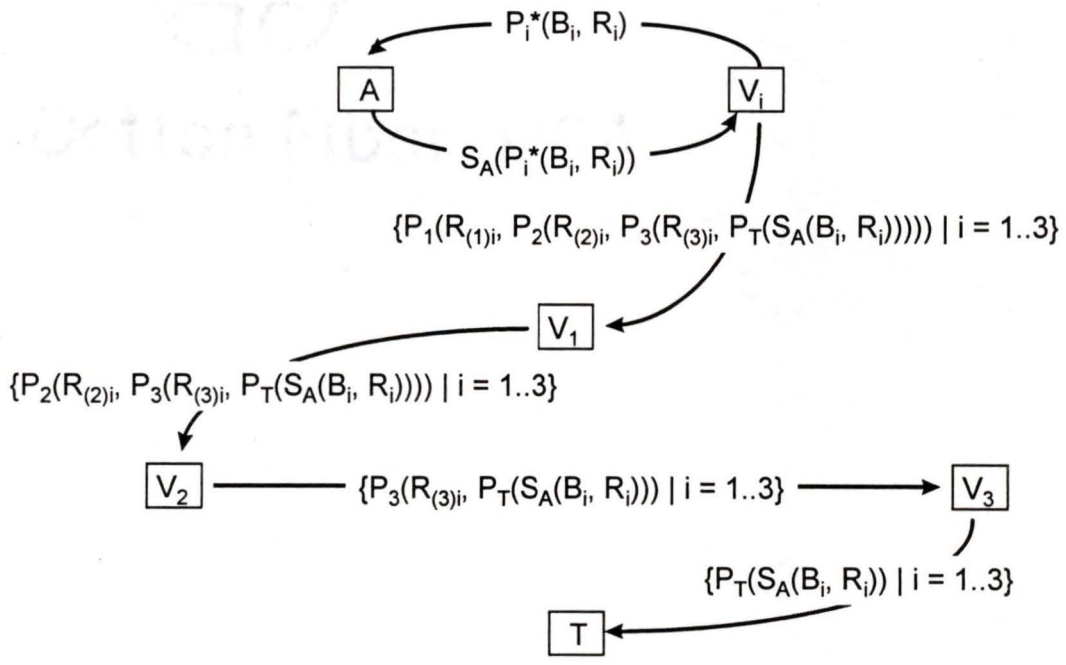


Figure 18. SAVE voting protocol

One obvious approach is to try to combine the peer protocol with a one-agent or a two-agent protocol to extract the best qualities of both. Here, we describe a protocol that does just that, preserving voter anonymity and allowing for vote verification in an non-

trusted environment, just like the zero-agent protocol, while enhancing its performance. It is referred to as the SAVE (Secure Anonymous Verifiable Electronic) voting protocol.

The SAVE protocol uses one public key cipher for signatures and for decryption. It has a tallier and a validator, in addition to the voters. A registrar is assumed for certain administrative tasks, such as maintaining the registered voters list. Like the zero-agent protocol, this protocol requires all voters who will participate to follow through on their commitment, and it needs a corresponding *participating* voters list to facilitate this. Unlike the zero-agent protocol, this list is determined in the SAVE protocol.

The SAVE protocol has three main stages: a validation stage, a ballot-scrambling stage, and a tallying stage. In the validation stage, voters have their completed ballots signed by the validator, without revealing the contents of the ballots to the validator. During the ballot-scrambling stage, all voters are involved in reordering the ballots so that each voter can ensure that no one else can trace the votes to their originators. In the tallying stage, the tallier collects, verifies, and counts the ballots.

Figure 18 shows the flow of the SAVE voting protocol for a very simple example election involving only three voters. As with the zero-agent protocol, this figure contains more details about the message flow because the protocol is difficult to explain verbally. A description of the protocol follows, while a more thorough explanation of the security issues is provided in section 9.1.4.

In the validation stage, each voter V_i generates a random identity number I_i^* and an election-specific public-secret key pair P_i^*, S_i^*. Each voter then creates a ballot message B_i by voting on a blank ballot form B, adds the random identity number to the ballot message, encrypts the result with the election-specific public key, then sends the result to the validator. The validator, who cannot read the encrypted messages, signs each message and returns it to its originator. Upon receipt of their signed ballot message from the validator, each voter verifies the validator's signature and decrypts the message with the election-specific secret key, leaving the equivalent of the original ballot message, together with the identity number, signed by the validator.

Next, each voter encrypts the signed ballot message with the tallier's public key, repeatedly generates and prepends a random number $R_{(x)i}$, then encrypts the result with voter V_x's public key, for x=n..1 (assuming an electorate of n voters). An example of the resulting message for the three-voter-electorate example in Figure 18 is shown near the top of that figure. As indicated in the figure, all voters send these initial messages to voter V_1(except, of course, voter V_1), from where the cycle will begin.

The next stage, which is described as the *ballot scrambling* cycle, proceeds exactly as the ballot scrambling cycle of the zero-agent protocol, except with the final voter (in this example, voter V3). This voter sends the resulting messages to the tallier instead of the first voter. There is no need for a validation cycle because the ballots are already validated and only the tallier is required to decrypt the messages.

In the final *tallying* stage, the tallier verifies the signature of the validator on each of the encrypted ballot messages, decrypts the messages, tallies the results, then posts the results and the ballot messages, together with their identity numbers. All voters can then verify that their ballot messages and identity numbers are among the posted results. Any discrepancy can be challenged using the signed ballot message from the validator.

Table 12 and Table 13 provide detailed symbolic descriptions of the SAVE protocol.

Table 12. SAVE voting protocol preconditions

voter V_i:	has public-secret key pair (P_i, S_i) knows the validator's public key and network address knows the tallier's public key and network address knows the public keys and network addresses of all eligible voters (i.e., has access to the eligible voters list)
validator:	has a public-secret key pair (P_v, S_v) knows the public keys and network addresses of all eligible voters (i.e., has access to the eligible voters list)
tallier:	has a public-secret key pair (P_t, S_t) knows the validator's public key

Table 13. SAVE voting system protocol

Step	Actor	Description of action	Symbolic representation	Assertions
1	V_i	Generate an election-specific public-secret key pair for blinding messages, and an election-specific ID number.	P_i^*, S_i^*, I_i^*	X *knows* P_i^* *iff* X == V_i X *knows* S_i^* *iff* X == V_i X *knows* I_i^* *iff* X == V_i
2	V_i	Acquire and complete a ballot	$B_i = vote_i(B)$	X *knows* B_i *iff* X == V_i
3	V_i	Create a message of the completed ballot and the election-specific ID number.	$<B_i, I_i^*>$	
4	V_i	**Blind** the message with the election-specific public key, then send the result to the validator.	$P_i^*(<B_i, I_i^*>) \rightarrow V$	
5	A	**Sign** the blinded message, return it to the voter, and update the voters list.	$S_A(P_i^*(<B_i, I_i^*>)) \rightarrow V_i$	A *knows* *identity*$(P_i^*(<B_i, I_i^*>))$ A ~~*knows*~~ P_i^* A ~~*knows*~~ B_i A ~~*knows*~~ I_i^*
6	A	At the end of the validation deadline, post the voters list for all voters to see.	$\{V_i : i = 1..\|E\|\}$	
7	V_i	**Un-blind** the message.	$S_A(<B_i, I_i^*>)$	
8	V_i	**Verify** the validator's signature	$B_i, I_i = P_A(S_A(<B_i, I_i^*>))$	
9	V_i	**Seal** the signed message with the tallier's public key	$P_T(S_A(<B_i, I_i^*>))$	
10	V_i	Generate random numbers for each of the voters in the voters list.	$R_{(1)i}..R_{(n)i} : n = \|E\|$	
11	V_i	Prepend the random number for voter n to the encrypted message	$R_{(n)i}, P_T(S_A(<B_i, I_i^*>))$	
12	V_i	**Seal** the result with voter n's public key.	$P_n(R_{(n)i}, P_T(S_A(<B_i, I_i^*>)))$	
13	V_i	Repeat the previous two steps with the random number and public key of voters n-1 through 1, in that order, then send the result to voter 1.	$M_i = P_1(R_{(1)i}, P_2(R_{(2)i}, P_3(\ldots (P_n(R_{(n)i}, P_T(S_A(<B_i, I_i^*>)))) \ldots))) \rightarrow V_1$	
14	V_1	Confirm that all messages have been received and that voter 1's message is among them.	There exists M_i $\forall$ i $\in$ {1..n}	V_1 *knows identity*(M_i) $\forall$ i $\in$ {1..n} V_1 *knows* M_i *iff* i == 1
15	V_1	**Extract** the messages encrypted with voter 1's public key and remove the outermost random number from each, then send the resulting messages in scrambled order to voter 2.	$M_i' = P_2(R_{(2)i}, P_3(R_{(3)i}, P_4(\ldots (P_n(R_{(n)i}, P_T(S_A(<B_i, I_i^*>)))) \ldots)) \rightarrow V_2$	

16	V_2.. V_{n-1}	Repeat the steps of the previous voter, except with the next public key and voter	$M_i'^* = P_{k+1}(R_{(k+1)i}, P_{k+2}(R_{(k+2)i}, P_{k+3}(\ldots (P_n(R_{(n)i}, P_T(S_A(<B_i, I_i^*>)))) \ldots))) \rightarrow V_{k+1}$ $(k = 2..n-1)$	V_x *knows* $M_i'^*$ *iff* $x == i$
17	V_n	**Extract** the messages encrypted with voter n-1's public key and remove the outermost random number from each, then send the messages in scrambled order to the tallier	$P_T(S_A(<B_i, I_i^*>)) \rightarrow T$	V_n *knows* $P_T(S_A(<B_i,I_i^*>))$ *iff* $i = n$ V_x *knows identity*($P_T(S_A(<B_i,I_i^*>))$) *iff* $x = i$, $\forall i \in \{1..n\}$
18	T	**Extract** the signed ballot messages.	$S_A(<B_i, I_i^*>) = S_T(P_T(S_A(<B_i, I_i^*>)))$	(T *knows* B_i T *knows* I_i^* T ~~*knows*~~ *identity*(B_i) T ~~*knows*~~ *identity*(I_i^*)) $\forall i \in \{1..n\}$
19	T	**Verify** the signatures	$B_i, I_i^* = P_A(S_A(<B_i, I_i^*>))$	
20	T	Post the results	$\{<B_i, I_i^*> : i \, e \in \{0..n\}\}$	x *knows identity*(B_i) *iff* x == V_i x *knows identity*(I_i^*) *iff* x == V_i

The set-up and validation stages can be done in parallel on an election-wide basis but not on a personal basis, meaning that voters must complete the set-up stage before beginning the validation stage, but any voter can begin the validation stage while another voter is still in the set-up stage. The ballot-scrambling stage, however, involves all voters as a group and thus cannot begin until all voters have completed the validation stage. Also, the tallying stage cannot begin until the ballot-scrambling stage has completed. Therefore, the voting process has to be monitored closely, with some time constraints imposed, in order to ensure that the protocol completes in a timely fashion. For example, voters should be required to complete validation by a specific time, and, during the ballot-scrambling stage, no voter should be able to hold the ballots for longer than a certain amount of time before relaying them to the next voter. Also, the tallier should be required to post the results in a minimum amount of time after the last voter in the ballot-scrambling stage is required to forward the ballots.

The validation stage is an opportunity to compile a list of participating voters before the error-sensitive cycle of the ballot-scrambling stage begins. Any voter completing

validation is expected to participate in the vote, so the chance of an absentee or a reluctant voter forcing the rerunning of the entire ballot-scrambling stage is reduced. This is the point at which the specific rules for the election can also be established (e.g., the maximum delay per voter, as previously described, the order of the voters in the ballot-scrambling stage, and the location of the tallier).

The SAVE protocol offers other desirable features. For instance, the design of the protocol ensures that an up-to-date election-specific list of participating voters is created before the election takes place. This happens because voters announce their intent to vote by validating their ballots. If the entire voting system is automated, except for the actual selections made by the voters, then there need not be a large time delay between validating and vote submission, so the likelihood of someone being on the participating voters list and not actually participating is lessened. If the elections are well-designed, then voters can still have the option of casting a null vote if they do not find any of the choices desirable. This would allow them to participate, and thereby enable the protocol, without actually voting.

Another advantage of the SAVE protocol is that a subdivided electorate can easily be supported for large-scale elections that might otherwise render the protocol too costly in terms of execution time. The electorate can be subdivided into several subgroups, or *precincts,* of more reasonable size, yet only one validator and one tallier are needed for all subgroups combined.

9. Analysis of the protocols

We examine the four main candidate protocols (Sensus, improved two-agent, zero-agent, and SAVE) for individual assessment and for comparison. In particular, we look at *security*, *performance*, and *qualitative* (non-performance aesthetic) characteristics. Note that the improved two-agent protocol and the SAVE protocol are presented as preferred alternatives to the Sensus and zero-agent protocols, and this is reflected in the analysis.

9.1 Security analysis

Each of the four protocols improves on the security of the conventional mail-in voting system by using cryptography for secure delivery of messages and by providing means for voters to verify that their votes were included in the final tally and counted correctly. Accuracy is also improved because modern computer hardware and error detection/correction techniques virtually eliminate the possibility of errors occurring. This is important not only for accurate counting but also for reducing opportunity for ballot stuffing.

Still, there are many ways to attack the protocols. Defences against such attacks can sometimes be merely administrative. For example, all computer-based voting protocols are susceptible to corrupt software, specifically the applications that run on the voter's behalf. The best method to prevent this kind of attack is to provide voters with the source code, which they can scrutinise and then compile locally or, alternatively, have a trusted third party do so. Attacks on the cryptographic algorithms can generally be deterred by using proven algorithms with sufficiently large key sizes and by good management of secrets (i.e., concealing passwords and other sensitive information).

There are two final problems to discuss, which apply to the verification property of all four electronic voting protocols. First, although voters are able to verify that their vote was included in the tally and was counted correctly, the claim that a voter can anonymously challenge a miscounted vote assumes goodwill by the tallier. In the Sensus protocol, for example, the challenge procedure is specified, and ostensibly requires the voter to resubmit the vote, using the original receipt from the tallier, along an anonymous path. For details on why this is a problem, see Appendix C - Challenging a miscounted vote - on page 120. The other point is that if voters have a means to verify that their votes were counted correctly (i.e., a receipt), the voting system is open to bribery and coercion since voters can also prove how they voted to a third party. Of course, since voters cast their ballots remotely in each of these protocols, one could argue that they are already exposed to such threats, but the receipts do provide a convenience for villains since they do not have to be present at every voter's location during the polling period. Resolution of these two problems without compromising the general security of a voting protocol might be an interesting research topic.

9.1.1 Sensus protocol

The Sensus system has few weaknesses. In comparison, the mail-in system's main security weaknesses exist in the delivery of ballots to the voters, the delivery of the completed ballots to the validator, the handling of the returned ballots (e.g., snooping by the validator, collusion with the tallier), and in the tallying (e.g., miscounting, ballot box stuffing). The Sensus system eliminates the possibility of undetected tampering with the delivery of the ballots to and from the voters by using encryption and digital signatures. The use of receipts eliminates the possibility of tampering with a voter's submission without the voter being able to detect it and provides the voter with a means to challenge an incorrectly counted vote. Most importantly, the voter's identity and vote cannot be connected because the blind signature allows validation without revealing the vote and

the anonymous channel allows delivery of the validated vote without revealing the sender's identity.

The blind signature scheme works if the public key algorithm is commutative (RSA, for example, has this property). It allows voters to conceal their ballots from the validator, get the validator's signature on the concealed ballots, and then reveal the signed ballot privately. There need not be any concern about what the validator is signing if the signing key the validator uses is only valid for ballot messages. If, for example, the tallier discovers something other than a valid ballot when it verifies (and removes) the validator's signature from the ballot message sent by the voter, then it knows foul play has occurred. Also, the voter cannot manipulate the ballot message signed by the validator because the probability of it being meaningful when the signature is later removed is very small.

There are some problems with the Sensus protocol. One problem is that there is no inherent feature of the Sensus protocol which prevents ballot box stuffing. Since the validator knows who voted and therefore who did not, it could pose as one or more valid voters who did not vote and submit votes to the tallier. This is possible because the tallier has no way of knowing who is sending it votes and can only determine the validity of the votes by the signature that they must bear; the validator, of course, provides this signature. The suggested remedy to this threat is to require **all** voters to submit a vote and, if appropriate, to allow *null* votes for those who do not wish to make a decision on the election issue(s).

The validator and the tallier can collude to reveal how a voter voted in the Sensus system if voters tend to submit their encrypted ballots to the tallier immediately after receiving a blind signature from the validator. Once the validator has validated a blinded ballot, it can inform the tallier of the name of the voter who is likely to submit a ballot for tallying next. This threat can be easily reduced by having voters wait a random amount of time between communications with the two agents. In fact, it can be eliminated if the two steps (validation and submission of the ballots) must be done during two separate, non-overlapping time periods. Although this requirement could

impose an inconvenient delay on the protocol, its effect could be minimised by fully automating the voting application and scheduling all instances of it (among the voters) to execute at and during a specific time (after voters have cast their votes, of course). Note, however, that the imposition of a deadline to separate the validation and tallying stages in time would make the Sensus protocol more like the original Fujioka, Okamoto, and Ohta protocol, which suggests that the value of the Sensus enhancement (instant tally receipts) is somewhat contentious.

The main weakness with the Sensus system and those systems which it is derived from is the need for anonymous channels between voters and the tallier. Simple unidirectional anonymous channels are difficult to establish because at least one station along the channel usually can identify the sender. Worse, in the Sensus system the anonymous path must be *returnable*. That is, a reply to a message sent along the channel can be returned to the originator of the message without knowledge of the location of the originator. This necessitates that information about the path must be retained, presumably by the forwarding servers that the authors suggest, so the servers must essentially be trusted by the voters. If a direct reply was not necessary, an anonymous path might be simpler to establish. The creators of the Sensus system and the Fujioka, Okamoto, and Ohta system do not describe how to implement an anonymous channel except to say that a chain of anonymous relay stations could be employed.[36]

9.1.2 Improved two-agent protocol

The improved two-agent protocol does not require an anonymous channel because the validator is responsible for delivering the votes to the tallier, and does so in one batch after all votes have been collected. This is the same way the mail-in system works except that the messages are electronic instead of paper, cryptography is used for sealing and signing instead of envelopes and handwriting, and a computer network is used for

[36] There **are** protocols designed to achieve anonymous communications. One excellent example demonstrates how electronic mail can be sent *and returned* without identifying the sender or the sender's address [9].

delivery instead of the mail system. No one can read a voter's ballot except the tallier because the voter encrypts the ballot with the tallier's public key, so the validator can only verify that a submitted vote came from a particular voter but cannot read its contents. Voters can verify that their votes were counted correctly because the final tally must be posted and each vote is displayed together with a secret random ID number that the voter generates.

The tallier provides the voter with a receipt of the vote and ID number encrypted with an election-specific public key generated by the voter and provided with the original vote. This encryption is necessary because the tallier must return the receipt to the voter through the validator without revealing the contents of the receipt (and therefore the vote). The validator can easily track which voter should get the receipt from the tallier by maintaining network connections with the voter and with the tallier during the vote in a single thread of execution.

The problem with the two-agent protocol is that the validator and the tallier have opportunity to collude to link voters with votes. Together, they hold all the necessary information to link voters to their votes, as they do in the Sensus system. Unlike in the Sensus system, votes travel directly from the validator to the tallier. Without proper scrutiny, the validator could attach identifying information to the encrypted ballots. Worse, the tallier could provide its secret key to the tallier.

This same collusion threat exists in the conventional mail-in system. To protect against it, the processing of the votes is scrutinised by either disinterested parties or by representatives of the opposing choices in the election decision. This is an expensive and cumbersome procedure that still requires the trust or acceptance of the electorate that those selected to scrutinise the voting process are trustworthy. It does prevent connection of voters to their votes, however, if the votes are completely detached from the identities of their originators upon receipt.

With the improved two-agent system, the validator application must be trusted to detach votes from voters. The obvious way to achieve this is to maintain the validator on a secure computer system that can be monitored and scrutinised by the voters or

trusted third parties, or both. Monitoring is necessary to ensure that no identifying information for the received votes and receipts is retained or passed on to the tallier. This permanently detaches voters' identities from their votes and from their receipts as well. Note that there is no need to trust the tallier as well: without co-operation from the validator, it cannot perform any treachery without being detected.

The Sensus protocol does not require the validator to be trusted. However, the two-way anonymous channel requires each intermediate message relay stations along the delivery path to retain information about the address of each station on either side of it, even if one of the stations is the voter's computer or the tallier. Therefore, the relay stations can potentially collude to trace the origin of the vote and provide this information to the tallier. At least one must be trusted to not collude with the others (see Figure 19), and even if one relay station does not participate in a collusion effort, the

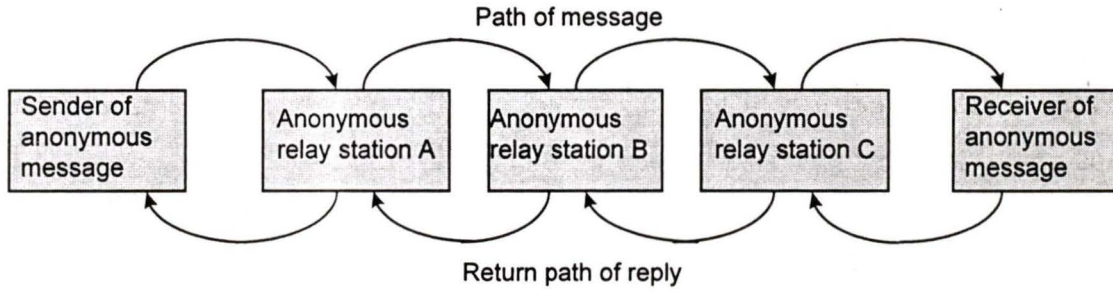


Figure 19. Two-way anonymous communications channel using relay stations

other stations could still bypass it in secret. There may be protocols that enable untraceable two-way anonymous messaging [9] but they would have to be reasonable in terms of processing and execution time. Protocols that have these properties of untraceability (e.g., ANDOS [38]) are notorious for being computationally intensive.

Note that, although the requirement to trust the validator may be a weak point in the improved two-agent protocol, it has some benefit. For example, the validator can help to prevent the tallier from stuffing votes, because it knows exactly how many voters voted and can immediately detect an excessive tally. This same quality can only be achieved in the Sensus protocol if the validator is trusted. Furthermore, the trust requirement is no worse than the mail-in protocol's requirement to trust the election administration. In fact, it is no worse than the Sensus protocol's requirement for an anonymous channel, since both assume the trust of at least one station along the delivery path of a completed ballot.

Also, the issue of having to trust the validator can be viewed in another way: it focuses the protocol's weakness, allowing for concentrated scrutiny. Just as a fuse creates a convenient weak point in an electrical circuit which will always fail first, the validator is the most likely point of attack in the two-agent system and therefore can be monitored to guard against malicious behaviour.

Finally, it may seem incongruous to require that the validator be trusted, yet to also depend on encryption to prevent the validator from reading the contents of one's vote. The explanation for this procedure is simple: Since trust in the validator may rely on monitoring by third parties or by other voters, then all messages may be viewable by anyone authorised to participate in the monitoring. Therefore, without encryption, vote messages could be read and the secrecy of the ballot compromised.

9.1.3 Zero-agent protocol

In the improved two-agent protocol, the participation of all voters (or designated trusted third parties) is required to ensure the honesty of the validator and thereby preserve voters' anonymity. The zero-agent protocol takes this concept to the electronic level by requiring all voters to directly participate in the detachment of votes from voters. For instance, in the first cycle each voter removes one of the *layers* of encryption from all of the multiply-encrypted ballot messages and shuffles the order of those messages.

No-one can trace votes through the first cycle, even if they can monitor all network traffic, because each voter gets an opportunity to uniquely modify the vote messages and then reorder them in private before sending them to the next voter. This unique modification occurs because each voter decrypts all of the messages with her secret key and removes the outermost random numbers from them. This prevents anyone else from reversing the shuffling of the votes because they would have to know the random numbers. A related attack would be to attempt to reconstruct the multiply-encrypted vote messages from the votes at the end of the protocol. This will not work either, because of the random numbers; although all voters know all of the random numbers in their own vote messages and learn one of the random numbers in all of the other vote messages, an attacker would need to know **all** of the random numbers in another vote message in order to reconstruct it.

The first cycle therefore guarantees that a voter's ballot is irreversibly reordered and untraceable. It does not prevent one of the voters from tampering with the vote messages. Any voter can drop one of the messages, add a message, or replace an existing one with a copy of an existing valid vote message. However, if a vote message was dropped or added, the next voter would immediately detect that the number of messages did not match the number of voters. If a message was replaced, then the affected voter would subsequently notice that her vote was missing, either in the remainder of the first cycle or in the second cycle, since she knows how her message appears after every encryption done to it.

Unfortunately, no-one can determine who modified a vote in the first cycle, in most cases. The best that a voter can do during the first cycle, in response to not finding her vote message intact among the messages, is to stop the election. This also applies if the modified message is discovered during the second cycle but was actually modified during the first cycle (e.g., a voter's message is replaced by another voter **after** the first voter processed the vote messages). In either case, a voter whose vote message is tampered with in the first round will detect the tampering before the election completes and therefore can stop it before the votes can be revealed. This cannot be guaranteed if

the tampering occurs in the second cycle, but, because at this stage the vote messages can be reconstructed and because everyone must sign them at each step, the perpetrator can be detected.

Another problem with the zero-agent protocol is that the participating electorate must be predetermined and **everyone** must participate, even if they do not wish to make a decision, or else the protocol will fail. Although this may sound restrictive, if null votes are supported then voters can still participate without making a decision. Furthermore, by requiring all voters to participate, the threat of ballot stuffing is eliminated since everyone knows how many votes should be in the final count (including null votes, if any).

Yet another problem is that the last voter discovers the results before everyone else. A remedy to this problem would be to enforce a special policy requiring that voter to distribute the results within a short time of receiving them. In fact, all participants could be required to respond within a predetermined time interval to prevent anyone from gaining an advantage by learning the results before the others, and to keep the election executing at a near-optimal pace. The problem of accidental disruption is small because of the high degree of reliability of modern computer systems; it can be further reduced by polling the participating computers prior to the election, to determine the state of their network connections and their voting applications, and making adjustments if necessary. The effects of intentional disruption can be reduced by dividing a large electorate into several smaller ones and by planning for the possibility of several reruns of the election.

9.1.4 SAVE protocol

The SAVE protocol uses blind signatures, a validator, and a tallier, like the Sensus protocol. However, instead of an anonymous channel it uses a ballot shuffling cycle like the zero-agent protocol. It has no need for a second cycle because validation of the vote is done in advance with the validator and this provides a receipt to prove that the voter actually voted (and in which way).

As with the zero-agent protocol, since each voter encrypts her ballot with the public key of **all** participating voters, then all voters must participate in decrypting the ballots before the tallier can read them. Each voter therefore gets the opportunity to render the ballot messages untraceable by shuffling their order. If a voter does not reorder the ballot messages, then someone with the ability to monitor the network traffic might track her vote. Again, as in the zero-agent protocol, another attack is to attempt to reconstruct the original multiply-encrypted vote messages at the end of the protocol by re-encrypting the votes with all voters' public keys. In the same way, this is impossible because no one can know or find out **all** of the random numbers for any one message except the voter who created that message.

Unlike the zero-agent protocol, the SAVE protocol does not allow the last voter in the cycle(s) to learn the election results before the others, because, even at the end of the ballot shuffling cycle, the messages must still be decrypted by the tallier. Like the zero-agent protocol, however, the SAVE protocol does not tolerate disruption and does not always allow the responsible party to be determined. If a voter discovers that her vote was tampered with (either replaced or modified), she can challenge the vote immediately. Unlike the Sensus system, this challenge cannot be done anonymously, because, without running the entire election over again, because there is no anonymous channel for the voter to communicate with the tallier on; however, this is not really a shortcoming (see Appendix C - Challenging a miscounted vote - on page 120).

9.2 Performance analysis

In assessing the performance of the four protocols, we seek to quantify two time characteristics for comparison and absolute measurement: (i) the total processing time per user and (ii) the total execution time for the protocol to complete. The total processing time is important for determining how a voting system will tax voters' computer systems. The total execution time is even more important because it is a measure of how long an election will take and therefore how practical it will be. These characteristics are difficult to measure without thorough complexity analysis; a full

working implementation of each protocol is required to get accurate performance statistics.

We feel that precise performance statistics are not only impractical but are also unnecessary at this point. Instead we adopt a simplified approach that still allows for a good assessment and comparison of the voting protocols. For instance, we consider that cryptographic functions, especially basic public key functions such as encrypting, decrypting, signing, and verifying, account for the bulk of the processing time in the various applications of an electronic voting system. Tests have shown that this is a sound claim when the applications perform infrequent I/O, have simple GUI or textual interfaces, and reasonably lean code. For example, an average public key encryption may consume in the order of one second of CPU time on a typical computer, while other processing related to that encryption might consume one tenth as much CPU time.

Since the cryptographic functions dominate voting protocol performance, the protocols can be compared by determining how many cryptographic functions are used and how they are used (e.g., in sequence or in parallel). Real time estimates are important for judging the acceptability of the protocols. For this, we use benchmark statistics from RSA Data Security Inc. for their BSAFE™ version 3.0 cryptographic tool-set. The particular statistics used are the encryption, decryption, signing, and verification times for the RSA algorithm using 1024-bit keys on an Intel Pentium 90 MHz PC [*RSA '96*]. These are 0.0086 seconds for encryption and verification and 0.14 seconds for decryption and signing. Other cryptographic functions, such as those based on message digests, random-number generators, and symmetric-key algorithms, are much faster (if used at all in any of these protocols), so they are not considered to be significant contributors to the overall performance statistics. The generation of cryptographic keys, which is performed in some of these algorithms, can be time-consuming (several seconds or more) but is infrequently done and normally can be done in advance of the election, so it is also not considered to be a significant factor in protocol performance.

Note that benchmark performance statistics for the BSAFE™ product are not assumed to be authoritative; another cryptographic tool-set, whether commercial or custom written, may provide performance that is substantially different. Also, our choice of the 1024-bit key size is arbitrary (although it is in fact an often-recommended minimum key size in the industry for security). With voting systems especially, one should prefer large keys because the encrypted information (i.e., the completed ballots) is sometimes valuable for a long period of time and therefore may be subject to a long-term attack. For example, a person's political decisions may be of value to a research group many years into the future.[37]

We must also consider the improvement of computer processing with time when analysing the future potential for an electronic voting system. For example, Moore's Law states that, on average, the ratio of processing power to equipment cost doubles every 18 months. In three years, we can expect voting systems to perform 4 times as well as they might today, on average. We may have already seen this effect, since the benchmark statistics were based on an Intel Pentium 90 MHz processor, yet, the current entry-level personal computer processor runs at 200 MHz or faster and has significant improvements in the system bus, on-board memory, and other areas. An even greater improvement in overall performance could be gained by using hardware implementations of the cryptographic algorithms since they can reduce encryption and decryption times by a factor of 100 or more [*Coulouris '94*]. Although a lack of standards, patent restrictions, and government regulations make them currently uneconomical to produce for the mass market, each of these limitations is disappearing. We may soon see popular central processors that contain built-in cryptographic functions, implemented in hardware or micro-code, as part of their instruction sets.

9.2.1 Sensus protocol

The Sensus protocol has two main steps after registration (we disregard the time requirements of the registration step as negligible because most of it can be done once

[37] ...for information on that person's political, social, and economic tendencies, for example.

for each voter **before** all elections). Table 14 shows the numbers of cryptographic function calls made by the different actors in the Sensus protocol during the voting period (i.e., not including the registrar). Using our selected benchmark statistics, the total cryptographic processing time is 1.18 seconds for each voter, including the processing done by the validator and the tallier to service the voter; 0.61 seconds of that time is local processing on each voter's computer. Since all of this processing occurs entirely in sequence for each voter, then the *minimum* total execution time for voting is 1.18 seconds plus network communications time and processing overhead.

Table 14. Cryptographic function calls in the Sensus protocol

Actor	#Encryptions	#Decryptions	#Signatures	#Verifications
Voter (each)	5	2	1	2
Validator	1	1	1	1
Tallier		1	1	1

Note that the minimum total execution time per voter assumes that the entire voting process is automated and that the voter does not delay between completing validation with the validator and beginning ballot submission with the tallier. It also assumes that the processing power of the validator and the tallier will be effectively dedicated to the voter while the voter is submitting a vote. This is possible in the Sensus protocol because the processing of simultaneous requests by multiple voters can be performed in parallel; for example, the validator and the tallier can be multithreaded and run on multiprocessor systems, and they can also simultaneously run on any number of individual computer systems. Additionally, the polling period can be made arbitrarily long so that the likelihood of multiple voters voting at the same time is low.

Realistically, however, occasional delays due to large numbers of voters validating or submitting their votes at the same time can be expected. The worst case scenario has all voters attempting to perform either action simultaneously. For instance, if all voters attempted to validate simultaneously, the turnaround time from each voter's perspective,

ignoring overhead, would range from 0.2886 seconds (from Table 14 and benchmark times) for the first voter serviced to 0.2886n/k seconds for the last voter serviced (assuming k individual processors and an electorate of n voters).

Despite the potential performance problems with a large electorate, the Sensus protocol is well-suited to large-scale elections. Its processing demands and general performance are, at worst, directly related to the size of the electorate, and can be mitigated by using more powerful processors, running multiple clones of the agents, increasing the network capacity, etc. Even if there is a limit to the number of voters that a powerful computing infrastructure can reasonably support before the cost of enhancement becomes unacceptable, this number is probably sufficiently large to handle most typical voting populations. Certainly an electorate in the thousands should result in a very acceptable minimum turnaround time for the voters (i.e., from the start of the validation step to the completion of the ballot submission step).

9.2.2 Improved two-agent protocol

The improved two-agent protocol has one stage from the voter's standpoint: submission of the vote. As with the Sensus protocol, all processing in this protocol can occur in parallel among the voters but executes entirely in sequence for each voter; the one exception is the verification of the validator's receipt by each voter, since this occurs while the tallier examines the voter's ballot. Table 15 shows the numbers of cryptographic function calls made by the different actors in the improved two-agent

Table 15. Cryptographic function calls in the improved two-agent protocol

Actor	#Encryptions	#Decryptions	#Signatures	#Verifications
Voter (each)	2	2	0	2
Validator	1	1	0	2
Tallier	1	1	1	1

protocol during the voting period (i.e., not including the registrar). Using our selected benchmark statistics, the total cryptographic processing time is 0.76 seconds for each voter, including the processing done by the validator and the tallier to service the voter; 0.30 seconds of that time is local processing on each voter's computer. Since all of this processing occurs entirely in sequence for each voter, then the *minimum* total execution time for voting is 0.76 seconds plus network communications time and processing overhead.

The network and processing overhead for the improved two-agent protocol, with 4 network transmissions of messages, is likely to be smaller than that of the Sensus protocol, with 6 transmissions. Otherwise, the two protocols are very similar in performance and how they are affected by electorate size, with the improved two-agent protocol enjoying an edge.

9.2.3 Zero-agent protocol

The performance of the zero-agent protocol is more directly affected by the size of the electorate, since processing for all voters occurs in sequence (i.e., each voter does some processing in turn), unlike the Sensus and improved two-agent protocols, in which voters submit their votes at any time during the polling period. Furthermore, the amount of work *each voter* performs depends on the number of participating voters. Therefore, we need to use example electorate sizes to illustrate the absolute performance of the protocol. In general, an electorate of 100 voters will be used since most departmental and faculty elections at the University of Victoria have electorates in this size range. Other numbers, such as 1000 (university-wide staff elections) and 10 (small departmental), may be used for basic trend analysis of the protocol's performance.

The zero-agent protocol has four main stages in which cryptographic functions are used. If we consider the electorate to consist of n voters, then each voter must perform (i) 2n encryptions in the first stage, (ii) n decryptions in the second stage, (iii) n signatures, n decryptions, and n verifications in the third stage, and (iv) n verifications in the fourth stage (except for the first voter). Table 16 summarises these cryptographic

function calls over the entire protocol. For an electorate of 100 voters, the total processing time for cryptographic functions is approximately 45 seconds per voter. For the total execution time of the protocol, we note that while the first and fourth stages can be done in parallel by all voters, the second and third stages must be done sequentially. Therefore, the minimum time is bounded by 2n encryptions in the first stage, followed by n^2 decryptions in the second stage, followed by n^2 signature-decryption-verification

Table 16. Cryptographic function calls in the zero-agent protocol

Actor	#Encryptions	#Decryptions	#Signatures	#Verifications
Voter (each)	2n	2n	n	2n
Validator	n/a	n/a	n/a	n/a
Tallier	n/a	n/a	n/a	n/a

triplets in the third stage, followed by n verifications in the fourth stage. For an electorate of 100 voters, this would mean a minimum execution time of approximately 4300 seconds (1 hour and 11 minutes) for the cryptography alone.

The execution time does not include other actions such as file I/O, network communications, general processing, and protocol delays. For the zero-agent protocol, there are $1 + n + n + 1 = 2n + 2$ sequential network transfers. With a 100-voter electorate, and assuming 64-bit random numbers, ballot packages would be, at maximum, slightly less than 1 Kilobyte in size and therefore could be transmitted in a negligible amount of time, including overhead, on average. However, in the middle two cycles, each voter sends all 100 vote messages to the next voter, so up to 100 Kb of data would be involved in each transmission, which might take perhaps a second[38] on a moderately loaded local area network. Therefore the contribution to the total execution time by network transmissions would be slightly more than 200 seconds. File I/O can be minimised by limiting file accesses and using buffers, so its contribution to the total

[38] Based on observations of repeated trials involving sending a ~100 Kb file from a workstation to a geographically close server on a lightly-loaded 10 Megabit-per-second Ethernet network.

time, along with general processing time, should be no more than a few seconds. Protocol delays, such as pauses by each voter before they perform some action, can be nearly eliminated by automating the entire process. Overall, then, cryptographic processing accounts for the vast majority of execution time (in this case well over 95%), which is why we consider it to be sufficient for assessing protocol performance.

9.2.4 SAVE protocol

Like the zero-agent protocol, the performance of the SAVE protocol is directly dependent on the size of the electorate because of its sequential vote scrambling cycle. Unlike the zero-agent protocol, the SAVE protocol uses a validator and a tallier (which, as with the Sensus protocol, can be the same agent). The protocol has three main stages in which cryptographic functions are used. For an electorate of n voters, each voter must perform (i) 2+n encryptions, 1 decryption, and 1 verification in the first stage and (ii) n decryptions in the second stage. The validator must perform 1 signature and the tallier must perform 1 decryption and 1 verification on behalf of each voter. Table 17 summarises these cryptographic function calls over the entire protocol.

Table 17. Cryptographic function calls in the SAVE protocol

Actor	#Encryptions	#Decryptions	#Signatures	#Verifications
Voter (each)	2+n	1+n	0	1
Validator	1	0	0	0
Tallier	0	1	0	1

With an electorate of 100 voters, the total cryptographic processing time would be about 15 seconds per voter, with very little extra processing time contributed by either agent on behalf of the voter. The minimum total execution time is bounded by 2+n encryptions, 1 decryption, 1 signature, and 1 verification for each of the voters in the first stage, followed by n^2 decryptions by all of the voters in the second stage, followed by n decryptions and verifications by the tallier in the final stage. For an electorate of 100

voters, this would result in a minimum execution time of approximately 1430 seconds (less than 24 minutes). As with the zero-agent protocol, this execution time does not include other actions such as file I/O, network communications, general processing, and protocol delays because they can be limited to contributing a small fraction of the overall time.

The SAVE protocol is potentially much faster than the zero-agent protocol. Using the benchmark statistics, the per-voter processing time is ((4 + n) * 0.14 + (3 + n) * 0.0086) seconds, or (0.1486n + 0.5464) seconds, for the SAVE protocol and (3n * 0.14 + 4n * 0.0086) seconds, or 0.4544n seconds, for the zero-agent protocol. The total execution time is:

$$((n^2 + n + 2) * 0.14 + (2n + 3) * 0.0086) \text{ seconds}$$
$$= (0.14n^2 + 0.4458n + 0.3058) \text{ seconds}$$
for the SAVE protocol, and
$$(3n^2 * 0.14 + (n^2 + 3n) * 0.0086) \text{ seconds}$$
$$= (0.4286n^2 + 0.0258n) \text{ seconds}$$
for the zero-agent protocol.

For both times, the SAVE protocol tends to be three times as fast as the zero-agent protocol for larger electorates (for electorates of 10 voters, for example, the multiple is about 2.5 instead of 3). Of course, neither the SAVE protocol nor the zero-agent protocol approaches the performance of the improved two-agent protocol and the Sensus protocols, in which the processing effort by each voter is constant and the turnaround time is, at worst, directly related to the size of the electorate. When the electorate size is large enough to cause significant degradation to the latter protocols in practice (e.g., network overloading, etc.), it would render the former protocols completely impractical because of their exponentially-growing execution times. For instance, with an electorate of 1000 voters, the SAVE protocol would require about 40 hours to complete and the zero-agent protocol would require 120 hours.

9.3 Qualitative analysis

The qualitative analysis covers those properties of the four electronic voting protocols that are difficult to quantify, such as ease-of-use, voter convenience, and cost. In general, the protocols are similar in these properties because they all use the same type of equipment (computers and networks).

Certainly all of the protocols are at least as convenient as the conventional mail-in system since they permit voters to participate without having to travel to a designated polling station. In fact, the electronic protocols are actually even more convenient because voters do not have to bring their completed ballots to special mail pick-up locations but can initiate and complete delivery from their network-attached computers. Furthermore, since the protocols all use (or at least can use) the Internet with no special restrictions and because voters retrieve their ballots instead of waiting for them to be delivered to a predetermined address, voters can participate in an election from anywhere that the Internet can be accessed.

An apparent advantage with the Sensus protocol and the improved two-agent protocol over the others is that they can also be configured to allow voters to resubmit (and therefore change) their votes anytime until the final tally of the election results. The zero-agent protocol and the SAVE protocol cannot support vote changing because (i) it is impractical, requiring the entire election to be rerun and (ii) it would expose the voter making the change since the previous results could easily be compared to the subsequent results. However, vote changing is unnecessary with *any* protocol **if** voters do not submit (and therefore commit to) their votes until just before the polls close. This is exactly what will happen with the zero-agent protocol and the SAVE protocol if they are automated, with the voter's decision being the only necessary manual action. In this situation, voters can change their votes at any time until the scheduled running of the election processing cycles peculiar to those protocols. Therefore, we see a general advantage for all four electronic voting protocols over the conventional protocols in that

voters can easily delay submission of their votes until the last moment, giving them the maximum possible time to settle on their decision(s).

There are some drawbacks to electronic voting protocols, in comparison to conventional voting systems. One problem is the cost of providing every voter with access to a personal computer and a network connection. A related problem is that if remote dial-up connections are necessary, their inferior performance compared to LAN connections (e.g., a fast Ethernet link can provide throughput that is orders of magnitude better than the fasted modems) will significantly affect performance; furthermore, they tend to be more expensive to use and more fault-prone.

10. A demonstration application

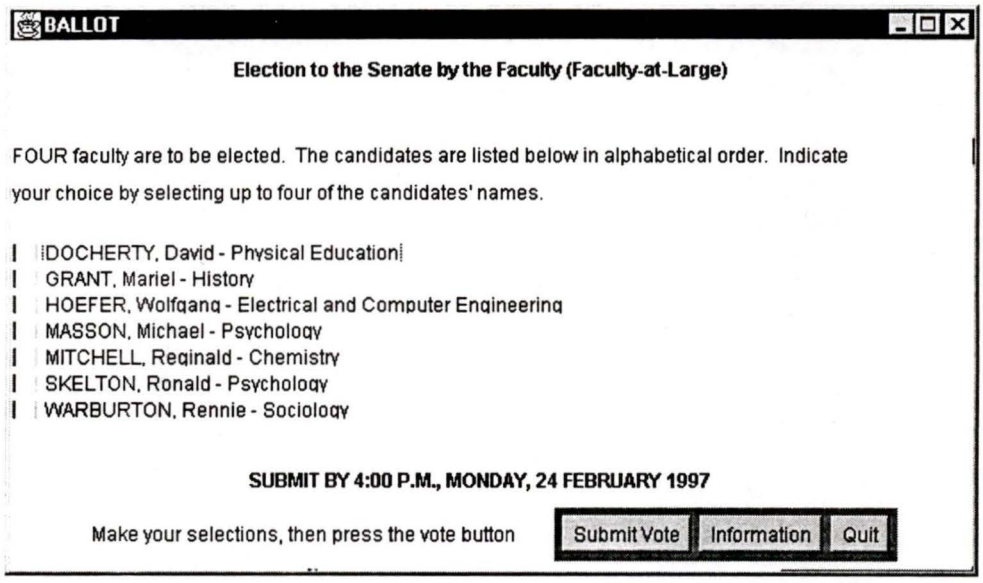


Figure 20. Screen shot of voting system implementation: multiple choice election

In order to demonstrate how an electronic voting system might work, we provide a working prototype of the improved two-agent protocol. This was chosen instead of the SAVE protocol because it only requires one voter to demonstrate; to adequately demonstrate the SAVE protocol requires at least 3 voters and, preferably, 10 or more. Figure 20 and Figure 21 show the appearance of the voting application for two sample elections based on actual elections held at the University.

Although security features were the primary reasons for pursuing and proposing an electronic voting protocol to replace the conventional mail-in system, one of the most critical design goals for our demonstration of the improved two-agent voting system was ease of use. It is important to impress on users the aesthetic qualities of the voting

system, so the implementation was designed to appear visually similar to the current mail-in system yet take advantage of the computer's power (e.g., input checking, network transmission, fast performance). This is another reason why the two-agent system was chosen in the first place; it is very similar to the mail-in system which the audience of the demonstration are familiar with.

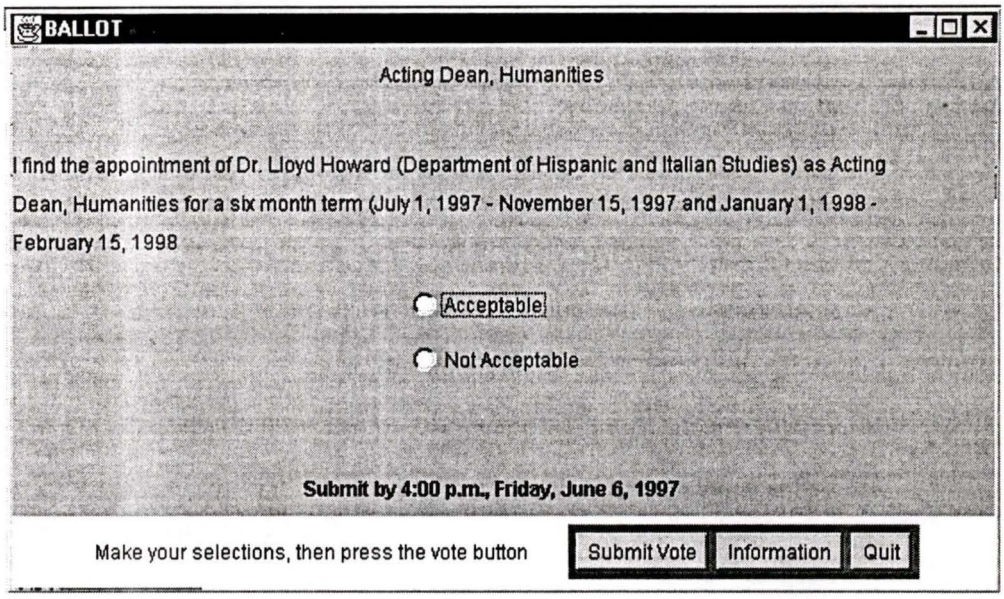


Figure 21. Screen shot of voting system implementation: YES/NO election

Other major design goals were simplicity and modularity from a coding perspective. The appeal of a simple design is mainly that the application can be easily understood and scrutinised by wary users. It also helps the debugging process and supports future enhancements. Modularity is desired primarily for debugging and extensibility. In addition, the application should use standards and existing tools as much as practicable to facilitate rapid deployment, a simplified modular design, and portability.

10.1 Design issues

The Java programming language was selected as the most suitable for developing the two-agent voting system application. Java's object-orientation simplifies graphical user interface development and encourages modular design in general. Its encapsulated networking classes make communications much easier to implement than in a language like C and its integrated multithreading capabilities are invaluable for an application such as an electronic voting system.

One of the greatest attractions of Java is that many applications can execute in compiled form on any platform that has a Java virtual machine and Java virtual machines are freely available for nearly every commercial computing platform. Another important feature of Java is that the most recent version of the *Java Developer's Kit*[39] (JDK), version 1.1, introduces a new security architecture and interfaces for incorporating cryptographic functions and other security tools into the language. JDK v1.1 also introduces many features that could be valuable for future enhancements to the application, such as remote method invocation (analogous to the *remote procedure call* concept used with procedural languages) and support for the JavaBeans component object architecture.

For this application, the validator and tallier servers were developed along with a voter application module. The registrar was not implemented because it is superfluous for a demonstration. Both the validator and the tallier are multithreaded to handle multiple simultaneous voters. They were also designed to handle multiple simultaneous elections: that is, one or more elections can be in progress at any time. All three applications run as *standalone* Java applications, as opposed to applets, so that they can engage in local file I/O (applets, which are Java application components that run under the control of a Web browser, are forbidden from accessing the file system of the computer which they are running on). The servers have simple command-line interfaces and display status information to the console so that the progress of the voting can be

[39] The Java Developer's Kit is the basic toolkit used by Java developers to create Java applications.

monitored. The voter interface is graphical and mimics typical ballot forms currently used in campus mail-in elections.

The interface of the voter application is defined as a separate class, called the *ballot class*,[40] which is responsible for the appearance of the ballot form, handling of events within it (e.g., clicking of buttons), and returning the results to the *voter class* in a prescribed format. The scheme is designed so that it is only necessary to change the file containing the compiled ballot class in order to allow the voter to participate in another election; the ballot class only needs to conform to published specifications to interface properly with the voter class. Using Java's remote method invocation feature, the voter application could be configured to allow the user to select an election from a list, then acquire the network location of the ballot class to invoke methods from.

The results, which describe the user's decisions from the ballot class to the voter class, are stored in a simple data structure: a byte value represents the choice made for each decision on the ballot. The interpretation of the byte values is election-specific but must be published in advance by the election administration so that the voter can verify from log information that the ballot class is properly setting the results. A simple example of the results data structure has the value of each byte representing, numerically, the ordinal selection made for the decision it is associated with (i.e., if the voter selects the third choice of the first decision and the second choice of the second decision, then the first result byte would contain the value 3 and the second byte would contain the value 2). A decision can also permit multiple choices in this scheme, as long as the rules restrict the total possible *combinations* for each decision to fewer than 256.

10.2 Implementation issues

A limitation of Java's JDK 1.1 is that it is currently only available for the Solaris and Windows NT platforms and the runtime component of Java version 1.1 is not readily

[40] Actually, another class is defined to provide an information dialogue box for the user. It is accessed from the ballot class and is provided by the election administration.

available yet either. This restricts demonstration of the voting system application to either of those platforms. Of course this will not be a problem for long as Java 1.1 will soon be deployed everywhere that 1.0 currently is (and may already be by the time of publishing).

It should also be noted that although JDK 1.1 was selected for its security architecture, none of its cryptographic tools was actually used in the demonstration application because their implementation and definition were still immature. The specification of the security component of Java is still in progress and, even worse, the implementation is incomplete, having only API's for digital signatures using the U.S. federal standard *Digital Signature Algorithm* (DSA) and two message digest algorithms, along with some key management tools. Unfortunately, the improved two-agent voting protocol requires an asymmetric (public) key algorithm that can also encrypt and decrypt data; DSA can only be used for digital signatures. A third-party package of Java classes was chosen for the demonstration application because it provides the basic public key algorithm functions required for this voting system. The code of the application is designed to quickly accommodate a different public key algorithm implementation with few necessary modifications, so algorithms now in the public domain should be considered.

The demonstration application is not a complete voting system. It needs much additional functionality and refinement before being suitable for practical use. A list of required enhancements is provided in the next section.

11. Conclusions & further work

11.1 Conclusions

There are some security problems in information systems at the University of Victoria which can be improved with cryptographic techniques. These problems persist because of practical more than technical limitations. In other words, the technology to remedy most of the major problems afflicting the University's information systems exists, but there are other issues, such as rules, regulations, internal politics, and public misinformation that inhibit adoption of cryptography-based solutions.

One problem of particular interest is that of our current conventional voting systems, which are not considered to be adequately secure. While electronic devices have been considered for voting systems for many years and computers have actually been used in electoral systems to some extent since the 1960s, comparatively recent developments in the field of computer science, most notably the invention of public key cryptography, have made completely automated secure voting systems possible.

We have examined four protocols for electronic voting systems and compared them to each other and to the conventional mail-in voting system currently used for departmental and faculty elections. All of the electronic protocols are generally superior to the conventional protocol in terms of security, performance, and intangible aesthetic qualities. Their main shortcoming is that they require all participants in the election to have and use a personal computer with a network connection. However, this is not a problem for the faculty of the University of Victoria (or any other Canadian university), especially in groups such as the Department of Computer Science where every faculty member has a computer. Also, there are many other organisations, such as large

companies and governments, where internal elections occur and personal computers are commonplace. It appears that the same will soon be true of society in general.

Two of the electronic protocols, the improved two-agent protocol and the SAVE protocol, were devised by the author to provide practical alternatives to the mail-in system and to other electronic systems. For instance, the improved two-agent protocol is modelled almost directly on the mail-in protocol but replaces its conventional elements with electronic media and logic. It uses cryptographic techniques to eliminate some of the weaknesses of the mail-in protocol, such as the susceptibility of the delivery system and the inability of voters to verify that their votes were counted. It was also found to be arguably no less secure than the Sensus system, which is based on the improved one-agent protocol, since both make trusting assumptions about how votes travel from voters to the tallier. We feel that, in the case of the improved two-agent protocol, these assumptions can be sustained by careful administration and control of the validator server.

The SAVE protocol was developed to improve on the performance of the zero-agent voting protocol without sacrificing its excellent security characteristics. In fact, the SAVE protocol actually improves on the zero-agent protocol because the tallier is the first to see the election results, not one of the voters. The performance gain of the SAVE protocol was approximately three times over the zero-agent protocol. However, the execution times of both protocols grow in proportion to the square of the size of the electorate, while the improved two-agent protocol and the Sensus protocol have execution times that grow no more than linearly with the electorate.

The electronic voting protocols were compared to the mail-in voting protocol but not to the polling booth system, which is also used at the University of Victoria (and beyond). Polling booth systems are usually preferred for large electorates in which voters may be difficult to reach at a fixed address. For example, at the University they are used for elections that involve the student body. Neither the zero-agent protocol nor the SAVE protocol are well-suited for replacing the polling booth protocol because of their sensitivity, in terms of performance, to large electorates and their need to be

restarted in the event of tampering. The improved two-agent protocol and the Sensus protocol are better suited to replacing the polling booth protocol.

In general, electronic voting systems can improve upon the conventional systems that are in use today because they offer stronger security, dramatically faster turnaround times, and significant cost savings. An important quality that can be argued for all of the electronic voting systems examined is that, because of their improvements in security, interface, performance, and convenience, together with their suitability for frequent use, they would make elections far more appealing to the electorate. This may encourage voters to attach greater value and significance to the results and to the democratic process. Consequently, it would help to improve voters' comfort with the concept of, and thus establish a demand for, direct democracy. This is important because a trend toward direct democracy is not likely to emerge from within the political mainstream; few politicians [16] would champion an electoral system which threaten to reduce their control over societal decision making.

11.2 Further work

The intent of the improved two-agent voting system application was to demonstrate the usability of an electronic voting implementation and to give some idea of its performance characteristics. Although it adds multiple simultaneous election handling to the basic protocol, this implementation is decidedly lean, providing only the critical vote processing, cryptographic procedures, networking functionality, and user interface characteristics. Voters complete ballots for any current election (one that has been defined and declared by the appropriate authorities) and submit the ballots to a validator elsewhere on the network. The validator verifies the validity of the source of the ballot and sends it to the tallier (also elsewhere on the network), then sends a receipt to the voter. Finally, the tallier examines and verifies the contents of the ballot, adds the vote to the election tally, then sends a receipt the voter by way of the validator. All messages are checked for integrity but no significant handling is performed for those that are not

well-formed. The following is a partial list of suggested improvements for the application:

1. All necessary functions need to be completed, including providing a registrar and adding the registration phases of the protocol. Also, handling of contingencies, such as resubmitting and challenging a vote, must be added.
2. Error trapping and handling must be completed; it is minimal at this point.
3. The format and content of configuration files that the agents read upon start-up and during their operation must be resolved to provide all of the information that the agents will need.
4. Logging of all transactions by the three servers and the voter client application is important and should be provided so that each can maintain a transaction history for every election that they participate in. For the voter client, this would include election-specific data such as the temporary voter ID and the public-secret key pair.
5. Greater operational efficiencies can be sought by using faster implementations of encryption algorithms (or faster algorithms), more efficient I/O (e.g., buffering), performance adjustments to the code and to the protocol, and run-time enhancements (e.g., using a *Just-In-Time* Java interpreter).
6. If possible, the voting application should be integrated with Web technology (e.g., as Java applets). This would allow it to be used with Web browsers, which are ubiquitous and well understood. It would also allow the application **and** the protocol to take advantage of the SSL feature to enhance overall system security. Currently it is not possible to run the voting application as Java applets under Web browsers because the browsers do not yet permit local file I/O (for security purposes). This will change when a comprehensive Web site certification scheme is incorporated into the Java language.
7. Scheduled operation of the voting system could produce several benefits, particularly for the convenience of the voter who would no longer have to "stand by" during the balloting process. Instead, the voter could complete the ballot at any time, even before the polls opened, and have the voting application perform the necessary

actions, log important information, and take appropriate recourses at the scheduled time.

8. Remote method invocation should be used to dynamically load ballot forms (i.e., the ballot class). Currently, the ballot class must be manually placed into the same directory as the voter class, replacing any ballot class that was used for another election.

Bibliography

[1] Adams, J., Malik, W., and Lynch, G. "Workstation -Based vs. Server-Based Single Sign-On," *GartnerGroup Monthly Research Review*, GartnerGroup Inc., January 1, 1996.

[2] Atkinson, R. "Security Architecture for the Internet Protocol," *Network Working Group*, RFC 1825, Standards Track, Naval Research Laboratory, August 1995.

[3] Beck, A. "Netscape's Export SSL Broken by 120 Workstations and One Student," article appearing in *HPCwire* Internet magazine, Tabor Griffin Communications, ©1995.

[4] Benaloh, J., and Tuinstra, D. "Receipt-free Secret-ballot Elections," *Proceedings of the Twenty-sixth Annual ACM Symposium on the Theory of Computing*, May 23-25, 1994, pp. 544-553.

[5] Brams, Steven J. and Fishburn, Peter C. "Does Approval Voting Elect the Lowest Common Denominator?" *PS: Political Science & Politics*, volume 21, pp. 277-284.

[6] C2Net Software Inc., "Hackers Smash U.S. Government Encryption Standard," Company Press Release, June 18, 1997.

[7] Campbell, V. "Democracy by Telephone: An Alternative to Revolution," Avant/Garde Magazine, January 1970.

[8] CCITT, Recommendation X.509, "The Directory-Authentication Framework," Comite Consultatif de Internationale Telegraphie et Telephonie, International Telecommunications Union, Geneva, 1989.

[9] Chaum, David. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Communications of the ACM*, Volume 24, Number 2, February 1981.

[10] Chaum, David. "Elections with Unconditionally Secret Ballots and Disruptions Equivalent to Breaking RSA," *Advances in Cryptology --- EUROCRYPT '88 Proceedings*, Springer-Verlag, 1988, pp. 177-181.

[11] Chaum, David. "Blind Signature Systems." U.S. Patent #4759063, July 19, 1988.

[12] Chen, L. and Burminster, M. "A Practical Secret Voting Scheme which Allows Voters to Abstain," *CHINACRYPT '94,* Xidian, China, November 1994, pp. 100-107.

[13] Coulouris, George, Dollimore, Jean, and Kindberg, Tim. *Distributed Systems: Concepts and Design*, Addison Wesley, London, 1994, p. 491.

[14] Cramer, R.J.F. and Pedersen, T.P. "Improved Privacy in Wallets with Observers," *Advances in Cryptology---EUROCRYPT '93 Proceedings*, Sprintger-Verlag, 1994, pp. 329-343.

[15] Cranor, L, and Cytron, R. "Design and Implementation of a Practical Security-Conscious Electronic Polling System," Technical Paper, Department of Computer Science, Washington University, St. Louis, January 1996.

[16] Democratech: © 1995, Demcratech Party of British Columbia.

[17] Denning, D. "The Future of Cryptography," *Internet Security Review*, Internet Security Inc., October 1995.

[18] Department of State (U.S.A.), "International Traffic in Arms Regulations (ITAR)," 22 CFR 120-130, Office of Munitions Control, November 1989.

[19] Diffie, W., and Hellman, M. "New Directions in Cryptography," *IEEE Transactions on Information Theory*, v. IT-22, n. 6, Nov 1976, pp. 644-654.

[20] The Economist "Democracy and Technology," Editorial article in *The Economist*, June 17th 1995, pp 21-23.

[21] Fujioka, A., Okamoto, T., and K. Ohta, "A Practical Secret Voting Scheme for Large Scale Elections," *Advances in Cryptology --- EUROCRYPT '91 Proceedings*, Springer-Verlag, 1991, pp. 243-256.

[22] Fuller, R. Buckminster "No More Secondhand God; and other writings," Anchor Books, Garden City, N.Y., 1971 (©1963), pp. 10-17.

[23] Hind, P. "Single Sign-On, Multiple Decision," *Infosecurity News*, Feb. 1997.

[24] Kohl, J., Neuman, B., and Ts'o, T. "The Evolution of the *Kerberos* Authentication Service," present at *EurOpen Spring 1991* conference, 1991.

[25] Merriman, W. Richard "To Collect the Wisest Sentiments: Representative Government and Direct Democracy," discussion guide from *The Jefferson Meeting on the Constitution project*, The Jefferson Foundation, 1986.

[26] Merritt, Michael "Cryptographic Protocols", Ph.D. dissertation, Georgia Institute of Technology, GIT-ICS-83/6, February 1983.

[27] Miller, S., Neuman, B., Schiller, J., and Saltzer, J. "Section E.2.1: Kerberos Authentication and Authorization System," Project Athena, MIT, December 1987.

[28] National Institute of Standards and Technology, NIST FIPS PUB 185, "Escrowed Encryption Standard," U.S. Department of Commerce, February 1994.

[29] Needham, B. "A better way to vote: Why letting the people themselves take the decisions is the logical next step for the West," *The Economist*, September 11, 1993.

[30] Nurmi, H., Salomaa, A., and Santean, L. "Secret Ballot Elections in Computer Networks," *Computers & Security*, v. 10, 1991, pp. 553-560.

[31] Plice, Samuel "Why Your Campus Should Consider Adopting OSF's DCE Standards," *CAUSE/EFFECT*, Volume 18 Number 1, Spring 1995, pp. 5-7.

[32] Rankin, Murray. "Questions & Answers about Freedom of Information and Privacy Protection," The B.C. Freedom of Information and Privacy Association, 1995.

[33] Ravitz, E. "Telephone voting would foster democracy," Boulder Sunday Camera, Boulder, Col, May 16, 1993.

[34] Reed, L. "Hang Up On Vote By Phone," Mackinac Center for Public Policy, No 91-05, April 14, 1991.

[35] RSA Laboratories "PKCS #1: RSA Encryption Standard," version 1.5, November 1993.

[36] RSA Data Security *BSAFE™ 3.0 Benchmarks*, RSA Data Security Inc., Revised July 1, 1996.

[37] RSA Data Security "How Secure is DES?" article posted on the World-Wide Web (URL http:\\www.rsa.com), RSA Data Security, Inc., © 1997.

[38] Schneier, Bruce *Applied Cryptography.* 2nd ed. New York, NY: Wiley, 1996.

[39] Speight, J. *Electronic Politics,* Traffor Publishing, Victoria, B.C., 1996.

[40] Yuochunas, Nancy "DCE: A Foundation for Administrative Software Collaboration in Higher Education," *CAUSE/EFFECT*, Volume 18 Number 1, Sprint 1995, pp. 8-9.

Appendices

Appendix A - A misconception about single sign-on

A common argument against single sign-on is that it increases the potential for damage by an attacker since the compromise of only one password exposes the entire set of services and resources of a user. Everything else being equal, single sign-on **does** weaken overall security if a workstation-based solution is used where all accounts and passwords are like-named. All an attacker needs to do is subvert the security of any account, whether it is the weakest or the most accessible. For example, if a user has accounts on several UNIX systems, a POP e-mail service, and a secure Web site, a hacker might target the e-mail service by monitoring the communications lines, setting up an impostor service, or perhaps exposing a bug in the client or server software. Of course this problem is the same in a non-single sign-on environment where users elect to set their passwords to a single value anyway.

This problem does not exist in a workstation-based single sign-on solution where accounts and passwords are varied because this situation is essentially the same as what users with multiple accounts are in today; the compromise of one account does not necessarily mean the compromise of any other account belonging to the user. There **is** such a threat if an attacker manages to break into the account manager on the user's system where all account names and passwords for the different services is stored. However, this locus of control can be made extremely difficult to break into by using strong encryption, access control, physical access limitations, and other security techniques. This is not the case for typical services that a user might have access to,

such as a UNIX account, where the security is predefined by a standard (de facto or de jure) and cannot be easily modified.

For server-based single sign-on, the secrecy of a user's password also of extreme importance because it protects all of a user's services and resources. However, this type of solution usually provides a substantial security improvement over the weakest security systems of the conventional alternatives. Some reasons are that cryptographic algorithms are stronger (e.g., RC-128 instead of DES), secure communications channels are used (e.g., data is encrypted when sent on the network, including passwords), and consistent security policies (e.g., passwords must be at least X characters in length and changed every Y days) are more easily established and enforced.

Another fact to consider is that when users have many accounts and choose not to (or are unable to) set all passwords the same, then they will tend to either choose easily remembered (and easily guessed) passwords or will store the account names and passwords somewhere. Either action heightens the susceptibility of the user's account security.

Finally, if one or more of a user's services are considered too sensitive to be in the same security pool as the other services, then there is no technical reason why they cannot be provided through separate accounts. This does diminish the account unification goal of single sign-on but it shows that the concept does not have to be all-or-nothing; two accounts are better than ten.

Appendix B - Stuffing the ballot box

Often, some eligible voters do not cast a vote or do not even register. When this happens, an unscrupulous agent can cast votes "on behalf" of those voters. This is called *vote stuffing* and, in voting systems without support for detection of a miscount, including traditional conventional systems, there is no way for anyone to stop this malpractice *as long as the posted results do not enumerate beyond the size of the known voting population.* Usually the tallier has opportunity to stuff votes but in some systems the validator may be able to as well.

If *strong detection*, where any voter can verify that **all** of the votes were counted correctly, is used, then voters may be able to detect vote stuffing, especially if non-participating voters submit an empty ballot instead of abstaining altogether. Unfortunately, because *strong detection* is very complicated, most systems use *weak detection*,. where voters can only verify that **their own** vote was counted correctly. In this scheme, voters cannot know if any particular vote in the final postings was actually fraudulent. Even the non-voting voters for whom the fraudulent votes were cast cannot know that this happened.

One preventative measure for this kind of fraud would be to require all eligible voters to vote. Unfortunately, this may not be enforceable. A better approach is to utilise the registrar. Assuming the registrar is independent, honest, and does not reveal its count of registered voters before the results are posted, then the tallier and the validator will only know the that the number of registered voters lies between the number of votes cast and the size of the voting population, inclusive. For every fraudulent vote added, there is an added risk that the final count will exceed the number of registered voters, in which case the registrar will discover the fraud (unless the registrar colludes with the other agents).

Appendix C - Challenging a miscounted vote

Two popular issues in designing electronic voting systems are how to enable voters to verify that their votes were correctly tallied and how voters can take action to rectify miscounted votes. A common solution to the *detection* issue requires the tallier to post each vote together with some information that only the person who cast the vote can identify with. The *correction* issue requires the voter to be able to present proof that her vote was received in good order, usually in the form of a digitally-signed electronic receipt. Correction becomes complex if it demands that we preserve the anonymity of the affected voter.

When votes are posted, the results must match what is posted. Therefore, the final tally can only be misrepresentative because of legitimate votes being left out or misrepresented, or because of vote stuffing. The first two cases will be easily detected by any affected voter. The last case is more difficult to detect but can be made extremely risky, as described in the previous appendix. It is loosely related to this topic so it is not discussed here.

In the simplest case where a miscount occurred because of an unintentional error by the tallier[41], the affected voter can resubmit the original ballot or a receipt, or both, after detecting the error. The tallier can then make the adjustment to the posted results. However, the likelihood of a miscount in a system that uses modern information technology, including error correction and guaranteed delivery schemes, can be virtually nullified. In such a system, any improperly counted votes will undoubtedly be caused by a malicious tallier who hopes that some or all of the affected voters will be (i) reluctant to challenge the tallier, (ii) not concerned enough to challenge the tallier, or (iii) unaware that their votes were changed.

---

[41] Note that we only mention the tallier here as the perpetrator, even though it is possible for the validator to affect votes in some systems. Where this is true, the defences against cheating by the tallier can be extended to apply to the validator without too much effort.

Presumably, should a voter discover that her vote was not correctly counted, she will proceed to challenge the tallier as described above. Most authors of electronic voting systems only describe the challenge process to this extent. However, any miscount in a well-designed electronic voting system is the result of malicious intent by the tallier and, if detected, will reveal the tallier as a cheat to the affected voter. The tallier is only likely to attempt a deliberate miscount if it is comfortable with the voter knowing it is dishonest and is confident that the voter will not extend this information to others. The tallier's strategy in this case will be to basically ignore the voter's challenge. Ultimately, the voter can only prove to others that the tallier cheated by revealing her vote.

Fortunately, some voters are willing to reveal their votes to others, as evidenced by the many signs that clutter lawns during national and provincial elections. As long as the tallier is not sure of who is willing to reveal their votes, it should be reluctant to risk a miscount. Therefore, the purpose of a vote challenge scheme in an electronic voting system is effectively to dissuade the tallier from cheating, not to correct a miscounted vote.

Note that this challenge process is based a form of miscount detection known as *weak detection*, where voters can only verify that **their own** votes were counted correctly. This necessarily places the burden of proof upon the affected voter. The alternative is *strong detection*, where any voter can verify that **all** of the votes were counted correctly. Strong detection would eliminate the threat of miscounting. Unfortunately, it is very complicated to implement in practice.

Appendix D - Voting by telephone

There are many devices that can be used for electronic voting, including the telephone. Voting by telephone was conceived and promoted decades ago [22] but started to become popular for discussion in the 1980s. Proponents argued that the ubiquity of suitable touch-tone telephony equipment make this an ideal medium for voting. The physical interface is well-known and the necessary logical interface is reasonably simple and somewhat familiar to most people (i.e., touch-tone response systems). Supporters [7][22][33][39] claim that, compared to conventional systems, voting by telephone is more economical, more ecological, and potentially more secure; most importantly, it will enhance the democratic process by attracting higher voter turnouts and by making more frequent elections feasible. A well-publicised case to introduce voting by telephone in a significant electoral system was a 1991 initiative in Boulder, Colorado for municipal elections.

However, telephones are far from secure. Bill Kimberling, deputy director of the U.S. Federal Election Commission's National Clearinghouse on Election Administration, is sceptical. Citing the likelihood of voter fraud and problems of security and privacy, Kimberling thinks proponents of large-scale telephone voting "have been watching Star Trek a little too much." Another strong opposition comes from the Mackinac Center for Public Policy [34].

The main problem is that telephones communicate in cleartext and can be easily tapped somewhere between the two endpoints. Telephones in offices and in multiple-unit dwellings, as well as wireless handsets, are especially susceptible to tapping. *Privacy-enhanced* telephones, which encrypt communications, **are** available today but are expensive, proprietary in the absence of a standard, and are certainly not in widespread use. Furthermore, there is no provision in the average telephone **or** in a *privacy-enhanced* telephone for obtaining a receipt that could be used to verify the voter's vote.

Index

VITA

Surname: Will Given Names: Mark Robert

Place of Birth: Prince Rupert, British Columbia, Canada

Educational Institutions Attended:

University of Victoria	1992 to 1997
University of Victoria	1985 to 1990
British Columbia Institute of Technology	1982 to 1984

Degrees Awarded:

B.Sc.	University of Victoria	1990

Honours and Awards:

University of Victoria Fellowship	1995-96
Graduated first class	1990
B.C. Government Scholarships	1985-86

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis:
Information Security in a Campus Computing Environment

Author 

Mark Robert Will

November 20, 1997