Chapter 6

Esoteric Protocols

6.1 Secure Elections

Computerized voting will never be used for general elections unless there is a protocol that both maintains individual privacy and prevents cheating. The ideal protocol has, at the very least, these six requirements:

1. Only authorized voters can vote.
2. No one can vote more than once.
3. No one can determine for whom anyone else voted.
4. No one can duplicate anyone else's vote. (This turns out to be the hardest requirement.)
5. No one can change anyone else's vote without being discovered.
6. Every voter can make sure that his vote has been taken into account in the final tabulation.

Additionally, some voting schemes may have the following requirement:

7. Everyone knows who voted and who didn't.

Before describing the complicated voting protocols with these characteristics, let's look at some simpler protocols.

Simplistic Voting Protocol #1

(1) Each voter encrypts his vote with the public key of a Central Tabulating Facility (CTF).
(2) Each voter sends his vote in to the CTF.
(3) The CTF decrypts the votes, tabulates them, and makes the results public.

This protocol is rife with problems. The CTF has no idea where the votes are from, so it doesn't even know if the votes are coming from eligible voters. It has no idea if eligible voters are voting more than once. On the plus side, no one can change anyone else's vote; but no one would bother trying to modify someone else's vote when it is far easier to vote repeatedly for the result of your choice.

Simplistic Voting Protocol #2

(1) Each voter signs his vote with his private key.
(2) Each voter encrypts his signed vote with the CTF's public key.
(3) Each voter sends his vote to a CTF.
(4) The CTF decrypts the votes, checks the signatures, tabulates the votes, and makes the results public.

This protocol satisfies properties one and two: Only authorized voters can vote and no one can vote more than once—the CTF would record votes received in step (3). Each vote is signed with the voter's private key, so the CTF knows who voted, who didn't, and how often each voter voted. If a vote comes in that isn't signed by an eligible voter, or if a second vote comes in signed by a voter who has already voted, the facility ignores it. No one can change anyone else's vote either, even if they intercept it in step (3), because of the digital signature.

The problem with this protocol is that the signature is attached to the vote; the CTF knows who voted for whom. Encrypting the votes with the CTF's public key prevents anyone from eavesdropping on the protocol and figuring out who voted for whom, but you have to trust the CTF completely. It's analogous to having an election judge staring over your shoulder in the voting booth.

These two examples show how difficult it is to achieve the first three requirements of a secure voting protocol, let alone the others.

Voting with Blind Signatures

We need to somehow dissociate the vote from the voter, while still maintaining authentication. The blind signature protocol does just that.

(1) Each voter generates 10 sets of messages, each set containing a valid vote for each possible outcome (e.g., if the vote is a yes or no question, each set contains two votes, one for "yes" and the other for "no"). Each message also contains a randomly generated identification number, large enough to avoid duplicates with other voters.
(2) Each voter individually blinds all of the messages (see Section 5.3) and sends them, with their blinding factors, to the CTF.
(3) The CTF checks its database to make sure the voter has not submitted his blinded votes for signature previously. It opens nine of the sets to check that they are properly formed. Then it individually signs each message in the set. It sends them back to the voter, storing the name of the voter in its database.

(4) The voter unblinds the messages and is left with a set of votes signed by the CTF. (These votes are signed but unencrypted, so the voter can easily see which vote is "yes" and which is "no.")
(5) The voter chooses one of the votes (ah, democracy) and encrypts it with the CTF's public key.
(6) The voter sends his vote in.
(7) The CTF decrypts the votes, checks the signatures, checks its database for a duplicate identification number, saves the serial number, and tabulates the votes. It publishes the results of the election, along with every serial number and its associated vote.

A malicious voter, call him Mallory, cannot cheat this system. The blind signature protocol ensures that his votes are unique. If he tries to send in the same vote twice, the CTF will notice the duplicate serial number in step (7) and throw out the second vote. If he tries to get multiple votes signed in step (2), the CTF will discover this in step (3). Mallory cannot generate his own votes because he doesn't know the facility's private key. He can't intercept and change other people's votes for the same reason.

The cut-and-choose protocol in step (3) is to ensure that the votes are unique. Without that step, Mallory could create a set of votes that are the same except for the identification number, and have them all validated.

A malicious CTF cannot figure out how individuals voted. Because the blind signature protocol prevents the facility from seeing the serial numbers on the votes before they are cast, the CTF cannot link the blinded vote it signed with the vote eventually cast. Publishing a list of serial numbers and their associated votes allows voters to confirm that their vote was tabulated correctly.

There are still problems. If step (6) is not anonymous and the CTF can record who sent in which vote, then it can figure out who voted for whom. However, if it receives votes in a locked ballot box and then tabulates them later, it cannot. Also, while the CTF may not be able to link votes to individuals, it can generate a large number of signed, valid votes and cheat by submitting those itself. And if Alice discovers that the CTF changed her vote, she has no way to prove it. A similar protocol, which tries to correct these problems, is [1195,1370].

Voting with Two Central Facilities

One solution is to divide the CTF in two. Neither party would have the power to cheat on its own.

The following protocol uses a Central Legitimization Agency (CLA) to certify voters and a separate CTF to count votes [1373].

(1) Each voter sends a message to the CLA asking for a validation number.
(2) The CLA sends the voter back a random validation number. The CLA maintains a list of validation numbers. The CLA also keeps a list of the validation numbers' recipients, in case someone tries to vote twice.

(3) The CLA sends the list of validation numbers to the CTF.

(4) Each voter chooses a random identification number. He creates a message with that number, the validation number he received from the CLA, and his vote. He sends this message to the CTF.

(5) The CTF checks the validation number against the list it received from the CLA in step (3). If the validation number is there, the CTF crosses it off (to prevent someone from voting twice). The CTF adds the identification number to the list of people who voted for a particular candidate and adds one to the tally.

(6) After all votes have been received, the CTF publishes the outcome, as well as the lists of identification numbers and for whom their owners voted.

Like the previous protocol, each voter can look at the lists of identification numbers and find his own. This gives him proof that his vote was counted. Of course, all messages passing among the parties in the protocol should be encrypted and signed to prevent someone from impersonating someone else or intercepting transmissions.

The CTF cannot modify votes because each voter will look for his identification string. If a voter doesn't find his identification string, or finds his identification string in a tally other than the one he voted for, he will immediately know there was foul play. The CTF cannot stuff the ballot box because it is being watched by the CLA. The CLA knows how many voters have been certified and their validation numbers, and will detect any modifications.

Mallory, who is not an eligible voter, can try to cheat by guessing a valid validation number. This threat can be minimized by making the number of possible validation numbers much larger than the number of actual validation numbers: 100-digit numbers for a million voters, for example. Of course, the validation numbers must be generated randomly.

Despite this, the CLA is still a trusted authority in some respects. It can certify ineligible voters. It can certify eligible voters multiple times. This risk could be minimized by having the CLA publish a list of certified voters (but not their validation numbers). If the number of voters on this list is less than the number of votes tabulated, then something is awry. However, if more voters were certified than votes tabulated, it probably means that some certified people didn't bother voting. Many people who are registered to vote don't bother to cast ballots.

This protocol is vulnerable to collusion between the CLA and the CTF. If the two of them got together, they could correlate databases and figure out who voted for whom.

Voting with a Single Central Facility

A more complex protocol can be used to overcome the danger of collusion between the CLA and the CTF [1373]. This protocol is identical to the previous one, with two modifications:

— The CLA and the CTF are one organization, and

— ANDOS (see Section 4.13) is used to anonymously distribute validation numbers in step (2).

Since the anonymous key distribution protocol prevents the CTF from knowing which voter got which validation number, there is no way for the CTF to correlate validation numbers with votes received. The CTF still has to be trusted not to give validation numbers to ineligible voters, though. You can also solve this problem with blind signatures.

Improved Voting with a Single Central Facility

This protocol also uses ANDOS [1175]. It satisfies all six requirements of a good voting protocol. It doesn't satisfy the seventh requirement, but has two properties additional to the six listed at the beginning of the section:

7. A voter can change his mind (i.e., retract his vote and vote again) within a given period of time.
8. If a voter finds out that his vote is miscounted, he can identify and correct the problem without jeopardizing the secrecy of his ballot.

Here's the protocol:

(1) The CTF publishes a list of all legitimate voters.

(2) Within a specified deadline, each voter tells the CTF whether he intends to vote.

(3) The CTF publishes a list of voters participating in the election.

(4) Each voter receives an identification number, I, using an ANDOS protocol.

(5) Each voter generates a public-key/private-key key pair: k, d. If v is the vote, he generates the following message and sends it to the CTF:

$I,E_k(I,v)$

This message must be sent anonymously.

(6) The CTF acknowledges receipt of the vote by publishing:

$E_k(I,v)$

(7) Each voter sends the CTF:

I,d

(8) The CTF decrypts the votes. At the end of the election, it publishes the results of the election and, for each different vote, the list of all $E_k(I,v)$ values that contained that vote.

(9) If a voter observes that his vote is not properly counted, he protests by sending the CTF:

$I,E_k(I,v),d$

(10) If a voter wants to change his vote (possible, in some elections) from v to v', he sends the CTF:

$I,E_k(I,v'),d$

A different voting protocol uses blind signatures instead of ANDOS, but is essentially the same [585]. Steps (1) through (3) are preliminary to the actual voting. Their purpose is to find out and publicize the total number of actual voters. Although some of them probably will not participate, it reduces the ability of the CTF to add fraudulent votes.

In step (4), it is possible for two voters to get the same identification number. This possibility can be minimized by having far more possible identification numbers than actual voters. If two voters submit votes with the same identification tag, the CTF generates a new identification number, I', chooses one of the two votes, and publishes:

$$I', E_k(I, v)$$

The owner of that vote recognizes it and sends in a second vote, by repeating step (5), with the new identification number.

Step (6) gives each voter the capability to check that the CTF received his vote accurately. If his vote is miscounted, he can prove his case in step (9). Assuming a voter's vote is correct in step (6), the message he sends in step (9) constitutes a proof that his vote is miscounted.

One problem with the protocol is that a corrupt CTF could allocate the votes of people who respond in step (2) but who do not actually vote. Another problem is the complexity of the ANDOS protocol. The authors recommend dividing a large population of voters into smaller populations, such as election districts.

Another, more serious problem is that the CTF can neglect to count a vote. This problem cannot be resolved: Alice claims that the CTF intentionally neglected to count her vote, but the CTF claims that the voter never voted.

Voting without a Central Tabulating Facility

The following protocol does away with the CTF entirely; the voters watch each other. Designed by Michael Merritt [452,1076,453], it is so unwieldy that it cannot be implemented practically for more than a handful of people, but it is useful to learn from nevertheless.

Alice, Bob, Carol, and Dave are voting yes or no (0 or 1) on a particular issue. Assume each voter has a public and private key. Also assume that everyone knows everyone else's public keys.

(1) Each voter chooses his vote and does the following:
   (a) He attaches a random string to his vote.
   (b) He encrypts the result of step (a) with Dave's public key.
   (c) He encrypts the result of step (b) with Carol's public key.
   (d) He encrypts the result of step (c) with Bob's public key.
   (e) He encrypts the result of step (d) with Alice's public key.
   (f) He attaches a new random string to the result of step (e) and encrypts it with Dave's public key. He records the value of the random string.
   (g) He attaches a new random string to the result of step (f) and encrypts it with Carol's public key. He records the value of the random string.

(h) He attaches a new random string to the result of step (g) and encrypts it with Bob's public key. He records the value of the random string.

(i) He attaches a new random string to the result of step (h) and encrypts it with Alice's public key. He records the value of the random string.

If E is the encryption function, R_i is a random string, and V is the vote, his message looks like:

$$E_A(R_5,E_B(R_4,E_C(R_3,E_D(R_2,E_A(E_B(E_C(E_D(V,R_1))))))))$$

Each voter saves the intermediate results at each point in the calculation. These results will be used later in the protocol to confirm that his vote is among those being counted.

(2) Each voter sends his message to Alice.

(3) Alice decrypts all of the votes with her private key and then removes all of the random strings at that level.

(4) Alice scrambles the order of all the votes and sends the result to Bob.
Each vote now looks like this:

$$E_B(R_4,E_C(R_3,E_D(R_2,E_A(E_B(E_C(E_D(V,R_1)))))))$$

(5) Bob decrypts all of the votes with his private key, checks to see that his vote is among the set of votes, removes all the random strings at that level, scrambles all the votes, and then sends the result to Carol.
Each vote now looks like this:

$$E_C(R_3,E_D(R_2,E_A(E_B(E_C(E_D(V,R_1))))))$$

(6) Carol decrypts all of the votes with her private key, checks to see that her vote is among the set of votes, removes all the random strings at that level, scrambles all the votes, and then sends the result to Dave.
Each vote now looks like this:

$$E_D(R_2,E_A(E_B(E_C(E_D(V,R_1)))))$$

(7) Dave decrypts all of the votes with his private key, checks to see that his vote is among the set of votes, removes all the random strings at that level, scrambles all the votes, and sends them to Alice.
Each vote now looks like this:

$$E_A(E_B(E_C(E_D(V,R_1))))$$

(8) Alice decrypts all the votes with her private key, checks to see that her vote is among the set of votes, signs all the votes, and then sends the result to Bob, Carol, and Dave.
Each vote now looks like this:

$$S_A(E_B(E_C(E_D(V,R_1))))$$

(9) Bob verifies and deletes Alice's signatures. He decrypts all the votes with his private key, checks to see that his vote is among the set of votes, signs all the votes, and then sends the result to Alice, Carol, and Dave.
Each vote now looks like this:

$$S_B(E_C(E_D(V,R_1)))$$

(10) Carol verifies and deletes Bob's signatures. She decrypts all the votes with her private key, checks to see that her vote is among the set of votes, signs all the votes, and then sends the result to Alice, Bob, and Dave.
Each vote now looks like this:

$S_C(E_D(V,R_1))$

(11) Dave verifies and deletes Carol's signatures. He decrypts all the votes with his private key, checks to see that his vote is among the set of votes, signs all the votes, and then sends the result to Alice, Bob, and Carol.
Each vote now looks like this:

$S_D(V,R_1)$

(12) All verify and delete Dave's signature. They check to make sure that their vote is among the set of votes (by looking for their random string among the votes).

(13) Everyone removes the random strings from each vote and tallies the votes.

Not only does this protocol work, it is also self-adjudicating. Alice, Bob, Carol, and Dave will immediately know if someone tries to cheat. No CTF or CLA is required. To see how this works, let's try to cheat.

If someone tries to stuff the ballot, Alice will detect the attempt in step (3) when she receives more votes than people. If Alice tries to stuff the ballot, Bob will notice in step (4).

More devious is to substitute one vote for another. Since the votes are encrypted with various public keys, anyone can create as many valid votes as needed. The decryption protocol has two rounds: round one consists of steps (3) through (7), and round two consists of steps (8) through (11). Vote substitution is detected differently in the different rounds.

If someone substitutes one vote for another in round two, his actions are discovered immediately. At every step the votes are signed and sent to all the voters. If one (or more) of the voters noticed that his vote is no longer in the set of votes, he immediately stops the protocol. Because the votes are signed at every step, and because everyone can backtrack through the second round of the protocol, it is easy to detect who substituted the votes.

Substituting one vote for another during round one of the protocol is more subtle. Alice can't do it in step (3), because Bob, Carol, or Dave will detect it in step (5), (6), or (7). Bob could try in step (5). If he replaces Carol's or Dave's vote (remember, he doesn't know which vote corresponds to which voter), Carol or Dave will notice in step (6) or (7). They wouldn't know who tampered with their vote (although it would have had to be someone who had already handled the votes), but they would know that their vote was tampered with. If Bob is lucky and picks Alice's vote to replace, she won't notice until the second round. Then, she will notice her vote missing in step (8). Still, she would not know who tampered with her vote. In the first round, the votes are shuffled from one step to the other and unsigned; it is impossible for anyone to backtrack through the protocol to determine who tampered with the votes.

Another form of cheating is to try to figure out who voted for whom. Because of the scrambling in the first round, it is impossible for someone to backtrack through the protocol and link votes with voters. The removal of the random strings during the first round is also crucial to preserving anonymity. If they are not removed, the scrambling of the votes could be reversed by re-encrypting the emerging votes with the scrambler's public key. As the protocol stands, the confidentiality of the votes is secure.

Even more strongly, because of the initial random string, R_1, even identical votes are encrypted differently at every step of the protocol. No one knows the outcome of the vote until step (11).

What are the problems with this protocol? First, the protocol has an enormous amount of computation. The example described had only four voters and *it* was complicated. This would never work in a real election, with tens of thousands of voters. Second, Dave learns the results of the election before anyone else does. While he still can't affect the outcome, this gives him some power that the others do not have. On the other hand, this is also true with centralized voting schemes.

The third problem is that Alice can copy anyone else's vote, even though she does not know what it is beforehand. To see why this could be a problem, consider a three-person election between Alice, Bob, and Eve. Eve doesn't care about the result of the election, but she wants to know how Alice voted. So she copies Alice's vote, and the result of the election is guaranteed to be equal to Alice's vote.

Other Voting Schemes

Many complex secure election protocols have been proposed. They come in two basic flavors. There are mixing protocols, like "Voting without a Central Tabulating Facility," where everyone's vote gets mixed up so that no one can associate a vote with a voter.

There are also divided protocols, where individual votes are divided up among different tabulating facilities such that no single one of them can cheat the voters [360,359,118,115]. These protocols only protect the privacy of voters to the extent that different "parts" of the government (or whoever is administering the voting) do not conspire against the voter. (This idea of breaking a central authority into different parts, who are only trusted when together, comes from [316].)

One divided protocol is [1371]. The basic idea is that each voter breaks his vote into several shares. For example, if the vote were "yes" or "no," a 1 could indicate "yes" and a 0 could indicate "no"; the voter would then generate several numbers whose sum was either 0 or 1. These shares are sent to tabulating facilities, one to each, and are also encrypted and posted. Each center tallies the shares it receives (there are protocols to verify that the tally is correct) and the final vote is the sum of all the tallies. There are also protocols to ensure that each voter's shares add up to 0 or 1.

Another protocol, by David Chaum [322], ensures that voters who attempt to disrupt the election can be traced. However, the election must then be restarted without the interfering voter; this approach is not practical for large-scale elections.

Another, more complex, voting protocol that solves some of these problems can be found in [770,771]. There is even a voting protocol that uses multiple-key ciphers

[219]. Yet another voting protocol, which claims to be practical for large-scale elections, is in [585]. And [347] allows voters to abstain.

Voting protocols work, but they make it easier to buy and sell votes. The incentives become considerably stronger as the buyer can be sure that the seller votes as promised. Some protocols are designed to be **receipt-free**, so that it is impossible for a voter to prove to someone else that he voted in a certain way [117,1170,1372].

6.2 SECURE MULTIPARTY COMPUTATION

Secure multiparty computation is a protocol in which a group of people can get together and compute any function of many variables in a special way. Each participant in the group provides one or more variables. The result of the function is known to everyone in the group, but no one learns anything about the inputs of any other members other than what is obvious from the output of the function. Here are some examples:

Protocol #1

How can a group of people calculate their average salary without anyone learning the salary of anyone else?

(1) Alice adds a secret random number to her salary, encrypts the result with Bob's public key, and sends it to Bob.

(2) Bob decrypts Alice's result with his private key. He adds his salary to what he received from Alice, encrypts the result with Carol's public key, and sends it to Carol.

(3) Carol decrypts Bob's result with her private key. She adds her salary to what she received from Bob, encrypts the result with Dave's public key, and sends it to Dave.

(4) Dave decrypts Carol's result with his private key. He adds his salary to what he received from Carol, encrypts the result with Alice's public key, and sends it to Alice.

(5) Alice decrypts Dave's result with her private key. She subtracts the random number from step (1) to recover the sum of everyone's salaries.

(6) Alice divides the result by the number of people (four, in this case) and announces the result.

This protocol assumes that everyone is honest; they may be curious, but they follow the protocol. If any participant lies about his salary, the average will be wrong. A more serious problem is that Alice can misrepresent the result to everyone. She can subtract any number she likes in step (5), and no one would be the wiser. Alice could be prevented from doing this by requiring her to commit to her random number using any of the bit-commitment schemes from Section 4.9, but when she revealed her random number at the end of the protocol Bob could learn her salary.

Protocol #2

Alice and Bob are at a restaurant together, having an argument over who is older. They don't, however, want to tell the other their age. They could each whisper their age into the ear of a trusted neutral party (the waiter, for example), who could compare the numbers in his head and announce the result to both Alice and Bob.

The above protocol has two problems. One, your average waiter doesn't have the computational ability to handle situations more complex than determining which of two numbers is greater. And two, if Alice and Bob were really concerned about the secrecy of their information, they would be forced to drown the waiter in a bowl of vichyssoise, lest he tell the wine steward.

Public-key cryptography offers a far less violent solution. There is a protocol by which Alice, who knows a value a, and Bob, who knows a value b, can together determine if $a < b$, so that Alice gets no additional information about b and Bob gets no additional information about a. And, both Alice and Bob are convinced of the validity of the computation. Since the cryptographic algorithm used is an essential part of the protocol, details can be found in Section 23.14.

Of course, this protocol doesn't protect against active cheaters. There's nothing to stop Alice (or Bob, for that matter) from lying about her age. If Bob were a computer program that blindly executed the protocol, Alice could learn his age (is the age of a computer program the length of time since it was written or the length of time since it started running?) by repeatedly executing the protocol. Alice might give her age as 60. After learning that she is older, she could execute the protocol again with her age as 30. After learning that Bob is older, she could execute the protocol again with her age as 45, and so on, until Alice discovers Bob's age to any degree of accuracy she wishes.

Assuming that the participants don't actively cheat, it is easy to extend this protocol to multiple participants. Any number of people can find out the order of their ages by a sequence of honest applications of the protocol; and no participant can learn the age of another.

Protocol #3

Alice likes to do kinky things with teddy bears. Bob has erotic fantasies about marble tables. Both are pretty embarrassed by their particular fetish, but would love to find a mate who shared in their . . . um . . . lifestyle.

Here at the Secure Multiparty Computation Dating Service, we've designed a protocol for people like them. We've numbered an astonishing list of fetishes, from "aardvarks" to "zoot suits." Discreetly separated by a modem link, Alice and Bob can participate in a secure multiparty protocol. Together, they can determine whether they share the same fetish. If they do, they might look forward to a lifetime of bliss together. If they don't, they can part company secure in the knowledge that their particular fetish remains confidential. No one, not even the Secure Multiparty Computation Dating Service, will ever know.

Here's how it works:

(1) Using a one-way function, Alice hashes her fetish into a seven-digit string.
(2) Alice uses the seven-digit string as a telephone number, calls the number,

and leaves a message for Bob. If no one answers or the number is not in service, Alice applies a one-way function to the telephone number until she finds someone who can play along with the protocol.

(3) Alice tells Bob how many times she had to apply the one-way hash function to her fetish.

(4) Bob hashes his fetish the same number of times that Alice did. He also uses the seven-digit string as a telephone number, and asks the person at the other end whether there were any messages for him.

Note that Bob has a chosen-plaintext attack. He can hash common fetishes and call the resulting telephone numbers, looking for messages for him. This protocol only really works if there are enough possible plaintext messages for this to be impractical.

There's also a mathematical protocol, one similar to Protocol #2. Alice knows *a*, Bob knows *b*, and together they will determine whether $a = b$, such that Bob does not learn anything additional about *a* and Alice does not learn anything additional about *b*. Details are in Section 23.14.

Protocol #4

This is another problem for secure multiparty computation [1373]: A council of seven meets regularly to cast secret ballots on certain issues. (All right, they rule the world—don't tell anyone I told you.) All council members can vote yes or no. In addition, two parties have the option of casting "super votes": *S*-yes and *S*-no. They do not have to cast super votes; they can cast regular votes if they prefer. If no one casts any super votes, then the majority of votes decides the issue. In the case of a single or two equivalent super votes, all regular votes are ignored. In the case of two contradicting super votes, the majority of regular votes decides. We want a protocol that securely performs this style of voting.

Two examples should illustrate the voting process. Assume there are five regular voters, N_1 through N_5, and two super voters: S_1 and S_2. Here's the vote on issue #1:

S_1	S_2	N_1	N_2	N_3	N_4	N_5
S-yes	no	no	no	no	yes	yes

In this instance the only vote that matters is S_1's, and the result is "yes."

Here is the vote on issue #2:

S_1	S_2	N_1	N_2	N_3	N_4	N_5
S-yes	S-no	no	no	no	yes	yes

Here the two super votes cancel and the majority of regular "no" votes decide the issue.

If it isn't important to hide the knowledge of whether the super vote or the regular vote was the deciding vote, this is an easy application of a secure voting protocol. Hiding that knowledge requires a more complicated secure multiparty computation protocol.

This kind of voting could occur in real life. It could be part of a corporation's organizational structure, where certain people have more power than others, or it could be part of the United Nations's procedures, where certain nations have more power than others.

Multiparty Unconditionally Secure Protocols

This is just a simple case of a general theorem: Any function of n inputs can be computed by a set of n players in a way that will let all learn the value of the function, but any set of less than $n/2$ players will not get any additional information that does not follow from their own inputs and the value of the output information. For details, see [136,334,1288,621].

Secure Circuit Evaluation

Alice has her input, a. Bob has his input, b. Together they wish to compute some general function, $f(a,b)$, such that Alice learns nothing about Bob's input and Bob learns nothing about Alice's input. The general problem of secure multiparty computation is also called **secure circuit evaluation**. Here, Alice and Bob can create an arbitrary Boolean circuit. This circuit accepts inputs from Alice and from Bob and produces an output. Secure circuit evaluation is a protocol that accomplishes three things:

1. Alice can enter her input without Bob's being able to learn it.
2. Bob can enter his input without Alice's being able to learn it.
3. Both Alice and Bob can calculate the output, with both parties being sure the output is correct and that neither party has tampered with it.

Details on secure circuit evaluation can be found in [831].

6.3 Anonymous Message Broadcast

You can't go out to dinner with a bunch of cryptographers without raising a ruckus. In [321], David Chaum introduced the Dining Cryptographers Problem:

> Three cryptographers are sitting down to dinner at their favorite three-star restaurant. Their waiter informs them that arrangements have been made with the maître d'hôtel for the bill to be paid anonymously. One of the cryptographers might be paying for the dinner, or it might have been the NSA. The three cryptographers respect each other's right to make an anonymous payment, but they wonder if the NSA is paying.

How do the cryptographers, named Alice, Bob, and Carol, determine if one of them is paying for dinner, while at the same time preserving the anonymity of the payer?

Chaum goes on to solve the problem:

> Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer to his right, so that only the two of them can see the outcome. Each

cryptographer then states aloud whether the two coins he can see—the one he flipped and the one his left-hand neighbor flipped—fell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number of differences indicates that NSA is paying (assuming that the dinner was paid for only once). Yet, if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is.

To see that this works, imagine Alice trying to figure out which other cryptographer paid for dinner (assuming that neither she nor the NSA paid). If she sees two different coins, then either both of the other cryptographers, Bob and Carol, said, "same" or both said, "different." (Remember, an odd number of cryptographers saying "different" indicates that one of them paid.) If both said, "different," then the payer is the cryptographer closest to the coin that is the same as the hidden coin (the one that Bob and Carol flipped). If both said, "same," then the payer is the cryptographer closest to the coin that is different from the hidden coin. However, if Alice sees two coins that are the same, then either Bob said, "same" and Carol said, "different," or Bob said, "different" and Carol said, "same." If the hidden coin is the same as the two coins she sees, then the cryptographer who said, "different" is the payer. If the hidden coin is different from the two coins she sees, then the cryptographer who said, "same" is the payer. In all of these cases, Alice needs to know the result of the coin flipped between Bob and Carol to determine which of them paid.

This protocol can be generalized to any number of cryptographers; they all sit in a ring and flip coins among them. Even two cryptographers can perform the protocol. Of course, they know who paid, but someone watching the protocol could tell only if one of the two paid or if the NSA paid; they could not tell which cryptographer paid.

The applications of this protocol go far beyond sitting around the dinner table. This is an example of **unconditional sender and recipient untraceability**. A group of users on a network can use this protocol to send anonymous messages.

(1) The users arrange themselves into a circle.
(2) At regular intervals, adjacent pairs of users flip coins between them, using some fair coin flip protocol secure from eavesdroppers.
(3) After every flip, each user announces either "same" or "different."

If Alice wishes to broadcast a message, she simply starts inverting her statement in those rounds corresponding to a 1 in the binary representation of her message. For example, if her message were "1001," she would invert her statement, tell the truth, tell the truth, and then invert her statement. Assuming the result of her flips were "different," "same," "same," "same," she would say "same," "same," "same," "different."

If Alice notices that the overall outcome of the protocol doesn't match the message she is trying to send, she knows that someone else is trying to send a message at the same time. She then stops sending the message and waits some random number of rounds before trying again. The exact parameters have to be worked out based on the amount of message traffic on this network, but the idea should be clear.

To make things even more interesting, these messages can be encrypted in another user's public keys. Then, when everyone receives the message (a real implementation of this should add some kind of standard message-beginning and message-ending strings), only the intended recipient can decrypt and read it. No one else knows who sent it. No one else knows who could read it. Traffic analysis, which traces and compiles patterns of people's communications even though the messages themselves may be encrypted, is useless.

An alternative to flipping coins between adjacent parties would be for them to keep a common file of random bits. Maybe they could keep them on a CD-ROM, or one member of the pair could generate a pile of them and send them to the other party (encrypted, of course). Alternatively, they could agree on a cryptographically secure pseudo-random-number generator between them, and they could each generate the same string of pseudo-random bits for the protocol.

One problem with this protocol is that while a malicious participant cannot read any messages, he can disrupt the system unobserved by lying in step (3). There is a modification to the previous protocol that detects disruption [1578,1242]; the problem is called "The Dining Cryptographers in the Disco."

6.4 DIGITAL CASH

Cash is a problem. It's annoying to carry, it spreads germs, and people can steal it from you. Checks and credit cards have reduced the amount of physical cash flowing through society, but the complete elimination of cash is virtually impossible. It'll never happen; drug dealers and politicians would never stand for it. Checks and credit cards have an audit trail; you can't hide to whom you gave money.

On the other hand, checks and credit cards allow people to invade your privacy to a degree never before imagined. You might never stand for the police following you your entire life, but the police can watch your financial transactions. They can see where you buy your gas, where you buy your food, who you call on the telephone—all without leaving their computer terminals. People need a way to protect their anonymity in order to protect their privacy.

Happily, there is a complicated protocol that allows for authenticated but untraceable messages. Lobbyist Alice can transfer **digital cash** to Congresscritter Bob so that newspaper reporter Eve does not know Alice's identity. Bob can then deposit that electronic money into his bank account, even though the bank has no idea who Alice is. But if Alice tries to buy cocaine with the same piece of digital cash she used to bribe Bob, she will be detected by the bank. And if Bob tries to deposit the same piece of digital cash into two different accounts, he will be detected—but Alice will remain anonymous. Sometimes this is called **anonymous digital cash** to differentiate it from digital money with an audit trail, such as credit cards.

A great social need exists for this kind of thing. With the growing use of the Internet for commercial transactions, there is more call for network-based privacy and anonymity in business. (There are good reasons people are reluctant to send their credit card numbers over the Internet.) On the other hand, banks and governments seem unwilling to give up the control that the current banking system's audit trail provides. They'll have to, though. All it will take for digital cash to catch on is for some trustworthy institution to be willing to convert the digits to real money.

Digital cash protocols are very complex. We'll build up to one, a step at a time. For more formal details, read [318,339,325,335,340]. Realize that this is just one digital cash protocol; there are others.

Protocol #1

The first few protocols are physical analogies of cryptographic protocols. This first protocol is a simplified physical protocol for anonymous money orders:

(1) Alice prepares 100 anonymous money orders for $1000 each.

(2) Alice puts one each, and a piece of carbon paper, into 100 different envelopes. She gives them all to the bank.

(3) The bank opens 99 envelopes and confirms that each is a money order for $1000.

(4) The bank signs the one remaining unopened envelope. The signature goes through the carbon paper to the money order. The bank hands the unopened envelope back to Alice, and deducts $1000 from her account.

(5) Alice opens the envelope and spends the money order with a merchant.

(6) The merchant checks for the bank's signature to make sure the money order is legitimate.

(7) The merchant takes the money order to the bank.

(8) The bank verifies its signature and credits $1000 to the merchant's account.

This protocol works. The bank never sees the money order it signed, so when the merchant brings it to the bank, the bank has no idea that it was Alice's. The bank is convinced that it is valid, though, because of the signature. The bank is confident that the unopened money order is for $1000 (and not for $100,000 or $100,000,000) because of the cut-and-choose protocol (see Section 5.1). It verifies the other 99 envelopes, so Alice has only a 1 percent chance of cheating the bank. Of course, the bank will make the penalty for cheating great enough so that it isn't worth that chance. If the bank refuses to sign the last check (if Alice is caught cheating) without penalizing Alice, she will continue to try until she gets lucky. Prison terms are a better deterrent.

Protocol #2

The previous protocol prevents Alice from writing a money order for more than she claims to, but it doesn't prevent Alice from photocopying the money order and spending it twice. This is called the **double spending problem**; to solve it, we need a complication:

(1) Alice prepares 100 anonymous money orders for $1000 each. On each money order she includes a different random uniqueness string, one long enough to make the chance of another person also using it negligible.
(2) Alice puts one each, and a piece of carbon paper, into 100 different envelopes. She gives them all to the bank.
(3) The bank opens 99 envelopes and confirms that each is a money order for $1000.
(4) The bank signs the one remaining unopened envelope. The signature goes through the carbon paper to the money order. The bank hands the unopened envelope back to Alice and deducts $1000 from her account.
(5) Alice opens the envelope and spends the money order with a merchant.
(6) The merchant checks for the bank's signature to make sure the money order is legitimate.
(7) The merchant takes the money order to the bank.
(8) The bank verifies its signature and checks its database to make sure a money order with the same uniqueness string has not been previously deposited. If it hasn't, the bank credits $1000 to the merchant's account. The bank records the uniqueness string in a database.
(9) If it has been previously deposited, the bank doesn't accept the money order.

Now, if Alice tries to spend a photocopy of the money order, or if the merchant tries to deposit a photocopy of the money order, the bank will know about it.

Protocol #3

The previous protocol protects the bank from cheaters, but it doesn't identify them. The bank doesn't know if the person who bought the money order (the bank has no idea it's Alice) tried to cheat the merchant or if the merchant tried to cheat the bank. This protocol corrects that:

(1) Alice prepares 100 anonymous money orders for $1000 each. On each of the money orders she includes a different random uniqueness string, one long enough to make the chance of another person also using it negligible.
(2) Alice puts one each, and a piece of carbon paper, into 100 different envelopes. She gives them all to the bank.
(3) The bank opens 99 envelopes and confirms that each is a money order for $1000 and that all the random strings are different.
(4) The bank signs the one remaining unopened envelope. The signature goes through the carbon paper to the money order. The bank hands the unopened envelope back to Alice and deducts $1000 from her account.
(5) Alice opens the envelope and spends the money order with a merchant.
(6) The merchant checks for the bank's signature to make sure the money order is legitimate.

(7) The merchant asks Alice to write a random identity string on the money order.

(8) Alice complies.

(9) The merchant takes the money order to the bank.

(10) The bank verifies the signature and checks its database to make sure a money order with the same uniqueness string has not been previously deposited. If it hasn't, the bank credits $1000 to the merchant's account. The bank records the uniqueness string and the identity string in a database.

(11) If the uniqueness string is in the database, the bank refuses to accept the money order. Then, it compares the identity string on the money order with the one stored in the database. If it is the same, the bank knows that the merchant photocopied the money order. If it is different, the bank knows that the person who bought the money order photocopied it.

This protocol assumes that the merchant cannot change the identity string once Alice writes it on the money order. The money order might have a series of little squares, which the merchant would require Alice to fill in with either Xs or Os. The money order might be made out of paper that tears if erased.

Since the interaction between the merchant and the bank takes place after Alice spends the money, the merchant could be stuck with a bad money order. Practical implementations of this protocol might require Alice to wait near the cash register during the merchant-bank interaction, much the same way as credit-card purchases are handled today.

Alice could also frame the merchant. She could spend a copy of the money order a second time, giving the same identity string in step (7). Unless the merchant keeps a database of money orders it already received, he would be fooled. The next protocol eliminates that problem.

Protocol #4

If it turns out that the person who bought the money order tried to cheat the merchant, the bank would want to know who that person was. To do that requires moving away from a physical analogy and into the world of cryptography.

The technique of secret splitting can be used to hide Alice's name in the digital money order.

(1) Alice prepares n anonymous money orders for a given amount.

Each of the money orders contains a different random uniqueness string, X, one long enough to make the chance of two being identical negligible.

On each money order, there are also n pairs of identity bit strings, I_1, I_2, . . . , I_n. (Yes, that's n different pairs on *each* check.) Each of these pairs is generated as follows: Alice creates a string that gives her name, address, and any other piece of identifying information that the bank wants to see. Then, she splits it into two pieces using the secret splitting protocol (see Section 3.6). Then, she commits to each piece using a bit-commitment protocol.

For example, I_{37} consists of two parts: I_{37_L} and I_{37_R}. Each part is a bit-committed packet that Alice can be asked to open and whose proper opening can be instantly verified. Any pair (e.g., I_{37_L} and I_{37_R}, but not I_{37_L} and I_{38_R}), reveals Alice's identity.

Each of the money orders looks like this:

```
Amount
Uniqueness String: X
Identity Strings:  I_1 = (I_1L, I_1R)
                   I_2 = (I_2L, I_2R)
                   ....
                   I_n = (I_nL, I_nR)
```

(2) Alice blinds all n money orders, using a blind signature protocol. She gives them all to the bank.

(3) The bank asks Alice to unblind $n - 1$ of the money orders at random and confirms that they are all well formed. The bank checks the amount, the uniqueness string, and asks Alice to reveal all of the identity strings.

(4) If the bank is satisfied that Alice did not make any attempts to cheat, it signs the one remaining blinded money order. The bank hands the blinded money order back to Alice and deducts the amount from her account.

(5) Alice unblinds the money order and spends it with a merchant.

(6) The merchant verifies the bank's signature to make sure the money order is legitimate.

(7) The merchant asks Alice to randomly reveal either the left half or the right half of each identity string on the money order. In effect, the merchant gives Alice a random n-bit **selector string**, $b_1, b_2, \ldots, b_n$. Alice opens either the left or right half of I_i, depending on whether b_i is a 0 or a 1.

(8) Alice complies.

(9) The merchant takes the money order to the bank.

(10) The bank verifies the signature and checks its database to make sure a money order with the same uniqueness string has not been previously deposited. If it hasn't, the bank credits the amount to the merchant's account. The bank records the uniqueness string and all of the identity information in a database.

(11) If the uniqueness string is in the database, the bank refuses to accept the money order. Then, it compares the identity string on the money order with the one stored in the database. If it is the same, the bank knows that the merchant copied the money order. If it is different, the bank knows that the person who bought the money order photocopied it. Since the second merchant who accepted the money order handed Alice a different selector string than did the first merchant, the bank finds a bit position where one merchant had Alice open the left half and the other merchant had Alice open the right half. The bank XORs the two halves together to reveal Alice's identity.

This is quite an amazing protocol, so let's look at it from various angles.

Can Alice cheat? Her digital money order is nothing more than a string of bits, so she can copy it. Spending it the first time won't be a problem; she'll just complete the protocol and everything will go smoothly. The merchant will give her a random n-bit selector string in step (7) and Alice will open either the left half or right half of each I_i in step (8). In step (10), the bank will record all of this data, as well as the money order's uniqueness string.

When she tries to use the same digital money order a second time, the merchant (either the same merchant or a different merchant) will give her a different random selector string in step (7). Alice must comply in step (8); not doing so will immediately alert the merchant that something is suspicious. Now, when the merchant brings the money order to the bank in step (10), the bank would immediately notice that a money order with the same uniqueness string was already deposited. The bank then compares the opened halves of the identity strings. The odds that the two random selector strings are the same is 1 in 2^n; it isn't likely to happen before the next ice age. Now, the bank finds a pair with one half opened the first time and the other half opened the second time. It XORs the two halves together, and out pops Alice's name. The bank knows who tried to spend the money order twice.

Note that this protocol doesn't keep Alice from trying to cheat; it detects her cheating with almost certainty. Alice can't prevent her identity from being revealed if she cheats. She can't change either the uniqueness string or any of the identity strings, because then the bank's signature will no longer be valid. The merchant will immediately notice that in step (6).

Alice could try to sneak a bad money order past the bank, one on which the identity strings don't reveal her name; or better yet, one whose identity strings reveal someone else's name. The odds of her getting this ruse past the bank in step (3) are 1 in n. These aren't impossible odds, but if you make the penalty severe enough, Alice won't try it. Or, you could increase the number of redundant money orders that Alice makes in step (1).

Can the merchant cheat? His chances are even worse. He can't deposit the money order twice; the bank will notice the repeated use of the selector string. He can't fake blaming Alice; only she can open any of the identity strings.

Even collusion between Alice and the merchant can't cheat the bank. As long as the bank signs the money order with the uniqueness string, the bank is assured of only having to make good on the money order once.

What about the bank? Can it figure out that the money order it accepted from the merchant was the one it signed for Alice? Alice is protected by the blind signature protocol in steps (2) through (5). The bank cannot make the connection, even if it keeps complete records of every transaction. Even more strongly, there is no way for the bank and the merchant to get together to figure out who Alice is. Alice can walk in the store and, completely anonymously, make her purchase.

Eve can cheat. If she can eavesdrop on the communication between Alice and the merchant, and if she can get to the bank before the merchant does, she can deposit the digital cash first. The bank will accept it and, even worse, when the merchant tries to deposit the cash he will be identified as a cheater. If Eve steals and spends

Alice's cash before Alice can, then Alice will be identified as a cheater. There's no way to prevent this; it is a direct result of the anonymity of the cash. Both Alice and the merchant have to protect their bits as they would paper money.

This protocol lies somewhere between an arbitrated protocol and a self-enforcing protocol. Both Alice and the merchant trust the bank to make good on the money orders, but Alice does not have to trust the bank with knowledge of her purchases.

Digital Cash and the Perfect Crime

Digital cash has its dark side, too. Sometimes people don't want so much privacy. Watch Alice commit the perfect crime [1575]:

(1) Alice kidnaps a baby.
(2) Alice prepares 10,000 anonymous money orders for $1000 (or as many as she wants for whatever denomination she wants).
(3) Alice blinds all 10,000 money orders, using a blind signature protocol. She sends them to the authorities with the threat to kill the baby unless the following instructions are met:
    (a) Have a bank sign all 10,000 money orders.
    (b) Publish the results in a newspaper.
(4) The authorities comply.
(5) Alice buys a newspaper, unblinds the money orders, and starts spending them. There is no way for the authorities to trace the money orders to her.
(6) Alice frees the baby.

Note that this situation is much worse than any involving physical tokens—cash, for example. Without physical contact, the police have less opportunity to apprehend the kidnapper.

In general, though, digital cash isn't a good deal for criminals. The problem is that the anonymity only works one way: The spender is anonymous, but the merchant is not. Moreover, the merchant cannot hide the fact that he received money. Digital cash will make it easy for the government to determine how much money you made, but impossible to determine what you spent it on.

Practical Digital Cash

A Dutch company, DigiCash, owns most of the digital cash patents and has implemented digital cash protocols in working products. Anyone interested should contact DigiCash BV, Kruislaan 419, 1098 VA Amsterdam, Netherlands.

Other Digital Cash Protocols

There are other digital cash protocols; see [707,1554,734,1633,973]. Some of them involve some pretty complicated mathematics. Generally, the various digital cash protocols can be divided into various categories. **On-line** systems require the merchant to communicate with the bank at every sale, much like today's credit-card protocols. If there is a problem, the bank doesn't accept the cash and Alice cannot cheat.

Off-line systems, like Protocol #4, require no communication between the merchant and the bank until after the transaction between the merchant and the customer. These systems do not prevent Alice from cheating, but instead detect her cheating. Protocol #4 detected her cheating by making Alice's identity known if she tried to cheat. Alice knows that this will happen, so she doesn't cheat.

Another way is to create a special smart card (see Section 24.13) containing a tamperproof chip called an **observer** [332,341,387]. The observer chip keeps a mini database of all the pieces of digital cash spent by that smart card. If Alice attempts to copy some digital cash and spend it twice, the imbedded observer chip would detect the attempt and would not allow the transaction. Since the observer chip is tamperproof, Alice cannot erase the mini-database without permanently damaging the smart card. The cash can wend its way through the economy; when it is finally deposited, the bank can examine the cash and determine who, if anyone, cheated.

Digital cash protocols can also be divided along another line. **Electronic coins** have a fixed value; people using this system will need several coins in different denominations. **Electronic checks** can be used for any amount up to a maximum value and then returned for a refund of the unspent portion.

Two excellent and completely different off-line electronic coin protocols are [225,226,227] and [563,564,565]. A system called NetCash, with weaker anonymity properties, has also been proposed [1048,1049]. Another new system is [289].

In [1211], Tatsuaki Okamoto and Kazuo Ohta list six properties of an ideal digital cash system:

1. Independence. The security of the digital cash is not dependent on any physical location. The cash can be transferred through computer networks.
2. Security. The digital cash cannot be copied and reused.
3. Privacy (Untraceability). The privacy of the user is protected; no one can trace the relationship between the user and his purchases.
4. Off-line Payment. When a user pays for a purchase with electronic cash, the protocol between the user and the merchant is executed off-line. That is, the shop does not need to be linked to a host to process the user's payment.
5. Transferability. The digital cash can be transferred to other users.
6. Divisibility. A piece of digital cash in a given amount can be subdivided into smaller pieces of cash in smaller amounts. (Of course, everything has to total up properly in the end.)

The protocols previously discussed satisfy properties 1, 2, 3, and 4, but not 5 and 6. Some on-line digital cash systems satisfy all properties except 4 [318,413, 1243]. The first off-line digital cash system that satisfies properties 1, 2, 3, and 4, similar to the one just discussed, was proposed in [339]. Okamoto and Ohta proposed a system that satisfies properties 1 through 5 [1209]; they also proposed a system that satisfies properties 1 through 6 as well, but the data requirement for a single purchase is approximately 200 megabytes. Another off-line divisible coin system is described in [522].

The digital cash scheme proposed in [1211], by the same authors, satisfies properties 1 through 6, without the enormous data requirements. The total data transfer for a payment is about 20 kilobytes, and the protocol can be completed in several seconds. The authors consider this the first ideal untraceable electronic cash system.

Anonymous Credit Cards

This protocol [988] uses several different banks to protect the identity of the customer. Each customer has an account at two different banks. The first bank knows the person's identity and is willing to extend him credit. The second bank knows the customer only under a pseudonym (similar to a numbered Swiss bank account).

The customer can withdraw funds from the second bank by proving that the account is his. However, the bank does not know the person and is unwilling to extend him credit. The first bank knows the customer and transfers funds to the second bank—without knowing the pseudonym. The customer then spends these funds anonymously. At the end of the month, the second bank gives the first bank a bill, which it trusts the bank to pay. The first bank passes the bill on to the customer, which it trusts the customer to pay. When the customer pays, the first bank transfers additional funds to the second bank. All transactions are handled through an intermediary, which acts sort of like an electronic Federal Reserve: settling accounts among banks, logging messages, and creating an audit trail.

Exchanges between the customer, merchant, and various banks are outlined in [988]. Unless everyone colludes against the customer, his anonymity is assured. However, this is not digital cash; it is easy for the bank to cheat. The protocol allows customers to keep the advantages of credit cards without giving up their privacy.