How to Stop a Cheater: Secret Sharing with Dishonest Participation

Erik Warren Selberg

July 22, 1993
CMU-CS-93-182

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This report was submitted in partial fulfillment of the requirements for the Senior Honors Research Program in the School of Computer Science at Carnegie Mellon University. Erik Selberg was the 1993 recipient of the Allen Newell Award for Excellence in Undergraduate Research.

Keywords: cryptography, secret sharing, distributed security, simultaneous exchange, zero-knowledge proof

Abstract

At times it is necessary to obtain a group decision from a number of different nodes over a large network. Secret sharing protocols allow a quorum q of a group of n people to arrive at decisions by having the quorum recompute a predetermined secret, such as an access code, while preventing less than q people from gaining any information about the secret. However, current protocols [6, 5] are vulnerable when participants cheat, for example by giving false information to other participants. In this work, I present a powerful new protocol which detects cheaters immediately and halts the exchange before any more information is revealed. In addition, it prevents cheaters from gaining any information without revealing an equal amount of their own. This protocol will present new paradigms in a variety of applications, such as electronic balloting and secure file system fault tolerance.

1 Introduction and Motivation

> In his will, a man left a safe to his eleven squabbling relatives. To each individual heir, he gave a secret. In his will, he said that if eight or more of his eleven relatives could put aside their differences and pool their resources, they could compute the safe combination from their individual secrets. Otherwise, the safe would remain locked and no one would inherit anything.

In order to execute the will in this parable, a special protocol is required to allow a group of people to share a secret such that a quorum of them can recompute the secret, yet less than that quorum can gain no information. In addition, such a protocol requires the prevention of cheating so that participants are unable to extract more information than they reveal.

A protocol of this nature is called a (q, n) *threshold scheme* or *secret sharing problem*. A secret sharing problem in general has the following characteristics:

1. a secret S is divided into n pieces, or *shares*;
2. knowledge of quorum q or more shares allows S to be easily computed;
3. knowledge of $q-1$ or less pieces reveals no information about S.

In addition, it must be assured that participants in the protocol are unable to cheat. The secret sharing problem implicitly assumes that shares will be pooled by the quorum to compute the secret. Various protocols have been created [5, 8, 1]. Some of these include techniques to catch cheaters who might put bogus values into the shared pool of information. Even if cheaters can be detected, a problem remains since cheaters can still see the contents of the pool before revealing their own shares. A workable protocol must be able to ensure that cheaters cannot see the contents of the pool without first adding their true shares to it. This would ensure that a cheating party could not gain any advantage.

2 Description

In order to solve a distributed secret sharing problem, a (q, n) threshold scheme can be implemented. However, a few extra constraints are needed:

4. Any party involved will be unable to extract someone else's share without revealing its own.

5. Any party involved with the protocol will be detected if it cheats.

and optionally:

6. New shares can be created which can replace or supplement older shares.

We assume that the secret is bounded by a public prime p, and $q \leq n$.

A protocol which has all of these characteristics can find application in a variety of settings. For example:

- *Distributed Decision Making*[4] Let the secret be a key, for example an access code. The question is whether the access code is allowed to be used. The solution: divide up the key into n shares and give one share to each of the n people involved. Let q be the number of "yes" votes necessary to decide to use the access code. When someone decides to vote "yes," she simply adds her share into a general pool. When q shares are in the pool, the access code can be computed and used. Otherwise, the access code will remain safely anonymous.

- *Hierarchical Access*[6] Similar to distributed decision making, let the secret be a key. However, instead of giving one share to everyone, prioritize people according to authority. In a business, for example, you could give the CEO q shares–the secret in essence. For each member of the board, you could give $\frac{q}{2}$ shares. For each manager, $\frac{q}{3}$ shares, and so forth. Thus, rather than requiring q people to convene, all that is necessary is to gather people whom together have q shares.

- *Secure File Storage and Fault Tolerance*[3, 2] Let the secret be a file. Let n be the number of disks available with each disk receiving one share, and let q be the minimum number of shares needed to re-create the file. Thus, in order for an interloper to gain access to the file, he must gain access to q disks. This provides much greater security for files than would exist by simply keeping the file in one central location as it requires an interloper to gain access to each individual disk. In addition,

this method provides superior fault tolerance to a simple backup copy scheme. In order for the file to be lost, more than $n - q - 1$ disks must fail. As this method does not require an extraordinary amount of additional data over the single backup scheme, it is doubly attractive.

3 Shamir's Solution

The first secret sharing protocol was created by Adi Shamir [6]. His protocol is not protected against cheating attacks.

In Shamir's original protocol, a *Dealer*, the person who knows the secret, creates a polynomial $f(x)$ of the form:

$$f(x) = x^q + a_{q-1}x^{q-1} + \cdots + a_1 x + S \pmod{p}$$

where the coefficients $a_1 \ldots a_{q-1}$ are chosen independently from [0, p). S is the secret (an integer modulo p) and p is a large public prime.

The *Dealer* sends to each participant $C_1 \ldots C_n$ $f(i)$ (assume that the participants know their index). Hereafter, we will refer to $f(i)$ as C_i's *share* s. When q participants desire to recompute the secret, they exchange their shares with each other. We will refer to q as a *quorum*. Once they have q shares $s_1 \ldots s_q$, they can create q equations. To find S, they need only solve for S and each a_i in the following system of equations:

$$\begin{aligned}
f(s_1) &= s_1^q + a_{q-1}s_1^{q-1} + \cdots + a_1 s_1 + S \pmod{p} \\
f(s_2) &= s_2^q + a_{q-1}s_2^{q-1} + \cdots + a_1 s_2 + S \pmod{p} \\
&\vdots \\
f(s_q) &= s_q^q + a_{q-1}s_q^{q-1} + \cdots + a_1 s_q + S \pmod{p}
\end{aligned}$$

where p and each $f(s_i), s_i$ are known.

To solve for S and $a_1 \ldots a_{q-1}$, all that is required is to solve for q unknowns with q equations in a modulo field. In matrix form ($\vec{u} = M\vec{v}$, where $\vec{u}$ and M are known) this is:

$$\underbrace{\begin{bmatrix} f(s_1) \\ f(s_2) \\ \vdots \\ f(s_q) \end{bmatrix}}_{\text{known}} = \underbrace{\begin{bmatrix} s_1^q & s_1^{q-1} & \ldots & s_1 & 1 \\ s_2^q & s_2^{q-1} & \ldots & s_2 & 1 \\ & \vdots & & & \\ s_2^q & s_2^{q-1} & \ldots & s_2 & 1 \end{bmatrix}}_{\text{known}} \underbrace{\begin{bmatrix} a_{q-1} \\ \vdots \\ a_1 \\ S \end{bmatrix}}_{\textit{unknown}} \pmod{p}$$

All that is needed to solve for $\vec{v}$ (and thus S) is a simple matrix inversion modulo p and multiplication. Computation modulo p is simple to do since the integers modulo p form a field ($\mathbb{Z}/p\mathbb{Z}$).

3.1 Proof of Correctness

Theorem 1 *A participant with $q-1$ (or less) shares can gain absolutely no information about the secret S.*

Proof: Assuming that a participant does have $q-1$ shares, he can create the following:

$$\begin{aligned} f(s_1) &= s_1^q + a_{q-1}s_1^{q-1} + \cdots + a_1 s_1 + S \pmod p \\ f(s_2) &= s_2^q + a_{q-1}s_2^{q-1} + \cdots + a_1 s_2 + S \pmod p \\ &\vdots \\ f(s_{q-1}) &= s_{q-1}^q + a_{q-1}s_{q-1}^{q-1} + \cdots + a_1 s_{q-1} + S \pmod p \end{aligned}$$

To find S, he has to solve for q unknowns with only $q-1$ equations. The best he can do is create an equation for S with one degree of freedom [7], which gives no information about the actual value of S. ∎

3.2 Weaknesses

There are two major weaknesses in Shamir's protocol:

1. Bogus values are undetectable.
2. Participants need not reveal their true share.

These two weaknesses are distinct, because even if a bogus value was detected, it would not necessarily give any information about the true value. However, should some participant A give another participant B invalid information after B has already given valid information to A, even if B could detect A's bogus information A will still have more information than B. To see this, consider the following example:

Example 1 *Assume that the secret S is* 13*, and that q is* 3*. Also assume that the* Dealer *has created the polynomial:*

$$f(x) = x^3 + 11x^2 + 4x + 13 \pmod{17}$$

and has distributed the following points to Alice, Bob, and Carl:

Alice: $(1, 12)$ *Bob:* $(2, 5)$ *Carl:* $(3, 15)$

Normal interpolation would result in the following equations:

$$\begin{aligned} 12 &= 1 + a + b + S \pmod{17} \\ 5 &= 8 + 4a + 2b + S \pmod{17} \\ 15 &= 10 + 9a + 3b + S \pmod{17} \end{aligned}$$

which translates into the following matrix form:

$$\begin{bmatrix} 11 \\ 14 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ S \end{bmatrix} \pmod{17}$$

which naturally results in $[a, b, S] = [11, 4, 13]$.

However, if Alice tries to fool Bob and Carl, she can submit $(1, 8)$ *instead of her actual value. Bob and Carl will then try to solve:*

$$\begin{aligned} 8 &= 1 + a + b + S \pmod{17} \\ 5 &= 8 + 4a + 2b + S \pmod{17} \\ 15 &= 10 + 9a + 3b + S \pmod{17} \end{aligned}$$

which translated in matrix form to:

$$\begin{bmatrix} 7 \\ 14 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ S \end{bmatrix} \pmod{17}$$

which yields the incorrect result of $[a, b, S] = [9, 14, 1]$. *However, to the eyes of Bob and Carl, everything is as it should be until the secret is actually used and is found to be in error. However, Neither Bob nor Carl can determine who cheated, as Alice can easily interpolate the incorrect answer and show that to Bob and Carl, demonstrating that she also has been duped and does not know the true secret. Yet Alice now has both Bob's and Carl's shares, and can compute* S *at her leisure.*

4 Ben-Or/Rabin Solution

Tal Rabin and Michael Ben-Or improved on the protocols of Shamir and others [8, 1] by introducing a zero-knowledge proof based upon *Check Vectors*

into the protocol [5]. The improvement to Shamir's secret sharing protocol that we are concerned about is as follows:

For every participant A, B the *Dealer* picks two positive non-zero integers $b_{AB}, y_{AB} \in \mathbb{Z}_p$ and calculates

$$c_{AB} = (b_{AB})(y_{AB}) + s_A \pmod{p}$$

Then instead of just distributing a share s_A to A, the dealer gives A s_A and y_{AB} and B the pair (b_{AB}, c_{AB}). This pair is known as a *Check Vector*. It is important to note that A keeps s_A and y_{AB} secret, just as B keeps (b_{AB}, c_{AB}) secret.

When a quorum of participants wish to recompute the secret, each participant A exchanges his information privately with participant B. B then uses his check vector (b_{AB}, c_{AB}) to ensure that:

$$s_A + (b_{AB})(y_{AB}) = c_{AB} \pmod{p}$$

Thus, A cannot try to pass B a bogus value.

4.1 Proof of Correctness

We need to show two things to prove that this protocol works as stated.

Lemma 2 *The probability of A deceiving B is $\frac{1}{p-1}$.*

Proof: In order for A to deceive B, A must send a s'_A and y'_{AB} such that:

$$s'_A + (b_{AB})(y'_{AB}) = c_{AB} \pmod{p}$$

However, there is only one possible y'_{AB} that will satisfy this equation. Thus, A has a probability of $\frac{1}{1-p}$ to pick the correct value, as he has no information about (b_{AB}, c_{AB}). ■

Lemma 3 *B has no information from his check vector (b_{AB}, c_{AB}).*

Proof: As mentioned in the preceding proof, for all s there exists a unique y such that:

$$s + by = c \pmod{p}$$

The converse, for every y there exists a unique s, also holds. Thus, B has no information about s_A. ■

4.2 Weaknesses

The weaknesses of Shamir's solution were twofold: participants could reveal bogus information into the protocol and thereby prevent the secret from being recomputed, and secondly they could give invalid information after receiving another participant's valid information. The Ben-Or/Rabin protocol detects fraudulent values handily, eliminating one of the weaknesses of Shamir's protocol. Unfortunately, the other still remains, namely:

- Participants need not reveal their true shares.

As this protocol involves an exchange of information, the following situation is quite possible:

Alice and Bob decide to recompute the secret. Bob sends his partial secret to Alice, but after receiving both Bob's secret Alice decides to send either a bogus value or nothing at all. Thus, Alice can now compute the secret, but Bob cannot, nor is he able to prevent Alice from doing so.

5 Bitwise Check Vector Solution

The idea behind the *ybc* Vector Protocol is to use a bitwise variant of the Ben-Or/Rabin Check Vector solution. The problem with the Ben-Or/Rabin solution is that at some point in the protocol participant A will have B's share, but B will have yet to receive A's share. Termination of the protocol at that point will result in A knowing B's share while keeping his own share private.

This problem can be resolved by exchanging bits instead of the full numbers. Thus, at any point in the protocol, A will only be at most 1 bit ahead of B, which is not a significant advantage, since exhaustive search techniques would allow an attacker to search the space of feasible factors. A one bit advantage translates into a factor of two in search time.

5.1 The *ybc* Vector Protocol

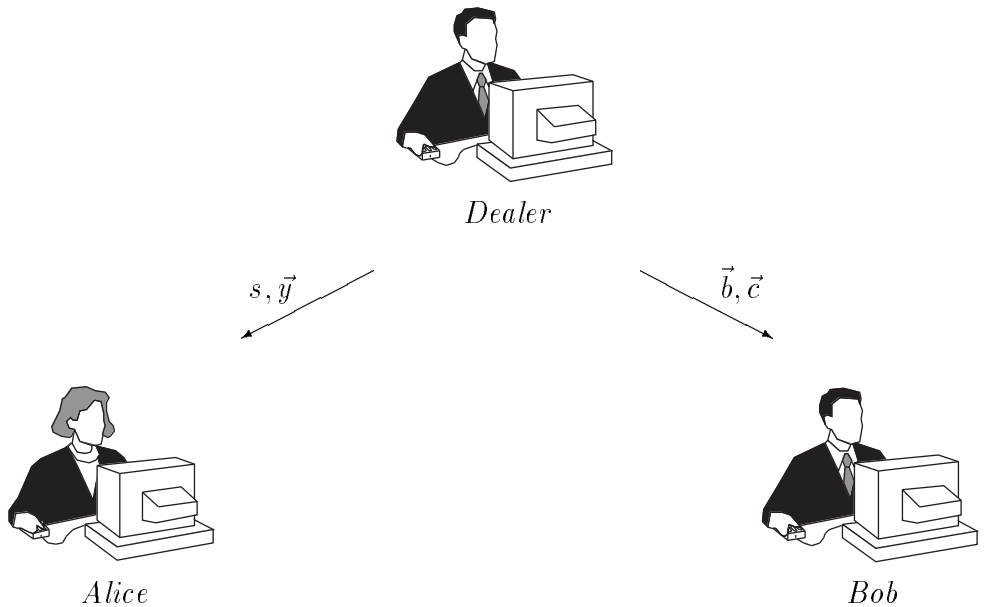
To describe the protocol, we will initially focus on the bit-by-bit exchange from Alice (A) to Bob (B). The full protocol between Alice and Bob occurs when Alice and Bob exchange bits interactively using the bit-by-bit protocol.

```
(Step 1)    The Dealer sends s and y to Alice; b,c to Bob.
            For each s_[i] do:
(Step 2a)       Alice sends s_[i] and y_i to Bob.
(Step 2b)       Bob verifies that s_[i] + b_i y_i = c_i  (mod p).
                If so:
                    Bob sends an acknowledgment.
                    Alice returns to Step 2a.
                Else:
                    Bob terminates exchange.
```

Figure 1: One-sided *ybc* Vector Protocol

Let p be a large published prime. Recall that p is the upper bound on S, the coefficients $a_{q-1} \ldots a_1$, and shares $s_1 \ldots s_n$.

The one-sided protocol works as detailed in Figure 1. The *Dealer* sends a k-bit share s and a vector $\vec{y}$ of k integers to Alice, and two vectors $\vec{b}$ and $\vec{c}$ of k integers to Bob,

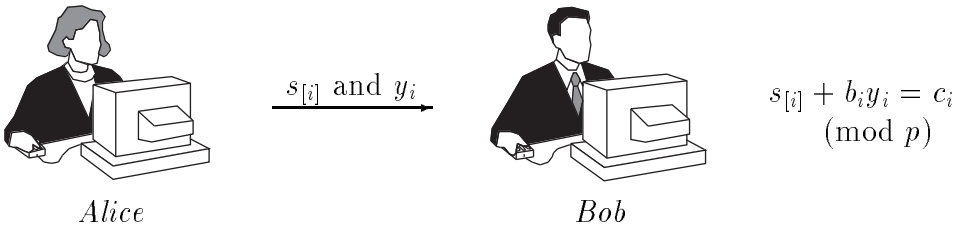


Step 1: *Dealer* distributes to *Alice* and *Bob*.

such that:

$$s_{[i]} + \vec{b}_i \vec{y}_i = \vec{c}_i \pmod{p}$$

where $\vec{b}$ and $\vec{y}$ are chosen at random, and $s_{[i]}$ denotes the i^{th} bit of s.



Step 2: *Alice* sends a bit to *Bob* who then verifies it.

At the end of this protocol, Bob will have Alice's entire share s, and will know that s is valid.

Theorem 4 *The probability of Alice deceiving Bob on any given bit is* $\frac{1}{p-1}$.

Proof: Without loss of generality, assume that Alice is going to try to fool Bob for bit $s_{[i]}$, and remain honest for the rest. To fool Bob, Alice needs to send Bob the pair $(\neg s_{[i]}, y_i')$ such that:

$$\neg s_{[i]} + b_i y_i' = c_i \pmod{p}$$

However, from Lemma 2, we know that there exists only one y_i' that will satisfy the equality. The probability of Alice picking the correct y_i' is $\frac{1}{p-1}$. ■

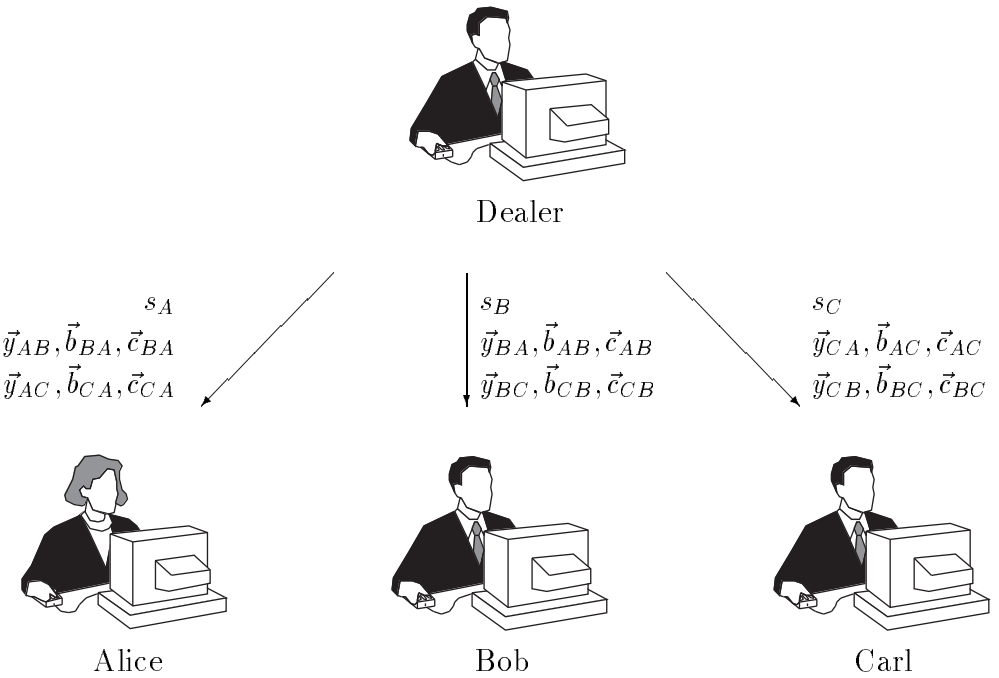
The complete protocol is simply a concurrent extension of the previous protocol between Alice and Bob. All participants privately transmit the first bit of their share along with the appropriate y_i privately to each other. They then verify all the bits they receive from the other participants, and continue to the next bit. This is formalized in Figure 2.

Example 2 *Consider the full protocol using three participants, Alice, Bob, and Carl. The* Dealer *begins by creating the polynomial and subsequently creating and sending the appropriate shares and* $\vec{y}, \vec{b}$, *and* $\vec{c}$ *to the participants,*

(Step 1) The *Dealer* distributes s_{C_j} and the $\vec{y}_{C_jC_l}, \vec{b}_{C_lC_j}$ and $\vec{c}_{C_lC_j}$ vectors.
For i = 1 to k (where k is the number of bits in s) do:
(Step 2) Each C_j privately sends $s_{C_j[i]}$ and $\vec{y}_{C_jC_li}$ to C_l
(Step 3) Each C_l verifies $s_{C_j[i]} + (\vec{b}_{C_jC_li})(\vec{y}_{C_jC_li}) = \vec{c}_{C_jC_li} \pmod{p}$.
If any C_l detects cheating by some C_m:
C_l terminates the exchange with C_m.
C_l notifies the other participants of C_m's attempt.
Else:
The protocol progresses to the next i.
(Step 4) Each C_j solves for S.

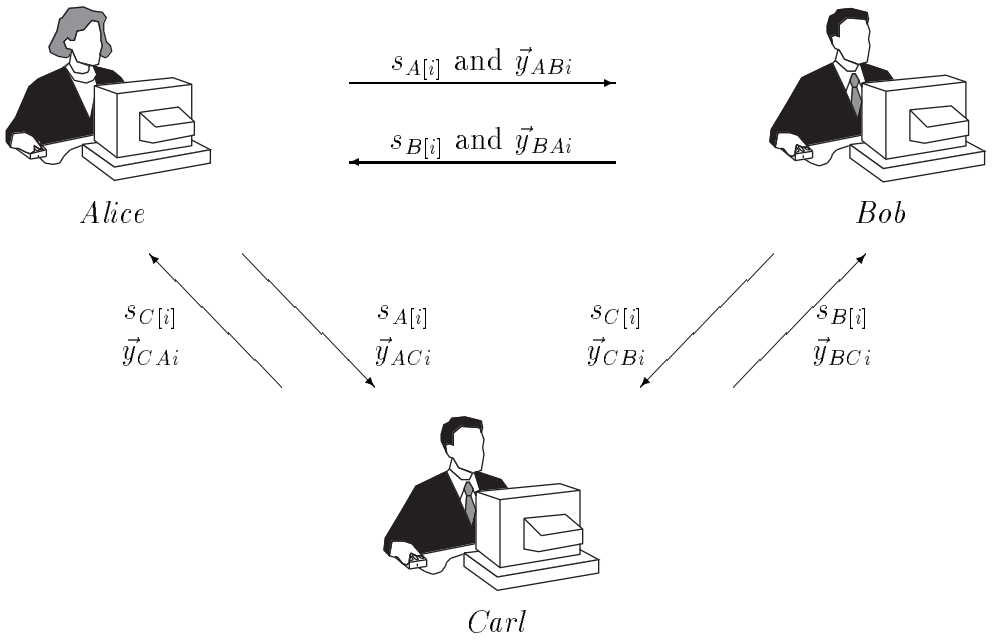
Figure 2: Full *ybc* Vector Protocol

as detailed in Step 1. For this example, assume that the polynomial is of degree 3*, which requires all three participants to reveal their shares to recompute the secret.*



Step 1: *Dealer* distributes to *Alice*, *Bob*, and *Carl*.

When all the participants decide to recompute the secret, they will privately exchange information bit-by-bit, as shown in Step 2.



Step 2: *Alice, Bob,* and *Carl* exchange shares.

After each bit is exchanged, Alice, Bob, and Carl will verify the validity of the bit using the appropriate check vectors as shown in Step 3. Again, the calculation is done after each *bit is exchanged and* not *after all bits have been exchanged.*

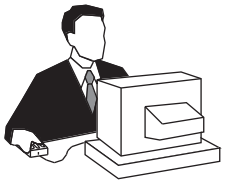
Alice	Bob	Carl
$s_{B[i]} + \vec{b}_{BAi}\vec{y}_{BAi} \stackrel{?}{=} \vec{c}_{BAi}$	$s_{A[i]} + \vec{b}_{ABi}\vec{y}_{ABi} \stackrel{?}{=} \vec{c}_{ABi}$	$s_{A[i]} + \vec{b}_{ACi}\vec{y}_{ACi} \stackrel{?}{=} \vec{c}_{ACi}$
$s_{C[i]} + \vec{b}_{CAi}\vec{y}_{CAi} \stackrel{?}{=} \vec{c}_{CAi}$	$s_{C[i]} + \vec{b}_{CBi}\vec{y}_{CBi} \stackrel{?}{=} \vec{c}_{CBi}$	$s_{B[i]} + \vec{b}_{BCi}\vec{y}_{BCi} \stackrel{?}{=} \vec{c}_{BCi}$
(mod p)	(mod p)	(mod p)

Step 3: Alice, Bob, *and* Carl *verify each others' bits.*

Once all three participants have received the other shares, they each create the appropriate equations and solve, as shown in Step 4.

Alice

$$\begin{bmatrix} s_A \\ s_B \\ s_C \end{bmatrix} = \begin{bmatrix} s_A^3 & s_A^2 & s_A & 1 \\ s_B^3 & s_B^2 & s_B & 1 \\ s_C^3 & s_C^2 & s_C & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ S \end{bmatrix} \pmod{p}$$



Bob

$$\begin{bmatrix} s_A \\ s_B \\ s_C \end{bmatrix} = \begin{bmatrix} s_A^3 & s_A^2 & s_A & 1 \\ s_B^3 & s_B^2 & s_B & 1 \\ s_C^3 & s_C^2 & s_C & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ S \end{bmatrix} \pmod{p}$$

Carl

$$\begin{bmatrix} s_A \\ s_B \\ s_C \end{bmatrix} = \begin{bmatrix} s_A^3 & s_A^2 & s_A & 1 \\ s_B^3 & s_B^2 & s_B & 1 \\ s_C^3 & s_C^2 & s_C & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ S \end{bmatrix} \pmod{p}$$

Step 4: Alice, Bob, *and* Carl *solve for* S.

The secret now being known, they are able to use it as they see fit.

5.2 Space Requirements

For each X, Y, where X and Y are distinct parties holding shares, the *Dealer* needs to create the following:

$\vec{y}_{XY}$ The $\vec{y}$ vector which X will use to exchange his share with Y.

$\vec{b}_{XY}$ The $\vec{b}$ component which Y will use to verify X's share.

$\vec{c}_{XY}$ The $\vec{c}$ component which Y will use to verify X's share.

Each participant C_i has three sets of vectors, each set of length $n-1$. Thus, each C_i requires additional space of $O(3(n-1))$. As there are n participants, this expands to $O(3(n-1)n)$ or simply $O(n^2)$ extra space for the entire protocol.

Note that each $\vec{y}_{XY}, \vec{b}_{XY}$, and $\vec{c}_{XY}$ must be different for each X and Y X; otherwise, two participants could collude and thus determine every $\vec{y}, \vec{b}$, and $\vec{c}$, which would enable them to cheat without detection.

6 Conclusions and Future Research

Secret sharing is unusable unless it can ensure that the people involved with the protocol are unable to cheat. The *ybc* vector protocol ensures that any person involved with the protocol is unable to gain any information from another person without revealing an equal amount of his own information (within a constant factor of 2). However, this protocol does have the following limitations:

1. A cheater can gain an advantage of one bit. Should a cheater decide to take this advantage during the middle of a transaction and then attempt to exhaustively search for the remaining bits, then he will have an advantage of a factor of 2 on the other person. If both parties have equivalent computational power, then this is not an advantage. However, this could be a consideration if the cheater has substantially more computational power (enough so that he can compute the secret by brute force before the victim is able to do something about it).

2. The memory requirements of this protocol are $O(n^2)$. For a reasonable n, this is not necessarily burdensome; however, for very large n, this can be a problem. For example, this protocol could be used with ease to implement electronic balloting with a constituent of 20, but would be clearly impractical for a popular vote with a constituency of the United States.

3. This protocol only allows for a one-time use of a secret. Once the secret has been revealed, all outstanding keys become useless, and there is no way to re-secure the secret. In the business hierarchy example presented earlier, the access code would need to be changed at every use, and

every person involved with the protocol would need a new share (as well as all extra data required).

Future research on secret sharing would require that we extend the protocol to overcome these limitations. One idea would be to explore the use of a secure co-processor [9, 10] with the protocol. Proper use could remove the need for any form of check vector requirement as well as allowing the secret to be re-secured; if the secure co-processor is the only entity that interpolates and discovers the secret, then no participant will ever have knowledge of it. This idea naturally scales to handle tamper-proof smart cards.

7 Acknowledgments

I would like to thank my advisor, Doug Tygar, for his amazing support and help in creating this work. I cannot thank my parents enough, without whom I doubt very much I would be here. I'd also like to thank the Logic & Computation seminar for giving me some different perspectives, the Information and Networking Institute for funding my Small Undergraduate Research Grant, and Mark Stehlik for being an all-around great guy.

A Notation

Check Vector A pair (b, c) used to verify a share s.

Dealer The person who originally has the secret and distributes the shares to the n people.

Participant $(A, B, C, etc.)$ A person involved with recomputing the secret.

k The number of bits in the secret.

n The number of people involved with the secret sharing.

p A large public prime.

q The quorum of people needed to re-create the secret (this must be less than n).

S The secret. For simplicity, we'll assume this is an integer modulo p.

s (*share*) An integer value which, when combined with other shares, re-computes the secret.

$s_{[i]}$ The i^{th} bit of s.

$\vec{y}_{ABi}$ The i^{th} component of vector $\vec{y}$ which is used by B to verify information from A.

References

[1] Josh D. Cohen. Keeping shares of a secret secret. Technical Report YALEU/DCS/TR-453, Yale University Dept. of Computer Science, February 1986.

[2] Maurice P. Herlihy and J. D. Tygar. How to make replicated data secure. In Carl Pomerance, editor, *Lecture Notes in Computer Science #293*. Advances in Cryptography, Springer - Verlag, 1987.

[3] Michael O. Rabin. Efficient dispersal of information for security load balancing and fault tolerance. Technical report, Aiken Computation Laboratory, Harvard University, 1986.

[4] Michael O. Rabin and J. D. Tygar. An integrated toolkit for operating system security. In W. Litwin and H.-J. Scheli, editors, *Lecture Notes in Computer Science #367*, Paris, France, June 1989. Foundations of Data Organization and Algorithms, Springer - Verlag.

[5] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). *Communications of the ACM*, pages 73–85, 1989.

[6] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[7] Gilbert Strang. *Linear Algebra and its Applications*. Harcourt Brace Jovanovich, Inc., third edition, 1988.

[8] Martin Tompa and Heather Woll. How to share a secret with cheaters. Research Report RC 11840, IBM Research Division, March 1986.

[9] J. D. Tygar and B. S. Yee. Pysically secure coprocessors. Technical Report CMU-CS-91-140R, Carnegie Mellon University, May 1991.

[10] J. D. Tygar and B. S. Yee. Strongbox. In Jeffrey L. Eppinger, Lily B. Mummert, and Alfred Z. Spector, editors, *Camelot and Avalon: A Distributed Transaction Facility*. Morgan Kaufmann, 1991.