Secret Bit Transmission Using a Deck of Cards

Lynn Reggia

December 19, 2007

1 Introduction

Alice, perhaps a spy or dishonest bridge player, wishes to transmit a single bit s' secretly to Bob. To ensure secrecy Alice does so by using a one-time pad, in this case, one bit long. A shared secret bit s must first be established with both Alice and Bob. Using s, Alice calculates $q = s \oplus s'$ and sends the value to Bob, who then retrieves $s' = q \oplus s$. Of course, Alice and Bob must first establish the bit s.

We will begin by presenting a protocol for the transmission of one bit. We will then extend this protocol to establish a multiple bit secret.

1.1 Assumptions

As always, assume that there is a malevolent third party Eve. Eve is trying to discover s. We assume Eve has knowledge of whatever protocol Alice and Bob are using, she has infinite computing power, and that she can hear everything said between Alice and Bob. Therefore it is essential that whatever protocol Alice and Bob use allows them to establish the bit s while providing no information to Eve. It is not good enough that it is only unlikely Eve discover s, after all she has infinite computing power, but it must be impossible that she can do better than simply guess.

1.2 A Deal of Cards

The shared secret bit s will be established through the use of a deck of n unique, ordered cards. Each of Alice, Bob, and Eve will be dealt a hand.

A random deal of cards is an ordered pair (a, b, e) where, out of the deck, Alice is dealt a cards, Bob is dealt b cards, and Eve is dealt $e = n - (a + b)$ cards. Each player only knows the cards held in their own hands. However, a, b, e, and n are all public information.

Under set parameters a *random deal* of cards can be used to establish a shared secret bit s between Alice and Bob.

2 A Randomized Algorithm

A random deal of cards (a, b, e) can be used to establish a secret bit s between Alice and Bob.

2.1 Definitions

A *pair* $p = (x, y)$ consists of two cards from the *random deal*, x and y. One of them is held by Alice and the other by Bob.

Assume Alice holds the card x in her hand and Bob holds the card y. If Alice and Bob both know the pair $p = (x, y)$ exists, they therefore know who holds which card by a simple process of elimination. Eve, however, even if she knows the pair p exists, cannot tell which card Alice holds and which one Bob holds. The best she can do is guess at who holds what.

2.2 Use p to Establish s

Assume that Alice and Bob can establish a pair $p = (x, y)$. In doing so they know who holds x and who holds y. Recall that x and y come from a deck of unique, ordered cards, so we can establish an operator to determine if $x < y$ or if $x > y$. We will have Alice and Bob agree beforehand that if Alice holds the lesser of the two cards, $s = 1$ and if Alice holds the larger card $s = 0$.

Once the pair p is established Alice and Bob can therefore determine s. Eve, however, has no knowledge of who holds what card, so she cannot determine s. The value s will be Alice and Bob's one time pad. [1]

2.3 Establishing a Pair

Alice and Bob can establish a pair p. Begin with a random deal (a, b, e) $a \geq 1, b \geq 1, e \geq 0$ from the deck and proceed as follows:

1. For simplicity, and without loss of generalization, call whichever player who holds the most cards Alice and the other player Bob. Therefore, in all cases $a \geq b$.

2. Alice selects a card x in her hand and a card y not in her hand to try to create a pair p. She then announces the pair, either $p = (x, y)$ or $p = (y, x)$, to everyone. She will randomly choose which of these two pairs to announce. This ensures that Eve cannot determine who holds what by the order in which x and y are announced in the pair.

3. If Bob holds y he says that p is a valid pair. Now Alice and Bob have established a pair between them and establish s.

4. If Bob does not hold y say that p is not a valid pair. Alice announces the locations of the cards x and y to all, namely that she holds x and Eve holds y. These cards are then discarded from the deck, and the protocol starts over with a new random deal $(a - 1, b, e - 1)$.

The protocol ends when either $a = 0$ or $b = 0$.

3 Establishing Multiple Bits

To transmit multiple bits Alice and Bob use several repeated rounds of the one bit procedure. Rather than stopping after successfully establishing a valid pair, and subsequently getting a shared bit, Alice and Bob discard both cards $x, y \in p$. Now the protocol repeats with a new random deal $(a - 1, b - 1, e)$.

This continues, with Alice and Bob either establishing shared bits or outing one of Eve's cards each round, until either $a = 0$ or $b = 0$. [1]

3.1 Code

To implement this protocol, a program using Java 5.0 was written. It consists of two classes: SecCards, and a supporting class, Person.

The Person class is used to represent Alice, Bob, and Eve. Each Person has two decks of cards, represented as Integer ArrayLists. One deck contains the cards in a Person's hand, the other contains all cards in the entire deck of which the Person does not know the location (i.e., the cards not in the Person's hand and that have not been removed from the deck from previous guesses). The Person class also contains a constructor to initialize the two decks and a toString() method to print out a Person object in a clear manner.

The SecCards class is the core of the program. It consists of four methods: main(), takeTurn(), getWorstCase() and playGame(). The main method calls the playGame() method, sending it the number of cards Alice has (a), the number of cards Bob has (b), and the total number of cards ($a + b + e$) , each time it wishes to run a game. This is done many times, and for many different values of a, b, and e. The main method stores the number of shared bits acquired by Alice and Bob after each game, and calculates and prints a, b, e, the minimum, maximum, mode, and average number of bits acquired for each set of runs, and the minimum number of bits guaranteed by a game played with values a, b, and e. The guaranteed number of bits is calculated by calling the getWorstCase() method. The playGame() method initializes each of Alice's, Bob's, and Eve's decks with a random deal of cards, and then repeatedly calls the takeTurn() method, sending which player's turn it is, Alice's or Bob's, depending on who has more cards. The takeTurn() method randomly chooses a card from the player's deck whose turn it is, and a card not in their deck, and then removes these cards from the appropriate decks, according to the above multiple bit protocol. If a shared bit is acquired, a counter is incremented.

3.2 Results

Through repeated testing of this protocol, I have found that frequently the number of bits acquired by Alice and Bob is higher than the minimum number of bits they are guaranteed. Given the situation with values $a = n$, $b = 200 - n$, and $e = a + b - 1$, Alice and Bob are never guaranteed to share a bit. However, if the protocol is run 1000 times each for $n = 10, 20..., 100$, the much higher average values in Table 1 are the result.

The average number of shared bits actually increases as a and b get closer to each other in this case. This makes sense, as Alice and Bob can only share as many bits as whichever one of them has the fewest cards.

Alice and Bob are guaranteed no bits also in the situation with values

Table 1:

$Alice$	Bob	Eve	MAX	MIN	AVG	$MODE$
10	190	199	10	6	9.07	9
20	180	199	20	10	16.449	17
30	170	199	28	14	22.315	22
40	160	199	35	17	26.78	27
50	150	199	39	21	30.544	30
60	140	199	43	22	33.121	33
70	130	199	47	23	34.856	33
80	120	199	47	25	36.075	35
90	110	199	50	25	36.678	36
100	100	199	49	27	36.946	38

$a = n$, $b = n$, and $e = 2n - 1$. Table 2 shows the results for running this protocol 1000 times each for $n = 10, 20..., 100$

In both of these situations, the average number of acquired bits is much greater than the guaranteed minimum number of bits. Further research is being done on concluding exactly why this is the case.

4 A Deterministic Algorithm

A deterministic (Non-random) protocol also exists for Alice and Bob to exchange a secret bit.

4.1 Single Bit Protocol

This protocol works similarly to the randomized algorithm. Begin with a random deal of cards (a, b, e) that will be used to establish the shared secret s.

1. Assign each of the $n = a + b + e$ cards a unique index 0 through $n - 1$.
2. If $e = 0$ and both $a \geq 0$ and $b \geq 0$ then we are finished. Alice and Bob have assigned each of the $\binom{n}{a}$ possible $(a, b, 0)$ deals a unique index. Alice and Bob now can establish a shared secret bit based upon the index.

Table 2:

Alice	*Bob*	*Eve*	*MAX*	*MIN*	*AVG*	*MODE*
10	10	19	8	1	3.82	4
20	20	39	13	3	7.545	7
30	30	59	19	6	11.21	11
40	40	79	23	7	14.898	15
50	50	99	27	9	18.484	18
60	60	119	31	13	22.34	23
70	70	139	36	18	25.768	26
80	80	159	39	19	29.631	29
90	90	179	43	23	33.252	32
100	100	199	48	25	36.944	36

3. If $e \neq 0$ Alice, Bob, and Eve consider the deck as consisting of $\lfloor n/2 \rfloor$ blocks of two cards each. So card $2m$ is in block m rank 0 and card $2m+1$ is in block m rank 1. If n is odd discard card $n-1$.

   For example, in a deck of $n = 6$ cards the three blocks are $(1,2)$, $(3,4)$, and $(5,6)$.

4. For each block m from 0 through $\lfloor n/2 \rfloor$, Alice and Bob each announce whether or not they hold one of the cards belonging to block m, that is, they hold either card $2m$ or $2m+1$ but not both.
   If both Alice and Bob hold one card belonging to block m they have established a pair and can compute a shared secret bit. Otherwise they know that Eve holds the other card of block m. They repeat this procedure of announcing block for each singleton set in their hand.

5. If Alice and Bob cannot establish a pair they establish a new random deal (a', b', e'). Each of Alice, Bob, and Eve discards all cards belonging to a block m if they hold only one card of that block. If Alice, Bob, or Eve hold both cards of a block m, they discard the rank 1 card of that block.

6. If $a' = 0$ or $b' = 0$ the protocol fails. Otherwise apply it recursively on the new random deal (a', b', e'). [1]

4.2 Exchanging Multiple Bits

As with the randomized protocol, enabling the exchange of multiple bits using the deterministic protocol requires some small changes.

1. Assign each of the $n = a + b + e$ cards a unique index 0 through $n - 1$.
2. As in the single bit protocol, consider the deck as consisting of two card blocks. If n is odd discard card $n - 1$.
3. For each block m from 0 through $\lfloor n/2 \rfloor$, Alice and Bob each announce whether or not they hold only one of the cards belonging to the block m.
4. If both Alice and Bob hold one card belonging to block m they have established a pair and can compute one shared secret bit as in the randomized algorithm. They discard both cards; otherwise they know that Eve holds the other card of block m and instead either Alice or Bob (Whomever holds the card) and Eve discard a card. They continue trying to establish further bits using the remaining blocks of the deal.
5. If $a' = 0$ or $b' = 0$ the protocol ends. Alice and Bob now establish a new random deal (a', b', e'). Each of Alice, Bob, and Eve discards all cards belonging to a block m if they have only one card of that block. If Alice, Bob, or Eve hold both cards of a block they discard the rank 1 card. Apply the protocol recursively on the new random deal (a', b', e')

The primary change in the multiple bit version is that the $e = 0$ case no longer exists.

4.3 Code

Again, to implement this protocol, a program using Java 5.0 was written. It consists of two classes: DetSecCards, and a supporting class, Person.

The Person class is used to represent Alice, Bob, and Eve. Each Person has a deck of cards, represented as an Integer ArrayList. The Person class also contains a constructor to initialize the two decks and a toString() method to print out a Person object in a clear manner.

The DetSecCards class is the core of the program. It consists of four methods: main(), getSubset(), doRound(), playGame(). The main method calls the playGame() method, sending it the number of cards Alice has (a), the number of cards Bob has (b), and the total number of cards $(a + b + e)$, each time it wishes to run a game. This is done many times, and for many different values of a, b, and e. The main method stores the number of shared bits acquired by Alice and Bob after each game, and calculates and prints a, b, e, and the minimum, maximum, mode, and average number of bits acquired for each set of runs. We're still working on calculating the minimum number of bits guaranteed by a game played using the deterministic protocol, and a method to calculate this will be added once this is accomplished. The playGame() method initializes each of Alice's, Bob's, and Eve's decks with a random deal of cards, and then repeatedly calls the doRound(), dealing with removing and replacing the $n - 1$ card if n is odd. The playGame() method also calls the getSubset() method after each round. The doRound() method implements one round of the deterministic protocol, dividing the deck into ranks and suits. For each shared bit acquired, a counter is incremented. The getSubset() method returns a new deck after removing all cards of rank 1.

4.4 Results

Through repeated testing of this protocol, I have found that frequently the number of bits acquired by Alice and Bob is higher than the minimum number of bits they are guaranteed. In this protocol, also, the average is even higher than the minimum than in the non-deterministic protocol. Given the situation with values $a = n$, $b = 200 - n$, and $e = a + b - 1$, Alice and Bob are never guaranteed to share a bit. However, if the protocol is run 1000 times each for $n = 10, 20..., 100$, the much higher average values in Table 3 are the result.

Again, the average number of shared bits actually increases as a and b get closer to each other in this case. This makes sense, as Alice and Bob can only share as many bits as whichever one of them has the fewest cards.

Alice and Bob are guaranteed no bits also in the situation with values $a = n$, $b = n$, and $e = 2n - 1$. Table 4 shows the results for running this protocol 1000 times each for $n = 10, 20..., 100$.

In both of these situations, the average number of acquired bits is much greater than the guaranteed minimum number of bits. Further research is being done on concluding exactly why this is the case.

Table 3:

$Alice$	Bob	Eve	MAX	MIN	AVG	$MODE$
10	190	199	10	0	4.744	4
20	180	199	16	2	9.335	9
30	170	199	22	5	13.127	13
40	160	199	30	8	16.828	16
50	150	199	31	9	20.083	20
60	140	199	32	10	22.272	22
70	130	199	35	14	24.619	24
80	120	199	35	15	25.82	26
90	110	199	40	15	26.987	28
100	100	199	39	15	26.898	25

Table 4:

$Alice$	Bob	Eve	MAX	MIN	AVG	$MODE$
10	10	19	7	0	2.716	3
20	20	39	11	1	5.453	6
30	30	59	15	2	8.109	8
40	40	79	18	3	10.915	11
50	50	99	22	6	13.55	12
60	60	119	26	8	16.199	16
70	70	139	28	10	18.964	19
80	80	159	34	13	21.66	23
90	90	179	37	14	24.464	24
100	100	199	41	17	27.283	26

References

[1] M. S. P. Michael J. Fischer and C. Rackoff. Secret bit transmission using a random deal of cards. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.*