CS-2362: Computer Security and Privacy
Instructor: Mahavir Jhawar

Coin Flipping by Telephone

Understanding and Implementation

May 01, 2022
Anjali Sajith, Shambhavi Kurup

Contents

Introduction

In this paper, we explore Manuel Blum's 1980 paper on coin flipping by telephone, a seemingly impossible problem. Blum starts by introducing the problem faced by recently divorced Alice and Bob currently living in different cities, trying to decide who gets the car by a coin flip over the telephone. At first glance, there seems to be no way for Bob to verify the results of a coin flipped by Alice.
This problem can be extended to any situation where Alice and Bob are adversaries that do not trust each other but want to communicate in a way that ensures that neither one of them can cheat and also ensures that both parties can check at every step that the other has not cheated.

Note: The corresponding code files are attached in the zipped file along with this report.

Ideation

The idea is to use a function that ensures that Alice is not able to cheat by changing the coin flip after Bob tells her his guess, and that Bob shouldn't be able to cheat by making a non-random guess. We start with a simple enough function f that maps the set of $2n$-bit strings to the set of n-bit strings. We define the process of the *coin flip* as follows:
Alice picks an $x \in \{0,1\}^{2n}$ randomly, computes $f(x)$ and sends it over to Bob. The function f here is public, and so is also available to Bob. The randomly picked x plays the role of the coin flip. Now, Bob will make his guess. In this case, since we are dealing with binary strings, Bob guesses whether the x picked by Alice is odd or even. Alice declares whether or not Bob was right, and sends Bob x so that he can compute $f(x)$ himself to verify that she is telling the truth.

For this to work, two things must be taken care of:
1. Once Alice sends $f(x)$ over to Bob, he must not be able to infer anything about x from $f(x)$ that would make his guess better than a random guess. f must not be invertible. This is the one-way property of f.
2. Once Bob sends Alice his guess of whether x is odd or even, Alice must not be able to change the originally picked x i.e., Alice must be unable to find a different $x' \in \{0,1\}^{2n}$ that maps to $f(x)$. This property of f is Second Pre-Image Resistance.

An attempt at constructing a simple f

Let $f : \{0,1\}^{200} \longrightarrow \{0,1\}^{100}$
For $x \in \{0,1\}^{200}$, let x be read as $x = x_1||x_2$ where $x_1, x_2 \in \{0,1\}^{100}$
So, x_1 is the binary string of the 100 most significant bits of x, and x_2 is the binary string of the 100 least significant bits of x.
$f(x) = x_1 \vee x_2$, where $\vee$ is the bitwise OR operator

Can Bob cheat?

We compute the probability of Bob winning the toss given $f(x)$ from Alice.
Say Bob uses the following strategy - he guesses that x is odd if $f(x)$ is odd, and he guesses that x is even if $f(x)$ is even.

We know that a binary string is even if its least significant bit is 0, and odd if its least significant bit is 1.
x is chosen at random $\implies \mathbb{P}[x \text{ is odd}] = \frac{1}{2} = \mathbb{P}\ [x \text{ is even }]$

$\mathbb{P}$ [Bob wins] $= \mathbb{P}$ [Bob wins $|$ x is odd] $\mathbb{P}$ [x is odd] + $\mathbb{P}$ [Bob wins $|$ x is even] $\mathbb{P}$ [x is even]

When x is odd, the least significant bit of x is 1 (from fundamental binary rules).
$\Rightarrow \mathbb{P}[f(x)$ is odd] = 1 (since $1 \vee 0 = 1 \vee 1 = 1$)
$\therefore \mathbb{P}[f(x)$ is odd $|$ x is odd] = 1.

When x is even, the least significant bit of x is 0 (from fundamental binary rules).
$\Rightarrow \mathbb{P}[f(x)$ is even] = 1/2 (since $1 \vee 0 = 1$, and $0 \vee 0 = 0$)
$\therefore \mathbb{P}[f(x)$ is even $|$ x is even] = $\frac{1}{2}$.

$$\mathbb{P}\ [\text{Bob wins}] = 1 \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$$

Clearly, the probability of Bob guessing correctly when given $f(x)$ is much greater than $\frac{1}{2}$.

Can Alice cheat?

We compute the probability of Alice winning by modifying x after receiving Bob's guess. This is irrespective of whatever strategy Bob uses to make his guess. We consider the two cases:

Case 1: x was even

This means that the least significant bit of x is 0.
If the least significant bit of x_1 is 1 (which has a probability of $\frac{1}{2}$), then the least significant bit of $f(x)$ would be 1. In this case, if Bob guessed correctly that x was even, *Alice can cheat* by simply flipping the least significant bit of x to 1, and sending this modified x to Bob for verification. The modified x would map to the same image since $0 \vee 1$ is also 1. On the other hand, if the least significant bit of x_1 is 0 (which also has a probability of $\frac{1}{2}$, then the least significant bit of $f(x)$ would be 0. In this case, if Bob guessed correctly that x was even, Alice would not be able to cheat since making x odd would yield a different image.

$$\therefore \mathbb{P}\left[\text{Alice wins by cheating} \mid x \text{ is even}\right] = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = \frac{1}{2}$$

Case 2: x was odd

This means that the least significant bit of x is 1.
The least significant bit of $f(x)$ clearly cannot be 0 in this case.
So the least significant bit of $f(x)$ is always 1 if x is odd. If Bob guesses correctly that x is odd, Alice can cheat by simply flipping the least significant bit of x to 0, and we can clearly see that the image of this modified x would remain the same.

$$\therefore \mathbb{P}\left[\text{Alice wins by cheating} \mid x \text{ is odd}\right] = 1$$

$\mathbb{P}$ [Alice wins by cheating] $=$ $\mathbb{P}$ [Alice wins by cheating $\mid x$ is odd] $\mathbb{P}$ [x is odd] + $\mathbb{P}$ [Alice wins by cheating $\mid x$ is even] $\mathbb{P}$ [x is even]

$$= 1 \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$$

Again we see that the probability of Alice winning by manipulating x after knowing Bob's guess is significantly greater than $\frac{1}{2}$.

The Protocol Used

Alice and Bob require a protocol that gives them both a success probability of $\frac{1}{2}$ each. We can construct this protocol using the ideas of quadratic residue and congruent modulo arithmetic. The following approach would allow this scheme to have similar security as RSA.

We begin with Alice choosing two large prime numbers that are congruent to $3 \mod 4$.
Let these prime numbers be p and q. p and q are not shared with Bob, but rather an n where $n = pq$ is sent to Bob. Bob will take in this n and will choose a random x between 1 and $n-1$. He will go on to calculate a number a where $a = x^2 \mod n$ such that $x^2 \equiv a \mod n$ and $0 < a < n$. This a is the number that will be sent to Alice. Now, Alice will try to guess the value of x using the sent a and her information on the factors of n. This is where the "coin flipping" takes place. Using the values of p and q and the value of a, Alice finds out all the possible values that could be equal to the x that Bob chose (i.e., the roots of the congruence). This implies that she finds any integer x that would satisfy $x^2 \equiv a \mod n$.

Here, there will be exactly four such integers (four such square roots) that could constitute the value of x. One of these four roots would naturally be x. Then another would be a different value x' and the remaining two values would be $n-x$ and $n-x'$. Out of these four, Alice sends one randomly picked value to Bob. Now, the scheme is set up in such a way that Alice would win if she correctly guesses the value of x, that is if she sends either x or $n-x$. In this case, Alice will then send Bob the factors p and q so that Bob can verify the process for himself. On the other hand, Alice would lose if she sends either x' or $n-x'$. In the case that Alice lost, Bob would use the value that Alice sent (which gives him all 4 possible square roots of x) and would then calculate the values of p and q and send it to Alice, along with the actual value of x, to prove his success.

Can Bob cheat?

Since we are starting off with 2 large prime numbers and then sending the product of these primes, it is very hard for Bob to identify the two primes p and q (since factorization is a computationally hard problem). The largest number that has been quantum factorized to date is 56,153. In a realistic scenario p and q would themselves be numbers with more than 100 digits, making the factorization near impossible. This allows the given function to be a one-way function, similar to hashing. If this wasn't the case, then Bob would be able to calculate the other possible square roots of x himself and use the Euclidean algorithm to figure out the values of p and q.

Suppose Bob does know another root x' along with x such that x' is not congruent to $x \mod n$. Then $x'^2 \equiv a \equiv x^2 \mod n$.

$\Rightarrow n$ divides $x'^2 - x^2$

$\Rightarrow n$ divides $(x'+x)(x'-x)$

Here if n divides $(x'-x)$, then it would mean that $-x' \equiv x \mod n$. But this contradicts our assumption.

If n divides $(x'-x)$ then it would mean that $x' \equiv x \mod n$. But this also contradicts our assumption. Therefore, as n does not divide $(x'-x)$ or $(x'+x)$, the factors of n, that is p and q must each divide either $(x'+x)$ or $(x'-x)$. That is

$p|(x'+x)$ and $q|(x'-x)$ *or* $p|(x'-x)$ and $q|(x'+x)$.

This implies that the greatest common divisor of n and $(x'-x)$ would be one of p or q and greatest common divisor of n and $(x'+x)$ would be the other. This is the method that Bob uses if he wins, to find p and q after getting to know the other square root x' from Alice's wrong guess.

Bob could cheat by sending Alice an a that has no square root. But this won't work since Alice knows the factorization of n, and would figure out that there are no square roots of a. Bob could also send Alice a randomly picked a in the required range. We just explained why this wouldn't work if a has no square roots. But even in the case where it does, this wouldn't work since Bob needs to know at least 2 roots to have any chance of winning at all.

Can Alice Cheat?

In the beginning, while selecting p and q, Alice can cheat by sending numbers that are either not relatively prime or not primes themselves. But this would cause a disadvantage for Alice herself as this increases the number of possible roots from which she must randomly guess Bob's choice of x, making her probability of success lesser than $\frac{1}{2}$. In the case where Alice does not cheat, there would only be 4 possible square roots of the congruence (four possible guesses) as n is a factor of two primes and each prime would give two possible square roots. In this scenario, Alice's probability of winning is $\frac{1}{2}$ as Bob accepts the value of x or $n-x$.

Time Complexity Improvement

In our first implementation of this protocol, we found the four roots of x by running the program from 1 to n and taking in every value when squared and divided by p, that would give remainder equal to $a \mod p$ and every value when squared and divided by q, that would give remainder equal to $a \mod q$. Then out of these values, we found the values that were common in both (there would always be four such values) and these were the roots of x. While this follows our logic and gives us the correct roots, it takes $O(p)$ time. In a realistic scenario, where the initial p and q themselves would be of 100 bits, a program that runs on polynomial time would take quite a while to complete and is not practical. Using properties of $3 \mod 4$ and number theory concepts such as Extended Euclidean Algorithm, Fermat's Little Theorem and the Chinese Remainder Theorem, we found a more efficient method to find the values of the roots.

We are generating a p and a q that is an odd prime. By Euler's Criterion, which states that $x^2 \equiv a(\mod p)$ will only have solutions if and only if $a^{\frac{p-1}{2}} \equiv 1(\mod p)$. By virtue of the creation of a, we know that $x^2 \equiv a(\mod p)$ will have solutions. This means that $a^{\frac{p-1}{2}} \equiv 1(\mod p)$ is true. Multiplying with a on both sides, we get $a^{\frac{p+1}{2}} \equiv a(\mod p)$. We can rewrite $a^{\frac{p+1}{2}}$ as $(a^{(\frac{p+1}{4})})^2$.

The significance of $3 \mod 4$

As p and q are of the form $3(\mod 4)$, $p+1$ is divisible by 4. Taking the square root of $a(\mod p)$ would be as follows:

$$\left(\pm a^{\frac{p+1}{4}}\right)^2 \equiv a^{\frac{p+1}{2}} \equiv a^{\frac{p-1+2}{2}} \equiv a^{\frac{p-1}{2}+1} \equiv a^{\frac{p-1}{2}} \cdot a \pmod{p}.$$

And hence, by virtue of p and q being of the form $3 \mod 4$, finding the solution to the square root of $a(\mod p)$ can be calculated easily by $\pm a^{\frac{p+1}{4}}$. This ease would not exist if p and q were of any other form, say $1(\mod 4)$.

We take this one step further, to reduce time complexity, by substituting $p = 4k+3$ in $a^{\frac{p+1}{4}}$, giving us $a^{\frac{4k+3+1}{4}} = a^{k+1}$. We use recursion to convert $k+1$ into its binary form, i.e., we make $k+1 = b_m.2^m + b_{m-1}.2^{m-1} + ... + b_0$ which would give us $a^{k+1} = a^{(b_0)}.(a^2)^{b_1}.(a^4)^{b_2}...(a^{2^m})^{b_m}$. Using this method, we calculate the solution to $a \mod p$ (let it be s_1), as well as $a \mod q$ (let it be s_2). By using the Extended Euclidean Algorithm, we are also able to calculate α and β such that $\alpha.p + \beta.q = 1$.

Now we take all the solutions that we found and consider the four values of
$t = \pm s_2 \alpha p \pm s_1 \beta q (\mod n)$.
We see here that $t^2 \equiv (\pm s_2 \alpha p)^2 \equiv (s_2)^2 (\alpha p)^2 \equiv (a \mod p)^{\frac{2}{2}} (1)^2 \equiv a (\mod p)$
Similarly, we find that $t^2 \equiv a (\mod q)$. Here, we see that the distinct primes p and q both divide t, which indicates that t will give us the solutions to the value of x. Thus, we calculate the four values of the roots without traversing through the entire range of n. Since we are using recursion to calculate the roots the program only takes $O(\log n)$ time to run, giving us a more efficient program.

An Optimally Fair Coin Toss - Moran, Naor, Segev

How their protocol improves on Blum's

Blum's coin-flipping protocol ensures that Alice cannot increase the probability of her getting the car (i.e., winning) by a non-negligible amount, and likewise, Bob cannot increase the probability of him getting the car. The protocol gives both parties an incentive to behave honestly and follow the steps of the protocol correctly, as not doing so will not result in a significant gain in the probability of their win. However, the protocol assumes that Alice is interested only in getting the car herself i.e., Alice does not wish to sway the outcome of the flip in Bob's favour. This means that the protocol does not protect against Alice cheating to increase the probability of Bob winning, and vice versa. This is what makes the protocol weak, compared to other ”strong” coin flipping protocols.
For example, Alice can potentially cheat in this way by sending Bob non-primes p and q. This would put her at a disadvantage as it would increase the number of possible roots from which she must randomly guess Bob's choice of x, and this would naturally make her success probability decrease significantly from the ideal $\frac{1}{2}$.

Moreover, if Alice or Bob are malicious, once the cheating party is labeled as malicious, the other honest party is allowed to halt without giving any output in Blum's protocol. But in many cases, we require a more robust protocol with does not allow this to happen. In other words, the protocol should always ensure that there is an unbiased output of the coin flip. So in cases where a malicious party aborts prematurely i.e., before it finds out the result of the coin flip results, fairness must still hold and the honest party is then required to output a bit.

Protocol Description

With trusted dealer:
The protocol described in the paper has two versions, the first being a simpler version involving the presence of a trusted third party in the pre-processing phase of the protocol. This trusted party simply computes and sends over separately the input that both parties use during the actual coin flip. It is important to note that Moran et al. go on to describe the second version of the protocol (the actual one proposed for real world implementation) that eliminates the need for this trusted dealer by replacing the functionality with another protocol with a constant number of rounds that keeps the time complexity of the protocol the same.

Before going into the protocol, we describe the basic idea behind the working using an example with cards where the outcome of the coin flip is equated to the colour of a card i.e., heads and tails are equivalent to red and black. All parties initially know how many rounds there are going to be in the protocol. Let the number of rounds be denoted by r. The trusted dealer also decides the outcome of the flip by randomly picking red or black at the beginning of the protocol. Then the dealer randomly picks a number between 1 and r. Let this number be i^*. In every round, the dealer gives a card to Alice and a card to Bob. Alice is not privy to the cards that Bob gets, and Bob is not privy to the cards that Alice gets. For the first $i^* - 1$ rounds, the cards that Alice and Bob receive are picked randomly by the dealer. The cards that they receive from round i^* to round r are all of the same colour, and this colour is the colour that the dealer picked in the beginning of the protocol as the outcome of the coin flip. To make this more clear, let's say that the number of rounds are 10, and the dealer picked i^* to be 8 and the outcome of the 'flip' to be red. Let us assume that Alice calls red and Bob calls black. In this case, the dealer would give random cards to Alice and Bob in rounds 1-7. But from the 8th round onwards, the dealer gives Alice and Bob only red cards.

If neither Alice nor Bob abort in the middle of the protocol, then the outcome of the flip is the colour of the card that Alice and Bob received in the last round. (Note: It is easy to see that the last card must be red no matter what i^* the dealer picked).

Now consider the case where one party, say Alice, is trying to cheat. Say the current round is 7, and the sequence of cards she has gotten till now is RBRBBBB. At this point, all Alice knows is that there are 10 rounds, and these are the cards she has received. She doesn't know what cards Bob has gotten, nor does she know what cards she will get in the subsequent rounds. Since the last four values she got are Black, she thinks that the dealer might have picked i^* to be some number greater than 3, and so the rest of the cards she is about to get from the dealer will all be black. This would mean that the outcome of the flip is black, and she would lose since her call was red. So she aborts during round 7, thinking that the actual outcome was black. This makes sense from her perspective since she thinks her current probability of winning is 0.
In such a situation, the outcome of the flip then becomes the colour of the card that Bob received in the previous round, which would be the 6th round here. Clearly, we see that the honest party (Bob) is required to output a bit (card) even if the malicious party (Alice) aborts in the middle of the protocol. It is also clear to see that the outcome of the flip remains completely random since without the knowledge of i^*, the probability of Bob's 6th card being red or black is exactly $\frac{1}{2}$ each.

With this basic idea, we can now describe the actual protocol. It begins with both parties being given a security parameter 1^n and r, where $r = r(n)$ and denotes the number of rounds in the protocol. Then we have a pre-processing phase in which the dealer randomly picks $\omega \in \{0,1\}$ (where ω is the outcome of the coin flip) and $i^* \in \{1,\ldots,r\}$ randomly. The pre-processing phase isn't complete here. The dealer picks all the bits to be given to the two parties - r bits for Alice and r bits for Bob - in the beginning itself.

The dealer picks $a_1,\ldots,a_{i^*-1},b_1,\ldots,b_{i^*-1} \in \{0,1\}$ randomly.
Then the dealer assigns $a_{i^*} = a_{i^*+1} = \cdots = a_r = a_{i^*} = a_{i^*+1} = \cdots = a_r = \omega$.

Note: In the following context, whenever we mention numbers being randomly picked from any set, we mean they are picked independently and with uniform distribution.

The idea is for the dealer to pre-process everything that the two parties will need for the protocol and give it to them in the beginning itself (this helps to later eliminate the need for the dealer from the protocol). So, from that point onwards, their is no interaction of either party with the dealer and he plays no role in the protocol. In the cards example, this would be equivalent to the dealer picking and setting aside 10 cards each for Alice and Bob and giving them their respective set of cards all together in the beginning. This is clearly not a good idea as the outcome of the coin flip can be gauged by simply looking at the last card given. This is dealt with using 2-out-of-2 secret sharing, which we will now explain.

We want Alice and Bob to have received their respective r bits at the beginning of the protocol, but we want the bits to be encrypted so that all the bits are not visible to them in the beginning itself. We want both parties to decrypt one new bit in each round. Essentially, in every round $i \in \{1,\ldots,r\}$, Alice should learn a_i and Bob should learn b_i. The 2-out-of-2 secret sharing scheme used for this is as follows:
Alice and Bob are both given r tuples, instead of r bits.
The dealer essentially splits every a_i into (a_{i1},a_{i2}), and b_i into (b_{i1},b_{i2}),
where $(a_{i1},a_{i2}),(b_{i1},b_{i2}) \in \{0,1\} \times \{0,1\}$ such that $a_{i1} \oplus a_{i2} = a_i$ and $b_{i1} \oplus b_{i2} = b_i$.
It creates these tuples by first randomly picking bits a_i1, and then computing a_{i2} by performing $a_i \oplus a_{i1}$. Similarly with tuples for Bob.

Clearly, a_i can only be reconstructed if one knows the values of both a_{i1} and a_{i2}, and b_i can only be reconstructed if one knows the values of both b_{i1} and b_{i2}. The secrets are then given to Alice and Bob to ensure that they get only one share each for each bit, and then in each round, exchange shares such that they can reconstruct their respective required bits.
Alice receives the following r tuples from the dealer: (a_{i1},b_{i1})
Bob receives the following r tuples from the dealer: (a_{i2},b_{i2})

In round i, Alice sends Bob b_{i1} and Bob sends Alice a_{i2}. This means that Alice sends Bob the part of the secret that he needs to reconstruct bit b_i, and Bob sends Alice the part of the secret that she needs to reconstruct a_i.
It is now clear to see that in every round, Alice and Bob will both learn a new bit each using the secret share that the other sends to them. The protocol goes on as follows till the last round i.e., the r^{th} round. If neither Alice nor Bob abort before the last round of the protocol, then the outcome of the coin flip is the bit $a^r = b^r = \omega$. If some party, say Alice is malicious and aborts prematurely in some round i, the protocol requires that the honest party still outputs an unbiased bit. This bit is Bob's bit of the previous round. This is essentially how their protocol ensures a more robust fairness than Blum's.

Yao's Protocol

Another protocol that attempts to provide a general solution for the problem of secure computation for two-party computation is Andrew Yao's protocol (also called the Garbled circuit). This protocol is an answer to Yao's Millionaire Problem (Alice and Bob wants to know who is richer among the two without revealing their wealth) and also enables two-party secure computation where two parties that do not trust each other can come together to jointly compute a function using their private inputs, without the need of a third party. A securely computed protocol would ensure *privacy*, where nothing is learnt from the protocol other than the output, and *correctness*, where the output is distributed according to the prescribed functionality. But in this protocol we can only ensure secure computation if all the parties involved are honest or *semi-honest* which means that the party follows the protocol but tries to gain additional information by analyzing messages that are received. If an adversary is malicious, secure computation would not hold true for this protocol.

In this protocol, Party 1 and Party 2 take in inputs x and y respectively. f is the functionality they compute. This f, described as a circuit, is known to both parties. Their aim is to use this functionality to jointly compute $f(x,y)$ without revealing their inputs x or y. Yao's protocol works by Party 1 generating an encrypted function (which here would be a "garbled" circuit) by computing the prescribed functionality on Party 1's input of x and then sending this to Party 2, along with Party 1's encrypted input. To ensure privacy is maintained, Party 2 will not be able to reveal anything about Party 1's input in this round. Party 1 and Party 2 proceeds in a total of n oblivious transfer protocol where Party 1 plays the sender and Party 2 plays the receiver. Here, Party 1's input is their encrypted input and Party 2's input is their input y. The output to this protocol would be Party 2's encrypted input. Using the encrypted input of theirs, Party 2 evaluates (decrypts) the circuit. This allows Party 2 to learn the output $f(x,y)$. Then Party 2 sends over $f(x,y)$ to Party 1. At the end of the protocol, both parties learn $f(x,y)$

Blum’s Protocol:

Alice		**Bob**
large primes p, q picked		
where $p \equiv q \equiv 3 \pmod 4$		
$n = pq$		
	$\xrightarrow{n}$	$x \xleftarrow{\$} \{1, \ldots, n-1\}$
		$a = x^2 \bmod n$
	$\xleftarrow{a}$	
Finds the four roots of $x^2 \equiv a \bmod n$:		
they are x, $n-x$, x', $n-x'$		
$y \xleftarrow{\$}$ {x,n-x,x’,n-x’}	$\xrightarrow{y}$	Output:
		if $y = x$ or $y = n-x$, Alice wins
		Alice must send p and q to Bob for verification.
		if $y = x'$ or $y = n-x'$, Bob wins
		Bob must compute p and q and send to Alice.

Moran, Naor, Segev's Protocol:

Pre-processing phase:

Dealer $(1^n, r)$

$i^* \xleftarrow{\$} \{1,\ldots,r\}$
$\omega \xleftarrow{\$} \{0,1\}$
$a_1,\ldots,a_{i^*-1}, b_1,\ldots,b_{i^*-1} \xleftarrow{\$} \{0,1\}$
$a_{i^*} = \cdots = a_r = b_{i^*} = \cdots = b_r = \omega$

$\forall 1 \leq i \leq r,$
$\quad a_{i1}, b_{i1} \xleftarrow{\$} \{0,1\}$
$\quad a_{i2} = a_i \oplus a_{i1}$
$\quad b_{i2} = b_i \oplus b_{i1}$

$\swarrow$ $\qquad\qquad$ $\searrow$

$(a_{i1}, b_{i1})\ \forall 1 \leq i \leq r$ $\qquad\qquad$ $(a_{i2}, b_{i2})\ \forall 1 \leq i \leq r$

Coin Flip:

Alice		**Bob**
$(1^n, r, (a_{i1}, b_{i1})\ \forall 1 \leq i \leq r)$		$(1^n, r, (a_{i2}, b_{i2})\ \forall 1 \leq i \leq r)$
	Round 1:	
	$\xrightarrow{b_{11}}$	
	$\xleftarrow{a_{12}}$	
$a_1 = a_{11} \oplus a_{12}$		$b_1 = b_{11} \oplus b_{12}$
	Round 2:	
	$\xrightarrow{b_{21}}$	
	$\xleftarrow{a_{22}}$	
$a_2 = a_{21} \oplus a_{22}$		$b_2 = b_{21} \oplus b_{22}$
	.	
	.	
	Round r:	
	$\xrightarrow{b_{r1}}$	
	$\xleftarrow{a_{r2}}$	
$a_r = a_{r1} \oplus a_{r2}$		$b_r = b_{r1} \oplus b_{r2}$

Output:

$a_r = b_r = \omega,$	if neither party aborts
$a_{i-1},$	if Bob aborts in round i
$b_{i-1},$	if Alice aborts in round i

Correctness:
In round i, Alice receives a_{i2} from Bob, and she already has a_{i1}.
She can now compute a_i as $a_i = a_{i1} \oplus a_{i2}$.
In round i, Bob receives b_{i1} from Alice, and he already has b_{i2}.
He can now compute b_i as $b_i = b_{i1} \oplus b_{i2}$.