Cryptography Project - Implementing Cut and choose protocol for voting

Navneet Agarwal - 140100090
Sanat Anand - 140040005
Sohum Dhar - 140070001

April 2017

Abstract

We have implemented the Multi-party Voting functionality in the UC setting against Computationally unbounded adversaries (Information theoretic setting). We've given the protocol and the proof outline here. We have implemented the protocol as a tool as part of our project.

Contents

1 Introduction

The idea of the problem is to allow for voting to happen without any malicious activity happening. For example it may happen that a party may give in a vote of 10 against the accepted convention of 0 and 1. No one gets to know the actual bit of the party so we have to design a protocol which can lead to success only if all the inputs are in the domain. The probability that the protocol doesn't abort when there is a corrupt input in the system is negligible.

2 Protocol Description

We have the functionality F available to us which implements the standard addition Functionality i.e given the votes of N parties it finds out the final vote value by adding up the values of these votes (This does not implement any integrity constraint of checking whether the votes are from {0,1} or some other value). We also have the And functionality which outputs the value of the And of the bits given to it by the various parties. Also, let N be the security parameter

- Run the functionality F, N times on random inputs (i.e. each party selects a random input from {0,1} for each execution of F)
- The aggregator chooses S $\subseteq$ [N] and asks the parties to reveal the inputs for the corresponding values during the N executions of F
- Check consistency of the values received in the second step by putting the values received by the parties in the second step into the functionality F and matching it with the value obtained from its actual execution in step 1. Do this for each of the N/2 executions of F.
- Execute the AND functionality N/2 times, once for each of the executions of F which has not been input revealed yet. The input of the parties for the AND here would be 1 in a particular functionality if their true input matches with the random input they gave to F and 0 otherwise.
- This would mean that the AND functionality corresponding to some execution of F would give 1 only if all the parties gave their true inputs for this execution of F. This would mean that the correct voting result is reflected by the output of F for this execution.
- We now check the number of executions of the AND functionality which result in 1 is within some reasonable bound around the expected number if the parties all behaved honestly. We use a multiplicative factor here as it helps our analysis. Therefore, if the number of AND results 1 is less than $0.5 * \mathcal{E}$, we would abort.

$$\text{Expectation of number of ANDs resulting in 1 } (\mathcal{E}) = \frac{N}{2 * d^m}$$

d = size of the domain (in this case it is 2)

m = number of parties

- The aggregator takes the majority of values in which the AND is 1 and output it as the final result

3 Proof Outline

3.1 Aggregator is corrupt

- If the aggregator is corrupt, then the only non-trivial thing for the adversary to do would be to learn something other than the honest parties residual function
- All communications between the aggregator and the parties is through the functionality F and AND which are assumed to be ideal
- Thus the security of the functionality guarantees the security of the new protocol. That is, if we were to break the security for this overall functionality then we could break the security of F or AND using that

3.2 Aggregator is honest

If the aggregator is honest, then the adversary could either attempt to learn something about the honest parties input or attempt to corrupt the input output by the honest aggregator. The only way it could do so is by deviating from the honest behaviour of parties. We will show that it cannot deviate too much. More formally, we show that the any deviation of the adversary cannot affect the final output or the probability of learning anything by more than a negligible probability.

3.2.1 Case 1 : Too many corrupt inputs sent

If c corrupt inputs are sent by the adversary in the executions of F, the probability of not being caught is negligible

- The prob of not being caught = $\frac{^{N-c}C_{N/2}}{^{N}C_{N/2}}$
- This can be proved using induction that the above element is $\leq 1/2^c$
- Therefore, if $c \geq \omega(log(poly(N)))$, then the probability of not being caught is negligible. Therefore, for the adversary to non-negligibly continue into the next step, $c = O(log(poly(N)))$

3.2.2 Case 2 : Too few corrupt inputs but too few ANDs as 1

The corrupt parties can potentially send very few corrupt inputs but send the bits of the AND as 1 for only those corrupt inputs. We show that this does not work as if it sends too few ANDs as 1, then the number of ANDs will be too low and the protocol will not continue into the next step. We also need to show that the probability of abort is negligible here in the case of honest parties.

- Expected Number of ANDs as 1 ($\mathcal{E}$) = $\frac{N}{2(2*d^m)}$. We have said that if the number of 1s obtained in AND is less that $0.5 * \mathcal{E}$, we will abort.
- We need to show that we don't abort if the inputs are being chosen randomly by the honest parties.
- We can show that by a careful application of the generic Chernoff bound that the probability of abort if the parties are honest is negligible.
- Therefore, by the nature of design the corrupt parties must give atleast $\Omega(N)$ AND values as 1 to avoid being aborted.

3.2.3 Case 3 : Few corrupt inputs and several AND values as 1

This is the final case where the corrupt parties have given few corrupt inputs and have given ample number of 1s in AND. Here we need to show that the majority taken by the aggregator will correspond to a corrupt value with only a negligible probability.

- We have seen that the number of 1s in the AND is $\Theta(N)$ and the number of corrupt inputs in just atmax $\Theta(log(poly(N)))$
- Therefore, there are all but log many honest inputs whose values are to be considered in the majority
- For the majority to correspond to one of the corrupt inputs, the number of corrupt inputs has to be substantially greater than the total number of inputs for which the AND is 1
- We show using appropriate probability bounds that, if all the conditions uptil this point are followed, the probability that the number of ANDs as 1 is less than twice the number of corrupt inputs is negligible thereby making it negligible for one of the corrupt functionality results to be output.

4 Implementation Details

4.1 Inputs to the parties and the aggregator

- The party has the set of ips of all parties, aggregator_ip, portno, vote value, security parameter k as the input
- The aggregator gets the set of ips of all parties, portno, security parameter k as its input

4.2 Connections among parties

- We have implemented the connections using socket programming
- We have assumed that the parties are having a secure channel amongst them for private communication and that the other parties' IPs, aggregator IP, port number and the security parameter values are known to all parties beforehand
- We have handled socket failures, like failure to bind to IP and port as a client/server, failure to read or write correctly from the socket. We retry if such failures occur and ensure the messages are correctly and completely delivered.
- Since ports are limited, we reuse ports for connections. The server and client set up a session and exchange a session id before proceeding for communication. This keeps different executions of protocols separate from each other.

4.3 ADD Functionality

- We have implemented a protocol in which every pair of parties share a random number among them first
- Then one of the two parties adds it to their value and the other party subtracts it from its value
- This is done for every pair of parties
- Finally every party sends its final value (which is the sum of this random number obtained though the previous process and its own input) to the aggregator who then adds it all to get the result vote value
- We had to run the above functionality N times so we used Multi-threading to achieve the same. We execute the N executions in batches. Of course, we need synchronization amongst the threads in writing to the log and manipulating the value to be sent to the aggregator for that execution.

4.4 AND Functionality

- This functionality is the same as vote just that if m input bit is 1, I send the calculated value (the random value obtained as a sum of the values obtained from step 1 and 2 of the previous part) and if my input bit is 0, I send a random value
- The aggregator then adds up all the values obtained from all the parties. If the resultant was 0 then it takes the AND value as 1 and otherwise it takes it as 0.

- The AND had to be computed on the remaining $N/2$ executions so we used threads to achieve the same. Of course, here too we need synchronization amongst the threads in writing to the log and manipulating the value to be sent to the aggregator for that execution.

4.5 Consistency Checks

- First check was that the values sent corresponding to the subset were consistent i.e their sum was equal to the precomputed value and their values were in the domain
- Second thing done was to check if the number of ands was within a factor of the expected value. If not then abort else give out the majority result (We do not abort in this case as such an abort can be made to be conditional by the adversary and it could potentially convey some information)
- To implement abort, we run a server on each party and aggregator listening for abort signal. We do not have conditional aborts. Of course we need to ensure that while we are sending abort signal we do not abort ourselves in between even if we receive an abort signal from some one else.

4.6 Corruptions Allowed

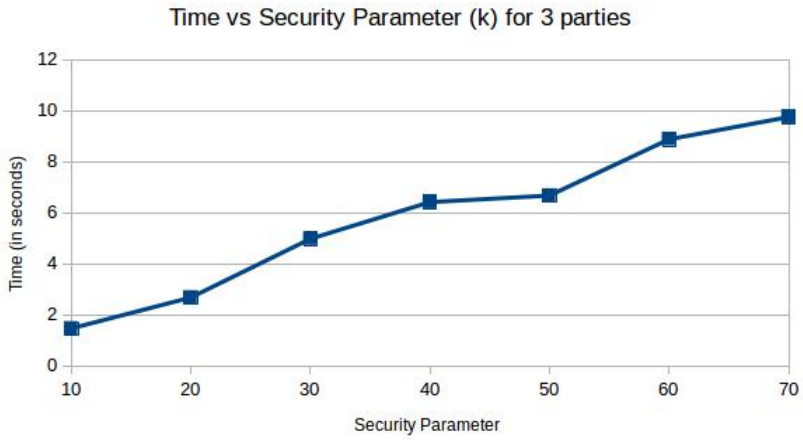
1. A corrupt party is allowed to decide any output that the protocol requires from the party. Since the program is multi threaded we need to synchronize between different threads waiting for input of corrupt party.
2. Such a party is also allowed to automate phases of the protocol such as N ADD Protocol executions, sending the N/2 bits and N/2 AND Protocols executions.
3. A corrupt aggregator is allowed to decide the N/2 indices to be sent to all the parties. It is also given an option to do the inconsistency check and decides what value it wants to output.

5 Experimentation results

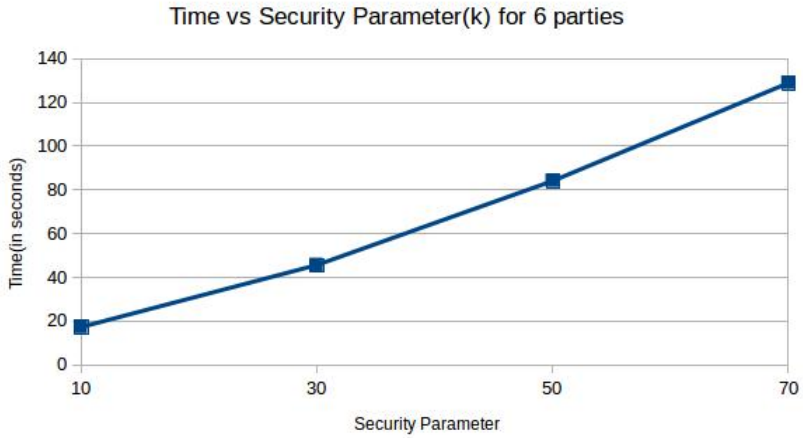
We have taken the security parameter as an input and have set N (Number of repetitions) as $k * 2^{(m+1)}$.

The following are the experimentation graphs obtained for different values of the security parameter and different number of parties.

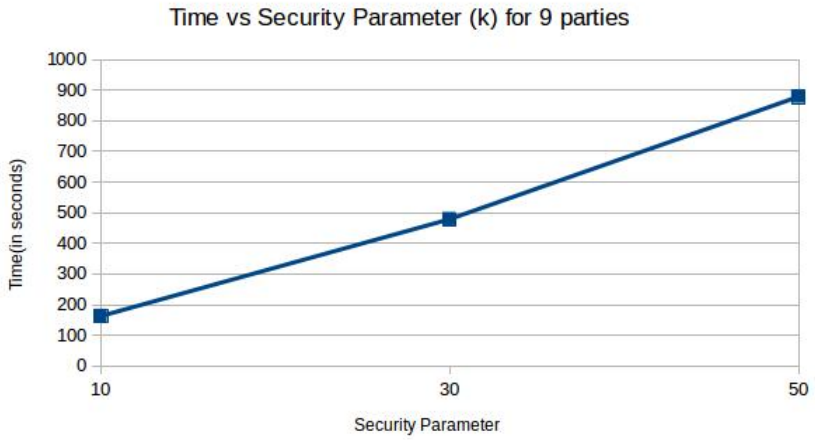
For 3 parties:



For 6 parties:



For 9 parties



5.1 Conclusions

Our system scales linearly with the security parameter in time and behaves reasonably well with up to 9 parties under standard security parameters. We have presented the experimental results obtained in the graphs above. The experiment was run on an SL machine with RAM 8 GB, Quad core with each core having 3.1 GHz processor. We were running all the parties on the same machine and we expect it to scale even better when parties are run on different machines. Since we are using multi threading, the machine utilizes 200 threads with each thread of each party making socket connections so we expect these many sockets to be available.

6 References

1. www.cs.princeton.edu/courses/archive/fall09/cos521/Handouts/probabilityandcomputing.pdf
2. en.wikipedia.org/wiki/Chernoff_bound
3. web.engr.oregonstate.edu/ rosulekm/pubs/dissertation.pdf
4. ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-876j-advanced-topics-in-cryptography-spring-2003/lecture-notes/lec050703.pdf
5. citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.3219&rep=rep1&type=pdf