Implementing Secure Auction Protocol using Mix and Match

Shariful Alam
Computer Science Department
Boise State University
Boise, Idaho, USA
sharifulalam@u.boisestate.edu

Mir Tahsin Imtiaz
Computer Science Department
Boise State University
Boise, Idaho, USA
tahsinimtiaz@u.boisestate.edu

Alejandro Anzola Ávila
Computer Science Department
Boise State University
Boise, Idaho, USA
alejandroanzolaa@u.boisestate.edu

***Abstract*—In Cryptography, privacy is an essential concept. Preserving privacy while doing computation on sensitive values is a desired property of secure multiparty computation. Hence, the computation should not reveal any verifiable information relating to the values. In this project, we implemented a secure, distributed multiparty auction protocol proposed by Jakobsson and Jules in 2000. We implemented distributed and secure operations on ciphertexts to determine the highest bid from the bidders. The secure operations include the implementation of bit by bit ElGamal encryption of client input bids with a joint public key, and a secure plaintext equality test (PET) on the ciphertexts to find whether the underneath plaintexts are equal in a secure way. To determine the winning bid, we implemented a secure greater than function which performs operations on the ciphertexts. Therefore no information regarding the underneath plaintext is revealed. Finally the servers jointly decrypt the winning ciphertext to get back the plaintext (winning bid).**

***Index Terms*—Mix and Match; Millionaire's Problem; Secure Auction; Distributed ElGamal; Plaintext Equality Test; Secure Multi-Party Computation;**

I. Introduction

In cryptography, the goal of secure multiparty computation or privacy-preserving computation is to come up with methods or approaches where all the parties can perform some computation on their inputs together without revealing any verifiable information related to their inputs. Hence the inputs remain secure. In [1], authors investigate various approaches for secure function evaluation and finally consider the inputs from the different parties as ciphertexts. Among various available cryptographic schemes, authors of [1] uses ElGamal cryptosystem [2] to achieve ciphertexts from multiple parties.

In this project, we implement a secure auction protocol described in [1]. We use distributed properties of ElGamal cryptosystem [2] used in [1]. We developed our secure auction system distributedly by implementing the multiple server and client architecture.

A brief working mechanism of our project is as follows, all servers separately generate their own private and public keys and broadcast the joint group public key generated from all the public keys. The bidders use this group public key to encrypt their bids and send them back to the server. After collecting the ciphertexts of all of the bidders, server shuffles the bids randomly; therefore, explicit knowledge of which bid was in which position gets lost. Then inside the server, we perform re-encryption on the encrypted bids; thus, we end up with different encryption of encrypted bids. We use the Plaintext Equality Test (PET) to compare two ciphertexts of inputs bids. Then, we implement a table to find the highest bid securely without revealing their underline plaintext. Thus, we solve the millionaire's problem [3].

In this project, our work is summarized as follows,

- Implement a distributed Client-Server Architecture
- Implement an ElGamal cryptographic scheme
- Solve the millionaire's problem by implementing a secure greater than function. Where we securely compare two ciphertexts to find the biggest one between them.
- To compare two ciphertexts equality, we implement a Plaintext Equality Test (PET)

The remaining of this report is organized as follows. We describe the necessary building block information in Section II. We detail auction protocol implementation in Sections III-A. We then present the limitation of our design in Section IV, and finally we conclude the work in Section V.

II. Building Blocks

A. ElGamal Encryption Scheme

ElGamal is an encryption scheme that uses the intractability of the discrete logarithm problem and the Diffie-Hellman problem.

1) The Discrete Logarithm Problem: Given the finite cyclic group $\mathbb{Z}_p^*$ of order $p-1$ where p is prime and $g \in \mathbb{Z}_p^*$ is a primitive element and another element $\beta \in \mathbb{Z}_p^*$. The DLP is the problem of determining the integer $1 \leq x \leq p-1$ such that:

$$g^x \equiv \beta \pmod{p}$$

2) Diffie-Hellman Problem: Given a finite cyclic group G of order n, a primitive element $g \in G$ and two elements $A = g^a$ and $B = g^b$ in G. The Diffie-Hellman problem is to find the group element g^{ab}.

3) ElGamal Encryption Protocol: It is an asymmetric encryption scheme that chooses a large prime p, a primitive element $g \in \mathbb{Z}_p^*$, a private key $\mathcal{K}_{\text{priv}}$, and a public key $\mathcal{K}_{\text{pub}}$ such that

$$\mathcal{K}_{\text{priv}} \equiv d \in \{2, \ldots, p-2\} \tag{1}$$
$$\beta \equiv g^d \pmod p \tag{2}$$
$$\mathcal{K}_{\text{pub}} = \langle p, g, \beta \rangle \tag{3}$$

The public key is transmitted to any recipient that wishes to send an encrypted message to the one that possesses the private key. Since the private key belongs to one person only, he/she is the only one who is able to decrypt the encrypted message. The encryption is done by choosing an ephemeral key $\mathcal{K}_{\text{E}} \equiv g^i \pmod p$ where $i \in \{2, \ldots, p-2\}$ is chosen randomly, and calculating a masking key $\mathcal{K}_{\text{M}} \equiv \beta^i \pmod p$ to encrypt the message m as:

$$[\![m]\!] \equiv m \cdot \mathcal{K}_{\text{M}} \pmod p$$

The pair $\langle \mathcal{K}_{\text{E}}, [\![m]\!] \rangle$ is sent to the one that possesses the private key, where he can decrypt it by calculating:

$$\mathcal{K}_{\text{M}} \equiv {\mathcal{K}_{\text{E}}}^d \pmod p \tag{4}$$
$$m \equiv [\![m]\!] \cdot {\mathcal{K}_{\text{M}}}^{-1} \pmod p \tag{5}$$

B. Distributed ElGamal

We employed the Distributed ElGamal where each of the servers generate their own public and private keys. The public keys are used to generate a common public key which in then broadcasted to the bidders.

Each of the servers selects $\mathcal{K}_{\text{priv}}(i) \in \mathbb{Z}_p^*$ randomly such that $\mathcal{K}_{\text{priv}}(i) \equiv d_i \in \{2, \ldots, p-2\}$ and computes the public key as:

$$\beta_i \equiv g^{d_i} \pmod p$$

Where the group key is:

$$\beta \equiv \prod_i \beta_i \pmod p \tag{6}$$
$$\beta \equiv g^{\sum_i d_i} \pmod p \tag{7}$$

If any party wishes to send a message m to the network it is encrypted as:

$$[\![m]\!] = \langle X, Y \rangle = \langle m \cdot \beta^r, g^r \rangle$$

where r is a random ephemeral value.

For the total decryption of a ciphertext, it is necessary for every server to partially decrypt the message with its own private key $\mathcal{K}_{\text{priv}}(i)$. The partial decryption leading to the fully decrypted plaintext is as

$$m = \frac{m \cdot \beta^r}{\prod_i Y_i}$$

where Y_i is derived from the private keys of the servers as follows,

$$Y_i \equiv Y^{d_i} \pmod p$$

C. Plaintext Equality Test (PET)

Given two ciphertexts $[\![m]\!]$ and $[\![m']\!]$, the plaintext equality test proves the equality $m = m'$ without the private keys and without revealing any information about the underneath plaintext.

It is done by checking whether dividing one of the two ciphertexts by the other results in the encryption of 1 as follows,

$$\frac{[\![m]\!]}{[\![m']\!]} \equiv \frac{m \cdot \beta}{m' \cdot \beta'} \equiv 1 \pmod p \tag{8}$$
$$\beta = \beta' \tag{9}$$

We can also use PET to check the equality of two different representations or re-encryptions of the same plaintext. If $[\![m]\!]$ and $[\![[\![m]\!]]\!]$ are encryption and re-encryption of the same ciphertext m, then we can apply the logic shown above as follows,

$$\frac{[\![m]\!]}{[\![[\![m]\!]]\!]} \equiv \frac{\langle m \cdot \beta^{r_1}, g^{r_1} \rangle}{\langle m \cdot \beta^{r_1+r_2}, g^{r_1+r_2} \rangle} \equiv \left\langle 1 \cdot \beta^{-r_2}, g^{-r_2} \right\rangle \tag{10}$$

We can decrypt this message in order to check whether it results in the plaintext 1. If it does, the two ciphertext represents the same plaintext.

D. Re-encryption

For a particular ciphertext $\langle X, Y \rangle$, it is possible to derive a completely random re-encryption $\langle X', Y' \rangle$ using only the public key. We can get this re-encryption of the ciphertext by computing,

$$\langle X', Y' \rangle = \langle X, Y \rangle \times \langle \gamma, \delta \rangle \tag{11}$$

Here, $\langle \gamma, \delta \rangle$ represents the encryption of the plaintext 1.

E. Greater than table

We developed a predefined table which is capable of finding the largest encrypted bit between two encrypted bits. By comparing all the bits of the binary representation of two numbers, the table can determine which number is greater than the other. The table holds preconditions for analysing the bits and the corresponding output for that particular scenario. In order to make sure that no servers can define any pattern from the table, the table is re-encrypted and re-shuffled periodically by all the servers. Table I shows the structure of the table described in this report. The "sign" column holds the previous condition from analyzing the previous two bits of the numbers. For example, if the current bit from the number A is 1, the current bit from the number B is 0, and the condition(sign) from the previous bits is that the numbers are equal so far, then the next condition will result in A being greater than B. This example corresponds to the 8th row of Table I. For simplicity of the implementation of the code, we considered the text "1" as the greater than symbol, "-1" as the less than symbol and "0" as the equal symbol.

TABLE I: Greater than comparison table

Ai	Bi	sign	OutA	OutB
0	1	>	>	<
0	0	>	>	<
1	1	>	>	<
1	0	>	>	<
0	1	=	<	>
0	0	=	=	=
1	1	=	=	=
1	0	=	>	<
0	1	<	<	>
0	0	<	<	>
1	1	<	<	>
1	0	<	<	>

F. Distributed Client-Server architecture

In our implementation, we use 3 separate servers and 4 clients. Among the three servers, one works as a coordinator. These remote servers are independent in nature and generate their own ElGamal private and public key pairs and share the public key with the coordinator server; therefore, their private key remains secret. The coordinator server calls these remote servers via Remote Method Invocation (RMI) and gets their public key. Upon receiving the public keys, the coordinator server generates a joint public key using the equation 6.

All the clients connect to the coordinator server. Once all clients are connected, the coordinator server broadcasts the joint public key to the clients. The client nodes, take the input bids from the users, convert them to binary representation and then use ElGamal cryptographic scheme with the joint public key to encrypt the input bid bit by bit. Then the El Gamal ciphertexts along with the ephemeral keys are sent to the server. Then the clients wait for the server response on the winning bid. Figure 1 shows, the high-level architecture of our distributed Client-Server structure.

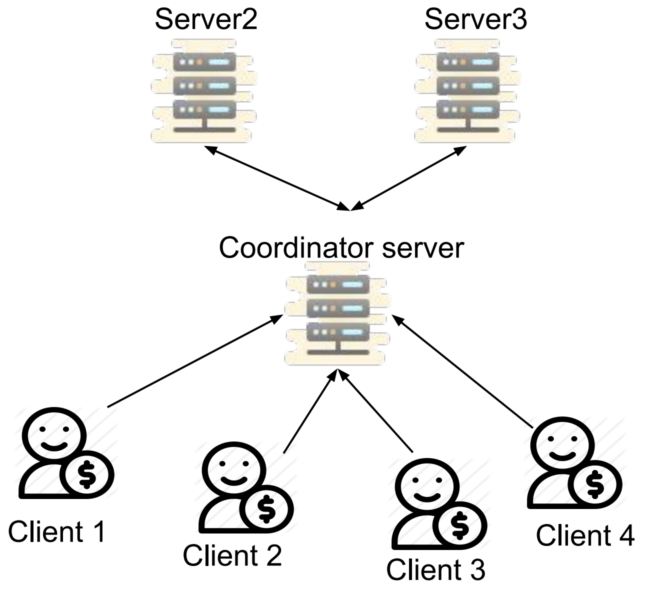

Fig. 1: Distributed Client-Server architecture

III. Implementing Auction Protocol

In the millionaire's problem, the goal is to compute who is the wealthiest between two people without revealing how much money each of them has. In the secure auction protocol, the idea remains the same. Here, multiple parties participate in an auction and submit their bids. The goal is to compute the highest bid securely. Thus no other information is revealed other than the highest bid after jointly decrypting the ciphertext of the winning bid by the participating servers at the end.

A. Auction Protocol

Since we are implementing the auction protocol proposed in [1], we try to follow the same processes that the authors present. Our implementation complies with the proposed protocol as follows,

- **Input protocol:** Each bidder encrypts their input bits using a joint group public key provided by the servers. To encrypt bit 0, we use the ElGamal encryption of plaintext '3' since The plaintext 0 has only a degenerate ciphertext in the ElGamal cryptosystem. And to encrypt bit 1, we use Elgamal encryption of plaintext '1'.
- **Mixing:** By using the joint group public key, all the servers perform random row-wise shuffling and re-encryption on the ciphertexts. We refer to this as blinding.
- **Matching:** After blinding, we use PET to compare two ciphertexts. We determine which one is higher between two ciphertexts using the greater than table. Details in III-D.
- **Output:** Upon detecting the ciphertext with the highest bid, all the servers jointly decrypt the ciphertext to get the final plaintext and send it to all the connected bidders. Details in III-E.

B. Distributed Client-Server

We use socket programming to implement our client-server architecture. To make our servers distributed, we leverage Java's Remote Method Invocation (RMI). While communicating with the remote servers via RMI, our coordinator server works as a client.

C. Re-Encryption

In the re-encryption part, we use the same common joint public key to encrypt the ciphertext again to get a new ciphertext. We accomplished this by multiplying the ciphertext with the encryption of plaintext 1. Equation 11 shows the re-encryption process of a ciphertext $\langle X, Y \rangle$.

D. Find highest bid

The servers evaluate the ciphertexts using the greater than table by a tennis tournament format in order to get the ciphertext of the winning bid. The ciphertexts and the greater than table is repeatedly re-encrypted and re-shuffled by all the servers so that none of the server can define any patterns from the values. After getting the winning cipher text, the servers decrypt the ciphertest together and broadcast the winning bid.

E. Jointly decrypt

After determining the final ciphertext of the highest bid from greater than function, all servers start jointly decrypting this ciphertext. One server individually can't decrypt the ciphertext because of the encryption and re-encryption with the joint public key. Therefore, in order to fully decrypt, the coordinator server sends the ciphertext to server 2. Server 2 does partial decryption with its private key and sends back the partially decrypted ciphertext to the coordinator. The coordinator then sends this partially decrypted ciphertext to server 3. Server 3 does another decryption on the ciphertext and sends back the partially decrypted ciphertext to the coordinator. Upon getting back the ciphertext from server 3, the coordinator server does the final decryption with its private key to get the original plaintext.

F. Send the result

Upon decrypting the ciphertext and getting the plaintext, the coordinator server broadcast this winning bid to all the connected clients.

IV. Limitation

In our implementation, we can provide secure computation on the user bids without revealing any verifiable information. However, our current implementation can handle only 4 clients at a time. Moreover, Coordinator server waits for four clients to submit their ElGamal ciphertext along with the ephemeral keys before it can start working.

V. Conclusion

In this project, we implemented a secure, distributed, multi-party auction protocol using the El Gamal encryption scheme. We used the distributed property of El Gamal to generate the joint public key and use this key to encrypt the input bids bit by bit. We also use this same key for re-encryption. Blinding and re-encryption on the ciphertext make sure that the servers do not have any explicit knowledge about the underneath plaintext of the ciphertext. Hence, this verifies the secure computation of the encrypted bids.

References

[1] M. Jakobsson and A. Juels, "Mix and match: Secure function evaluation via ciphertexts," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2000, pp. 162–177.

[2] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[3] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 1982, pp. 160–164.