University of Bristol

Background Chapter

Secure Two Party Computation

Abstract

We shall be producing an implementation of the Secure Two Party Computation protocol using Cut-and-choose put forward by Prof. Lindell in [1]. We shall also implement other previous protocols that are also secure against Malicious and Covert adversaries for the purposes of comparison.

Author: Nicholas Tutte (nt1124)
Supervisor: Prof. Nigel Smart

1 Background Chapter

1.1 Problem definition

Secure multi-party computation(SMC) is a fundamental problem in Cryptography. We have a set of parties who wish to cooperate to compute some function on inputs distributed across the parties. However, these parties distrust one another and do not wish their inputs to be known to the other parties. We shall be focusing on the case where there are only two parties(S2PC), but most two party approaches can be generalised to the multi-party case.

A commonly used example is the Millionaires problem. A group of rich persons wish to find out who among them is the richest, but do not wish to tell each other how much they are worth. Here the parties are the rich (and somewhat vain) individuals. Their inputs are their net worths and the function will return the identifier of the individual with the highest input. Finally no party should be able to divine anything about anothers inputs, apart from what the parties can infer from their own input and the output.

Whilst this is not exactly an inspiring application, it does explain convey the problem concisely. We shall cover further applications later in 1.4.

Throughout we will assume that we can establish a secure and authenticated channel of communication to all other parties in the computation. That is we assume communications between two parties cannot be eavesdropped upon or altered, and that we can detect attempts to impersonate another party.

1.2 Formal ideal model

There are three main properties that we wish to achieve with any SMP protocol,

- Privacy, the only knowledge parties gain from participating is the output.
- Correctness, the output is indeed that of the intended function.
- Independence of inputs, no party can choose it's inputs as the function of other parties inputs.

In this sense we define the goal of an adversary to compromise any one of these properties.

We compare any protocol to the *ideal* execution, in which the parties submit their inputs to a universally trusted and incorruptible external party who then computes the value of the function and returns the output to the parties. We say that the protocol is secure if no adversary can attack the protocol with more success than they can achieve than the ideal model.

1.3 Security levels

Having established what goals the adversary wishes to achieve and how we can measure if said adversary has a valid attack, we next deal with the question of the capabilities of the adversary. We use three main models to describe the threat level of the adversary.

1.3.1 Semi-honest Adversary

The Semi Honest(SH) adversary is the weakest adversary, and requires the strongest limitations on their capabilities. The SH adversary has also been referred to as "honest but curious", because in this case the adversary is not allowed to deviate from the established protocol in any way (i.e. they play fair/are honest), but at the same time they will do their best to compromise one of the aforementioned security properties by examining the data they have legitimate access to. This is in some ways analogous to the classic "passive" adversary.

1.3.2 Malicious Model

The Malicious adversary is allowed to employ any polynomial time strategy and is not bounded by the protocol (they can run arbitrary code instead), but we assume that for the adversary to be successful they must be able to guarantee they will not be detected. This is in some ways analogous to the classic "active" adversary.

1.3.3 Covert Model

The Covert adversary model is very similar to the Malicious model, again bounded by polynomial time with freedom to ignore the protocol, however in this case the adversary is willing to accept a certain probability of detection. In this case the adversary is playing a cost-benefit trade off and to achieve security we need ensure the adversary's attack is detected with a high enough probability that the adversary will not consider it worth the risk.

We call the probability that such an adversary will be caught the "deterrent probability", usually denoted using ϵ. Often protocols providing security against Covert adversaries take a Security parameter which varies the probability of detecting cheating.

1.4 Applications

At first it might appear that SMC lacks applications beyond those like the trivial example provided in 1.1. In fact as this is often the first example given many dismiss SMC as of limited usefulness or as a pointless toy. Here we shall provide a number of other applications either already in use or in development.

1.4.1 Secret Auctions - Danish Beets

In Denmark a significant number of farmers are contracted to grow sugar beets for Danisco (a Danish bio-products company). Farmers can trade contracts amongst themselves (effectively sub-contracting the production of the beets), bidding for these sub-contracts is done via a "double auction". Farmers do not wish to expose their bids as this gives information about their financial state to Danisco and so refused to accept Danisco as a trusted auctioneer, similarly all other parties (e.g. Farmer union) already involved are in some way disqualified. Rather than rely on a completely uninvolved party like an external auction house the farmers use an SMC-based approached described in [2]. Since 2008 this auction has been run multiple times

As far as team behind this auction are aware this is the first large scale application of SMC to a real world problem, this application example in particular is important as it is a concrete practical example of SMC being used to solve a problem demonstrating this is not just a Cryptological gimmick.

1.4.2 Database queries

Imagine the case where one party holds a database, and the other wishes to perform a query upon this database. However, for whatever reason the querying party does not wish to reveal what their query is, nor does the holder of the database wish to give the database to the querier. This particular problem and related problems have attracted significant attention in the academic community.

1.4.3 Distributed secrets

Consider the growing use of physical tokens in user authentication, e.g. the RSA SecurID. When each SecurID token is activated the seed generated for that token is loaded to the relevant server (RSA Authentication Manager), then when authentication is needed both the server and the token compute 'something' using the aforementioned seed. However, this means that in the event of the server being breached and the seed being compromised the physical tokens will need to be replaced. Clearly this is undesirable, being both expensive both in terms of up front costs and reputation.

In the above scenario we clearly need to store the secret(the seed) somewhere, but if we can split the seed across multiple servers and then get the servers to perform the computation as a SMC problem (where each server's input is their share of the secret, the output the value to compare to the token's input) then we can increase the cost to an attacker, as they will now have to compromise multiple servers. Such a service is in development by Dyadic Security (full disclosure, my supervisor Nigel Smart is a co-founder of Dyadic).

1.4.4 AES Encryption

A classic test/benchmark for any general S2PC protocol is to perform an AES encryption where the message and key are held by two parties, so P_A can input a message to be encrypted, P_B without knowing the message will encrypt it using it's key, returning the encrypted ciphertext to P_A and at no point did P_A know what key was used, nor P_B what the plaintext was.

1.5 Yao Garbled Circuits

1.5.1 Overview

Yao garbled circuits are one of the primary avenues of research into Secure multi-party computation. Yao proposed in [8] that the two parties are designated the Builder and the Executor. The Builder then constructs a circuit representing the function the parties wish to compute, this circuit is "garbled" in such a way that it can stil be executed.

This garbled circuit, hardcoded with the Builders input data, is sent to the Executing party who then obtains the data representing their input form the Builder via Oblivious Transfer. The Executor then evaluates the circuit and obtains the output of the function.

1.5.2 Details

As noted above we first represent the function to compute as a binary circuit. Denote the two parties as P_1 and P_2, we will assume WLOG that P_1 is constructing the circuit whilst P_2 is executing. Now take a single gate of this circuit with two input wires and a single output wire. Denote the gate a G_1 and the input wires as w_1 and w_2, let b_i be the value of w_i where $b_i \in \{0, 1\}$. Here we will take the case where w_i is an input wire for which P_i provides the value. Define the output value of the gate to be $G(b_1, b_2) \in \{0, 1\}$. Next we garble this gate.

P_1 (the circuit building party) garbles each wire by selecting two random keys of length l, for the wire w_i call these keys k_i^0 and k_i^1. The length of these keys (l) can be considered our security parameter, and should correspond to the length of the key needed for the symmetric encryption scheme we'll be using later. Further P_1 also generates a random permuation $\pi_i \in \{0, 1\}$ for each w_i, we define $c_i = \pi_i(b_i)$. The garbled value of the i^{th} wire is then $k_i^{b_i} \| c_i$, we then represent our garbled truth table for the gate with the table indexed by the values for the c_1 and c_2.

$$c_1, c_2 : E_{k_1^{b_1}, k_2^{b_2}}(k_3^{G(b_1,b_2)} \| c_3)$$

Where $E_{k_i,k_j}(m)$ is some encryption function taking the keys k_i and k_j and the plaintext m. Since the advent of AES-NI and the cheapness of using AES we will use AES with 128 bit keys to make this function. Suppose that $AES_k(m)$ denotes the AES encryption of the plaintext m under the 128 bit key k. We define E_K (and it's inverse D_K) as follows,

$$E_K(m) = AES_{k_n}(AES_{k_{n-1}}(...AES_{k_1}(m)...))\text{where } K = \{k_1, ..., k_n\}$$
$$D_K(m) = AES^{-1}_{k_1}(AES^{-1}_{k_2}(...AES^{-1}_{k_n}(m)...))\text{where } K = \{k_1, ..., k_n\}$$

We extend both the encryption function and the truth table this to a gate with more than two inputs in the obvious fashion.

Then P_1 (the builder of the circuit) sends this garbled version of the circuit to P_2 (the executor of the circuit). P_1 should send the garbling key for it's input bit ($k_1^{b_1}$), the full encrypted truth table and $c_1 = \pi(b_1)$. Then P_2 needs to get $k_2^{b_2} \| c_2$ from P_2 without revealing the value of b_2. This is done by an Oblivious Transfer where P_1 inputs k_2^0 and k_2^1 and P_2 inputs b_2. P_2 receives the output $k_2^{b_2} \| c_2$ from the OT and learns nothing about $k_2^{(b_2 \oplus 1)}$, P_1 gets no output and learns nothing about the value of b_2.

P_2 can then look up the entry in the encrypted truth table indexed by c_1 and c_2 and decrypt it with using $k_1^{b_1}$ and $k_2^{b_2}$. This will give P_2 a value for $k_3^{G(b_1,b_2)} \| c_3$. Then by using π_3^{-1}, P_2 can extract a value for $G(b_1, b_2)$.

This can be extended to a full circuit, the input wires belonging to the circuits builder are hard coded and their garble keys and permutated values are sent to the executor. The values for the input wires belonging to the executor are obtained by the executor via Oblivious transfer with the builder. The executor is only given the permutations for the output wires, and therefore the intermediate values are protected.

1.5.3 Security of Yao Garbled Circuits

A naive implementation of a protocol using Yao Garbled Circuits only provides Semi-honest security. For a formal proof of Semi-honest security see [3], we shall briefly give an intuitional explaination of why naive Yao Garbled Circuits are not secure in the presence of Malicious or Covert adversaries.

Consider the case where the circuit building party is Malicious, at no point does the executing party verify that the garbled circuit provided by the builder actually computes the function the builder claims it does. This clearly breaks the Correctness requirement, but also opens up an attack on the Privacy and Independence of inputs.

There is nothing to stop the Builder providing a circuit that outputs the input of the Executor in a way that the Builder can recover, as such the Privacy of the Executor's inputs fails in the presence of a Malicious Builder. Furthermore, the Builder could provide a circuit where its input is a function of the Executor's input, thus defeating the Independence of inputs requirement.

1.5.4 Security against Malicious and Covert Adversaries

Several extensions of Yao's original protocol have been proposed in order to achieve security against Malicious and Covert adversaries. Mostly depending on an appraoch dubbed "cut and choose" which provides statistical security (detects cheating with a certain probability). This relates to the old solution to dividing a cake fairly, one party cuts the cake in two, then the other party chooses a slice. In our case the Builder builds s many garbled circuits and sends them to the Executor. Each of these circuits is chosen with probability $\frac{1}{2}$ as a check circuit that will be evaluated to test if they yield the correct result.

If all check circuits pass then the Executor evaluates the remaining circuits as usual and returns the most common output. If one or more circuits produces an incorrect result the this indicates cheating, furthermore if any check circuits fail during evaluation this is also taken to indicate cheating.

This means s acts as a security parameter and the probability of detecting cheating is expressed in terms of s. For example the protocol proposed in [1] detects cheating with probability 2^s.

1.6 Oblivious Transfer

1.6.1 Introduction to Oblivious Transfer

Oblivious Transfers protocols allow for one party(called the receiver) to get exactly one out-of two (and can be extended to out-of-n) values from another party (called the Sender). The receiver is oblivious to the other value(s), and the Sender is oblivious to which value the receiver received.

Oblivious Transfers (OTs) were first suggested by Rabin in [4]. We formally define the functionality of a 1 out of 2 OT protocol in Figure 1. Oblivious Transfers are vital to Yao Garbled Circuits, used to give the circuit executor the garble keys representing its inputs, without telling the circuit builder what those inputs were.

Receiver	**Sender**
Inputs : $b \in \{0,1\}$	Inputs : X_1, X_2
Outputs : X_b	Outputs : $\emptyset$

Figure 1: Formal definition of the functionality of a one out of two OT protocol.

The security of Oblivious Transfers is defined in a similar way to that of SMPC, adversaries have three main levels of power, Semi-honest, Malicious and Covert. These adversaries are analogues to the adversaries we defined in 1.3.

A protocol is considered secure Semi-honest adversaries if it a semi-honest adversary in the sender role cannot learn anything about which value the receiver requested, nor can a semi-honest adversary in the role of the receiver learn anything about values other than the one it requested. The protocol being secure against Malicious or Covert adversaries is defined by the naive extension of the Semi-honest case.

1.6.2 Even-Goldreich-Lempel SH OT Protocol

Here we look at a Semi-honest protocol for 1-out-of-2 case based on asymmetric encryption, taken from [5] which is itself based on the work done in [6].

Receiver	**Sender**
Inputs : $b \in \{0, 1\}$	Inputs : X_1, X_2
Outputs : X_b	Outputs : $\emptyset$

- **Receiver:** Generates a public/private key pair (E, D), where E is the public key.

- **Receiver:** Sets $PK_b = E$, choose PK_{1-b} at random from the same distribution as the public keys. Send PK_0 and PK_1 to the Sender.

- **Sender:** Encrypt X_0 using PK_0 as C_0 and encrypt X_1 using PK_1 as C_1. Send C_0 and C_1 to the Receiver.

- **Receiver:** Receives C_0 and C_1, then decrypt C_b using D. Output this decrypted value.

This protocol assumes that it is possible to generate randomly in the *form* the public key without revealing information about the private key. Given this assumption this protocol is clearly only Semi-honest as the Sender must trust the Receiver that PK_{1-b} was indeed generated randomly and that the Receiver did not generate a decryption key. However, so long as a both parties are following the protocol, neither party can garner more information than they are supposed to be allowed to.

1.7 Oblivious Transfer using dual-mode cryptosystem

The basis of the Oblivious transfer protocol we shall be using comes from [7], in particular we shall be using the realisation of the dual-mode cryptosystem based on Decisional Diffie-Hellman problem (effectively a slightly modified Elgamal). Whilst I shall not go into depth on this protocol we shall give a broad overview of the dual-mode cryptosystem.

The Dual-mode cryptosystem has two modes (*messy* and *decryption*), selected during the setup, a Common Reference String(CRS) is also generated at setup. The receiver uses their input bit and the CRS to generate a “base” public key and private key. The public key is then sent to the Sender, who computes two public keys derived from the base public key and the CRS. Each of the Sender’s input values are then encrypted with one of these keys and the resulting ciphertexts sent to the Receiver. Finally the Receiver uses its private key in order to decrypt the ciphertext relating to its input bit.

The properties of the dual-mode cryptosystem ensure that the Receiver can only decrypt one of the values. When generating the “base” keys the key generation takes as a parameter a decryptable branch ($\sigma \in \{0, 1\}$), and the resulting private key is associated with that branch. When encrypting using the public key we specify a branch (say $b \in \{0, 1\}$) on which to encrypt, the resulting ciphertext can only be decrypted if $\sigma = b$. Encrypting on the other branch

1.8 Progress so far

So far (02/02/2015) we have implemented the naive Yao garbled circuit using 1.7 for the Oblivious Transfer. We use the circuits provided on Prof. Smart’s website [9] and use the AES non-expanded circuit provided as our main benchmark. Running on the Bristol

Crypto groups test machines Diffie and Hellman (Get specs here) my implementation can perform a single AES-128 block encryption in $\sim$

References

[1] Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries,
Lindell (2014).

[2] Multiparty Computation Goes Live,
Peter Bogetoft et al. (2008)

[3] A Proof of Security of Yao's Protocol for Two-Party Computation,
Lindell and Pinkas (2006).

[4] How to Exchange Secrets with Oblivious Transfer,
Michael O. Rabin (1981)

[5] Secure Computation Lecture Series,
Lecture 5 - Oblivious Transfer,
Pinkas (2014)

[6] A randomized protocol for signing contracts,
Even, Goldreich and Lempel (1985)

[7] A Framework for Efficient and Composable Oblivious Transfer,
Peikert, Vaikuntanathan and Waters (2008)

[8] How to Generate and Exchange Secrets, A. Yao (1986).

[9] http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/, Tillich and Smart.