University of Bristol

4^{th} Year Thesis

Secure Two Party Computation

A comparison of recent protocols

Abstract

We shall be producing implementations of several proposed Secure Two Party Computation protocols, several of which till now unimplemented, with a view to comparison. Until now we have had only theoretical comparisons of these protocols, as such this has made it difficult to know which approach is the most promising.

In particular we have implemented the protocols described in [2], [1] and [3]. Furthermore we have varied the sub-protocol used in our implementation of [2] to compare performance therein.

Author: Nicholas Tutte (nt1124)

Supervisor: Prof. Nigel Smart

1 Introduction

1.1 Problem definition

Secure multi-party computation(SMC) is a long standing 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(S2C), 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 individuals, each party's inputs is their net worth and the function will return the identifier of the individual with the highest input and no party should be able to divine anything about another party's inputs, apart from what can be inferred from their own input and the output.

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.

Some time should be taken at this point to note also that we are pursuing a generalised solution to this problem. Many SMC protocols exist that are designed for a specific problem, whilst these will tend to be more efficient than any general solution applied to the same problem they are limited to one problem. We will be focusing on general protocols that can be applied to any computation.

1.2 The ideal model

There are three main properties that we wish to achieve with any SMC 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 via secure channels. This trusted party 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 the goals of the adversary and how we can measure if said adversary has a valid attack, we next deal with 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 adversary is the weakest adversary, with very limited capabilities. The Semi-honest 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 (i.e. they 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), furthermore the Malicious adversary does not care if it is caught cheating so long as it achieves its goal in the process. 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 adverse to being caught cheating and is therefore slightly weaker than the Malicious adversary. A Covert adversary will accept a certain probability of detection, this probability represents the point at which the expected payoff/benefit of cheating without detection outweighs the expected cost/punishment for getting caught, effectively a game theory problem.

We call the probability that a Covert 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 real world application. 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 (an expensive option) the farmers use an SMC-based approached described in [4]. Since 2008 this auction has been run multiple times

As far as team behind this auction are aware this was 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 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 clean up 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 Prof. Nigel Smart is a co-founder of Dyadic).

1.4.3 PROCEED - Computation on encrypted data

Recently US Defence Advanced Research Projects Agency (DARPA) has been running a project called PROCEED. The eventual goal being the ability to efficiently perform computations on encrypted data without knowledge of the data. This could be used by companies such as Google to continue to provide services requiring computation on personal data without intruding on the privacy on their users.

1.5 Yao Garbled Circuits

1.5.1 Overview

Yao garbled circuits are one of the primary avenues of research into Secure multiparty computation. Yao first proposed garbled circuits [12]. 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 still be executed.

This garbled circuit, hardcoded with the Builders input data, is sent to the Executing party who then obtains the data representing its input from 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 denote the party building the circuit by P_1 and the executing party by P_2.

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 and let w_3 be the output wire. Let $b_i \in \{0,1\}$ be the value of w_i. 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 $G(b_1, b_2) \in \{0,1\}$. We now garble this gate in order to obscure the inputs and outputs.

P_1 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 a 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 permutation $\pi_i \in \{0,1\}$ 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 and $AES_k^{-1}(c)$ denotes the decryption of ciphertext c under key k. We define E_K (and it's inverse D_K) as follows,

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

This is the intuitive extension of AES to multiple keys, chaining the encryption under all of the keys in a set order.

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 $k_2^{(1-b_2)}$, 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 using $D_{k_1^{b_1},k_2^{b_2}}(\cdot)$. 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 permuted 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 wire bit values are protected.

1.5.3 Security of Yao Garbled Circuits

A naive implementation of a protocol using Yao Garbled Circuits provides only Semi-honest security. For a formal proof of Semi-honest security see [5], we shall briefly give an intuitive explanation of why naive Yao Garbled Circuits are not secure in the presence of Malicious or Covert adversaries.

Consider the case where P_1 is Malicious, at no point does a naive P_2 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. Alternatively, 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 approach 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. A subset of these circuits are chosen to be opened for the purpose of checking if they are correct.

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 cheating in the protocol proposed in [2]. goes undetected at probability 2^{-s}

This Cut and Choose approach creates additional problems to be solved. For example whilst evaluating the many circuits we must now also ensure that both parties' inputs are consistent (same inputs to each circuit) else they might be able learn many outputs, each revealing something they should not have been able to discover.

In [11] the example is given of computing the inner product of two binary strings, in this situation the Executing party could give many different inputs each with a single bit set to 1. The output of the circuit would then give the Executor the value of the Builder's input bit corresponding to the high bit in the Executor's input.

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 k-out-of-n for $k < n$) 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 [6]. 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 SMC, the focus is on Semi-honest(passive) and Malicious(active) adversaries. Security against these adversaries is usually either computational or statistical.

A protocol is considered secure with regards to Semi-honest adversaries if neither 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 adversaries is defined by the obvious extension of the Semi-honest case.

Usually proof of the security of an OT is carried out via *simulation*, proving that the protocol under consideration is equivalent to the ideal model with a trusted third party taking both parties inputs and sending the proper outputs to each party.

1.6.2 Even-Goldreich-Lempel SH Protocol

Here we look at a Semi-honest protocol for 1-out-of-2 case based on asymmetric encryption, taken from [7] which is itself based on the work done in [8]. See Figure 2 for the abstracted protocol.

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.

Figure 2: The abstracted Even-Goldreich-Lempel protocol.

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.

So long as a both parties are following the protocol, neither party can garner more information than they are supposed to be allowed to. Whilst this on its own is of limited real world application (we require a certain level of trust in someone we explicitly don't trust) some work has focused on taking Semi-honest protocols and extending them to be secure in the presence of Malicious adversaries via techniques like coin-flipping and zero knowledge proofs.

1.7 Oblivious Transfer using dual-mode cryptosystem

The basis of the Oblivious transfer protocol we shall be using comes from [9], in particular we shall be using the realisation of the dual-mode cryptosystem based on Decisional Diffie-Hellman problem. 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 we specify a branch (say $b \in \{0, 1\}$) which defines which public key derived from the base key to use, the resulting ciphertext can only be decrypted if $\sigma = b$.

Further properties of the cryptosystem are that the first outputs of the setup of each modes are indistinguishable from one another, that use of a trapdoor value for the messy mode will reveal the encryptable branch of a public key and that a trapdoor value for the decryptable mode will allow the generation of keys decryptable on both branches.

It should be noted that the security of their protocol depends on a trusted setup of the CRS. Peikert et al. consider this a reasonable assumption and in at least the DDH based realisation of the dual-mode cryptosystem it is possible to have both parties take part in the CRS setup in order to assure both parties neither has access to the trapdoors.

2 Implementation Details

2.1 Purpose of Implementation

It should be made abundantly clear that the implementation provided is not intended for real world use, instead it is for the purposes of comparing the performance of the protocols under consideration. For example we have not established a secure connection between the two parties.

Where possible I have implemented everything myself and reused the same code across protocols, rather than using available libraries. This maintains a consistent quality of implementation, using libraries where appropriate would improve the quality of the implementation it would do so in an uneven manner as many areas cannot be done using a library. This could potentially give one protocol an unfair advantage over another leading to skewed results.

3 Results

3.1 Experiments

3.2 Measurement metrics

3.3 Testing Environment

All tests were carried out between two test machines each with an i7-3770S CPU clocked at 3.10 GHz.

3.4 Benchmarks

Here I give some benchmarks of key parts of my implementation such as communication, ECC encryption and circuit evaluation.

References

[1] Y. Lindell, B. Pinkas. *Secure two-party computation via cut-and-choose oblivious transfer* Journal of Cryptology 25(4), 680–722 (2012).

[2] Y. Lindell, *Fast cut-and-choose based protocols for malicious and covert adversaries*, R. Canetti, J.A. Garay, (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 1–17. Springer, Heidelberg (2013).

[3] Y. Huang, J. Katz, D. Evans. *Efficient Secure Two-Party Computation Using Symmetric Cut-and-Choose*,
2013.

[4] P. Bogetoft, D. Christensen, I. Damgård et al,
Secure Multiparty Computation Goes Live, (2008)

[5] Y. Lindell, B. Pinkas. *A proof of security of Yao's protocol for two-party computation.* Journal of Cryptology 22(2), 161–188 (2009)

[6] How to Exchange Secrets with Oblivious Transfer,
M. Rabin (1981)

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

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

[9] C. Peikert, V. Vaikuntanathan, B. Waters. *A framework for efficient and composable oblivious transfer.* In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008)

[10] Y. Lindell and B. Pinkas. *Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer.* In TCC 2011, Springer (LNCS 6597), pages 329–346, 2011

[11] An Efficient Protocol for Secure Two-Party Compuation in the Presence of Malicious Adversaries, Y. Lindell and B. Pinkas (2007).

[12] A. Yao. How to Generate and Exchange Secrets. In 27th FOCS, pages 162–167, 1986.

[13] Bristol Cryptography Group. Circuits of Basic Functions Suitable For MPC and FHE. http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/.