Dissertation Type: Research - GG1K

DEPARTMENT OF COMPUTER SCIENCE

Secure Two Party Computation
A practical comparison of recent protocols

Nicholas Tutte

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Thursday 26$^{\text{th}}$ March, 2015

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of **GG1K** in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Nicholas Tutte, Thursday 26$^{\text{th}}$ March, 2015

Prelude

Executive Summary

Abstract

We present implementations of several recently proposed Secure Two Party Computation protocols and perform experiments for the purpose of comparing their performance across a range of computations. Until now we have had only theoretical comparisons of these protocols, making it difficult to know which approach is the most promising.

In particular we have implemented the protocols described in [2], [4] and [5] and additionally we experiment with modifying [4] to use [5] instead of [2] as a sub-protocol.

Summary of Achievements

- I implemented the protocols described in [2, 4], to the best of my knowledge these are the first implementations of these protocols.
- I experimented with modifying the sub-protocol used for the Secure computation to detect cheating in [4], exchanging the use of [2] for [5].
- I have run practical comparisons of all the implemented protocols on a variety of circuits/-computations and provided some analysis of the results.

Supporting Technologies

- Unless otherwise stated all tests have been run upon the Bristol Cryptography Group's Diffie and Hellman machines. These machines are identical and have dedicated network cards for communications between each other.
- All code is in either C or C++, using the OpenMP library for parallelism in the shared memory paradigm. Furthermore AES-NI support is enabled.
- Extensive use has been made of the GNU Multi Precision Arithmetic Library.
- The net code was provided by my supervisor Prof. Nigel Smart.
- The AES implementation I use was mostly provided by my supervisor Prof. Nigel Smart (coded by Dr. Dan Page). Though I have extended this as it did not provide non-AES-NI decryption.
- The SHA-2 implementation used was taken from [INSERT CITATION HERE].

Contents

Chapter 1

Introduction

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 reveal their inputs to the other parties. Using SMC we can perform the desired computation without any party ever knowing the other's inputs.

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 greatest input. Additionally, at the end of the computation 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.

For many years Yao's protocol [12] has been the most attractive avenue of theoretical research, mainly due to its conceptual simplicity and constant round nature. In particular recent work has endeavoured to produce variants of Yao's protocol that can provide security in the presence of malicious adversaries ([1], [2], [4], [5], [9], [10], [11]) and to improve the efficiency of the original protocol itself ([7], [8]).

Our contributions are as follows,

- To the best of our knowledge we provide the first implementations of the protocols of [2] and [4].
- We put forward and implement a modification of [4] using [5] for the sub-computation rather than [2] as originally proposed.
- We measure the performance of each protocol on several of the classic SMC benchmark computations.

Chapter 2

Background to Secure Multiparty Computation

2.1 Security Properties

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.

Informally 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. More formally a protocol is secure if it

2.2 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 capability of the adversary.

2.2.1 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.

Example

The Semi-Honest model seems at first to be somewhat pointless, after all we are only assured of security if the adversary follows the rules of the protocol.

2.2.2 Malicious Adversary

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.

Example

2.2.3 Covert Adversary

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 [14].

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.

Example

This model is usually appropriate when there are tangible consequences to a party being caught cheating. For example consider the case where a group of companies are required to report on certain business dealing

2.3 Applications of SMC

Here we take time to motivate the study of SMC by giving several actual or proposed applications.

2.3.1 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 [6]. Since 2008 this auction has been ran 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.

2.3.2 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).

2.3.3 PROCEED - Computation on encrypted data

Recently US Defence Advanced Research Projects Agency (DARPA) ended a programme 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.

2.4 Yao's Protocol

2.4.1 Overview

Yao garbled circuits are one of the primary avenues of research into Secure multi-party 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 Builder's input data, is sent to the Executing party who then obtains the data representing its input from the Builder via Oblivious Transfer (for details on OT see Section 3). The Executor then evaluates the circuit and obtains the output of the function.

2.4.2 Yao Garbled Circuits

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 (see Section3) 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.

2.4.3 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 [13], 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.

2.4.4 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 in the classic protocol returns the most common output. If one or more circuits produces an incorrect result 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 [4] goes undetected with probability 2^{-s}.

This Cut and Choose seems very simple conceptually, but creates several subtle new 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 [1] 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.

Chapter 3

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 [15]. We define the functionality of a 1-out-of-2 OT protocol in Figure 3.1. Oblivious Transfers are vital to Yao Garbled Circuits, used to give the circuit Executor data it needs to evaluate the circuit under their input without revealing to 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 3.1: 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.

We primarily use OTs based on the Peikert-Vaikuntanathan-Waters OT (PVW-OT) from [18] or more precisely the modifications of the PVW-OT suggested in [2] and expanded on in [4]. However, we also use the Naor-Pinkas (NP-OT) from [19] for the protocol in [5].

3.1 Oblivious Transfer using dual-mode cryptosystem

The basis of the Oblivious transfer protocol we shall be using comes from [18], 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.

Chapter 4

Protocols

4.1 Lindell and Pinkas 2011

The protocol proposed in [2] is a significant improvement on their previous proposal [1]. Firstly his protocol gives an improved deterrent probability of $\epsilon = 1 - 2^{-0.311s}$, further the work in [3] showed how to achieve a slightly improve deterrent probability of $\epsilon = 1 - 2^{-0.32s}$. Secondly it removes the need for the very large number of commitments entails in [1] and thirdly it does not require the preprocessing of the circuit that inflates the number of input wires for the Executor and thus the number of Oblivious transfers needed.

The main new idea in this protocol is a modification of the PVW-OT from [18]. We refer to this new OT as the "Cut and Choose OT"(CnC OT), the Receiver generates a random $J \subset [1, ..., s]$ during the setup such that $|J| = \frac{s}{2}$, this set represents a subset of the s circuits to be opened.

This set J is then used to generate s many CRSs, each CRS to be used for a different circuit that the Builder sent. For the j^{th} CRS if $j \in J$ then an OT using this CRS will reveal both values input by the sender rather than the usual 1-out-of-2 values, otherwise the usual OT functionality holds.

When running the OT to get its garbled inputs the parties use the j^{th} CRS for all OTs concerning the j^{th} circuit. Clearly this requires that the Executing party be able to prove that only $\frac{s}{2}$ many of the CRSs allow the recovery of both inputs. This is achieved via a Zero Knowledge Proof detailed in Appendix B of [2].

4.2 Lindell 2013

In [4] Lindell proposed an further improvement on his work with Pinkas in [2]. The primary breakthroughs of this paper are the removal of the requirement that exactly half the circuits are selected as check circuits, instead each circuit is selected with probability $\frac{1}{2}$. A further breakthrough [detail the secure computation to detect cheating here]...

4.3 Huang, Katz and Evans 2013

Concurrently to Lindell's work in [4] Huang, Katz and Evans produced a protocol also based along the same cut and choose paradigm, however in their approach the parties symmetrically generate a set of circuits and then evaluate each others circuits.

Chapter 5

Implementation Details

5.1 Purpose of Implementation

It should be made abundantly clear that the implementation provided is not intended for real world use with actual confidentiality on the line, instead it is for the purposes of comparing the performance of the protocols under consideration.

Whilst the protocols have been implemented faithfully some of the lower level details not relevant to a comparison of the protocols are ignored, for example we do not established a secure connection between the two parties.

Where possible we 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.

Yao Garbled Circuits implementation

Elliptic Curve implementation

Throughout unless otherwise stated we have worked in Elliptic Curve groups, in particular on the curve specified in

Secret Sharing and Multi-precision Polynomials

For the Zero Knowledge Proof of Knowledge specified in Appendix B [2] we need a Secret Sharing Scheme. I have implemented Shamir's Secret Scheme. This scheme is based on how many points are needed to uniquely define a polynomial curve.

Consider a polynomial K of degree n. Then we can denote this polynomial as $K = \sum i = 0na_i \cdot x^i$. Any such polynomial of degree n can be uniquely defined given $n + 1$ (or more) points on the curve, given fewer points then this

Peikert, Vaikuntanathan and Waters OT

.

Other - AES, SHA-2

.

Chapter 6

Experiments

We shall be using the circuits provided in [20] for our experiments with varying randomised inputs, in particular we shall consider

- AES (expanded and non-expanded),
- 32-bit Addition,
- 32-bit Multiplication,
- SHA-256 hashing.

6.1 Measurement metrics

We shall be focusing on three main metrics for measuring performance of the protocols for both parties, namely CPU time used, wall clock time used and data sent.

We shall break these metrics down further by part of the protocol.

6.2 Testing Environment

All tests were carried out between two test machines each with an i7-3770S CPU clocked at 3.10 GHz with 8096 KB of cache and 32 GB of RAM. These machines possess dedicated network card for communications with the other member of the pair. Compilation was performed with g++ version 4.4.7.

6.3 Benchmarks

Here I give some benchmarks of key components in my implementation such as communication, ECC encryption and circuit evaluation. I include these measurements so that others intending to implement these protocols with more efficient (e.g. library supplied) components can get a rough idea of what performance improvement they can expect.

Chapter 7

Conclusions

Bibliography

[1] Y. Lindell and B. Pinkas. *An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries.* To appear in the Journal of Cryptology. (Extended abstract appeared in EUROCRYPT 2007, Springer (LNCS 4515), pages 52–78, 2007.)

[2] 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

[3] A. Shelat, C.H. Shen. Two-Output Secure Computation with Malicious Adversaries. In EUROCRYPT 2011, Springer (LNCS 6632), pages 386–405, 2011.

[4] 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).

[5] Y. Huang, J. Katz, D. Evans. *Efficient Secure Two-Party Computation Using Symmetric Cut-and-Choose*, In 33rd International Cryptology Conference (CRYPTO 2013), 2013.

[6] P. Bogetoft, D. Christensen, I. Damgård et al. *Secure Multiparty Computation Goes Live*, In Financial Cryptography and Data Security 2009, Springer LNCS 5628, pages 325-343, 2009.

[7] Benny Pinkas, Thomas Schneider, Nigel P. Smart and Stephen C. Williams. *Secure Two-Party Computation is Practical*, ASIACRYPT 2009, December 6-10, 2009.

[8] V. Kolesnikov and T. Schneider. *Improved garbled circuit: Free XOR gates and applications.* In Automata, Languages and Programming – ICALP 2008, Springer-Verlag (LNCS 5126), pages 486 - 498, 2008.

[9] S. Jarecki and V. Shmatikov. *Efficient Two-Party Secure Computation on Committed Inputs.* In EUROCRYPT 2007, Springer (LNCS 4515), pages 97 - 114, 2007.

[10] J. Nielsen and C. Orlandi. *LEGO for Two-Party Secure Computation.* In TCC 2009, Springer (LNCS 5444), pages 368 - 386, 2009.

[11] T. Frederiksen, T. Jakobsen, J. Nielsen, et al. *MiniLEGO: Efficient Secure Two-Party Computation from General Assumptions*, In Advances in Cryptology - EUROCRYPT 2013, Springer (LNCS 7881), pages 537 - 556, 2013.

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

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

[14] I. Abraham, D. Dolev, R. Gonen and J. Halpern. *Distributed Computing Meets Game Theory: Robust Mechanisms for Rational Secret Sharing and Multiparty Computation*, Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, pages 53 - 62, 2006.

[15] M. Rabin. *How to exchange secrets with oblivious transfer.* Technical Report, TR-81, Aiken Computation Lab, Harvard University, 1981.

[16] B. Pinkas. *Secure Computation Lecture Series*, Lecture 5 - Oblivious Transfer, 2014.

[17] S. Even, O. Goldreich and A. Lempel. *A randomized protocol for signing contracts*, In Communications of the ACM, Vol. 28 Iss. 6, pages 637 - 647 (1985)

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

[19] Naor and B. Pinkas, *Efficient Oblivious Transfer Protocols*, Proceedings of SODA 2001 (SIAM Symposium on Discrete Algorithms), 2001.

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

[21] D. McGrew, K. Igoe and M. Salter, *Fundamental Elliptic Curve Cryptography Algorithms*, RFC 6090, February 2011.