Secure Two-Party Computation: Generic Approach and Exploiting Specific Properties of Functions Approach

A. Anasuya Threse Innocent, K. Sangeeta
Department of CSE
Amrita School of Engineering
Amrita Vishwa Vidyapeetham University
Bangalore, India
{a_anasuya, k_sangeeta}@blr.amrita.edu

ABSTRACT: *Introduced by Yao in early 1980s, secure computation is being one among the major area of research interest among cryptologists. In three decades of its growth, secure computation which can be called as two-party computation, or multiparty computation depending on the number of parties involved has experienced vast diversities. Research has been carried out by exploiting specific properties of functionalities and generic approach to achieve efficient practical secure computation protocols. This paper considers the application secure two-party computation of AES-128 for comparison of the above two approaches.*

Keywords: Secure Computation, 2PC, Generic Approach, Garbled Circuits, Oblivious Transfer

Received: 10 December 2013, Revised 12 January 2014, Accepted 17 January 2014

1. Introduction

Secure computation, the term introduced by Yao [1] in early 1980's, is a part of cryptography in which two or more parties with private inputs wish to compute some joint function of their inputs. The problem behind secure computation can be stated as follows, "*Consider a set of parties who do not trust each other, nor the channels by which they communicate. Still, the parties wish to correctly compute some common function of their local inputs, while keeping their local data as private as possible.*" In other words, "*combining information while protecting it as much as possible*" is termed as secure computation. When this computation involves only two parties then it is called as *secure two-party computation* or simply as *two-party computation* denoted as 2*PC*.

The definition of 2PC as given by Oded Goldreich [2] can be stated as follows: "*In two-party computation, two parties with respective private inputs x and y, wish to jointly compute a functionality* $f(x, y) = (f_1(x, y), f_2(x, y))$, *such that the first party receives* $f_1(x, y)$ *and the second party receives* $f_2(x, y)$."

Likewise the idea behind *Multi-Party Computation* is that it should enable three or more parties to compute any function of their choosing on their secret inputs, without revealing their inputs to other parties. The formal definition of *secure multiparty computation* or simply *multiparty computation* (normally denoted as *SMC* or *MPC*) can be stated as: "*MPC is an marray functionality denoted* $f: (\{0, 1\}^*)^m \rightarrow (\{0, 1\}^*)^m$, *such that the m-parties with their corresponding m inputs,* $[x] = (x_1, x_2, \ldots, x_m)$, *wish to jointly compute a functionality* $f([x]) = (f_1([x]), f_2([x]), \ldots, f_m([x]))$, *such that the ith party should receive* $f_i([x])$." One example is an election, voters want their vote to be counted but they do not want their vote made public. Another example is,

studying the effect of a newly developed medicine on patients without revealing their individual test results or identities. So, need is to conduct computation on private data that reveals nothing more than the desired output and what is inherently leaked by it.

Secure computation can be used in variety of applications ranging from simple coin tossing, mutual agreement, to complex applications like electronic voting, electronic auctions, private data retrieval, analysis of sensitive information, privacy preserving biometric identification, private editing in cloud etc. Even though secure computation can be applied for variety of applications, due to privacy concerns and constraints in efficiency, it is not yet widely used in real life applications.

The requirements for secure computation protocols are: privacy, correctness, and independence of inputs. *Privacy* means that the parties involved in computation learn only the final output of the computation and nothing else, *correctness* means that the output is correctly distributed among the parties, and, *independence of inputs* means that the parties involved cannot make their inputs depending on other parties' inputs. The secure computation protocol design can follow the generic approach suggested by Yao, or can exploit properties of some functionalities to create one. This paper compares the above two approaches for secure computation and concludes that the generic approach is the best and practicable in case of semihonest treat model for 2PC protocols.

The paper is organized as follows; Section 2 explores the literature available for 2PC, Section 3 describes a comparative study of existing 2PC protocols studied on AES-128, and next Section concludes the paper and suggests an efficient system.

2. Prior Work

The 2PC protocol was first introduced by Yao [1] to solve the millionaires' problem. The cryptographic background needed to carry out 2PC protocols, garbled circuits and oblivious transfers were clearly described by Lindell and Pinkas [3]. Researchers' have been working on for the past three decades to improve the efficiency of secure computation protocols, in particular the communication complexity of protocols measured in bits, the time complexity of the computations done by the protocol, communication complexity in number of messages, and communication complexity in number of rounds [4]. The reason for focusing on the bit complexity is that the bit complexity is the main bottleneck responsible for secure computation protocols not being practical. Construction of secure computation protocols follows two approaches (i) generic approach which relies on completeness theorems for secure computation, allows protocols for computing any function *f* starting from a Boolean circuit representation of the function, *f* (ii) second approach exploits the specific properties of a function to design special purpose secure computation protocols. The protocols following second approach [5, 6, 7, 8, 9, 10, 11, 12, 13] are proven to be more efficient than most of the existing generic approach protocols, reason being that each are function-specific; which is also the drawback since each protocol must be designed, implemented, and proven secure but cannot be reused or applied for other applications.

The generic approach, introduced by Yao [1,14] and developed by Goldreich [2,15] is based on completeness theorem. The main components in generic approach are garbled circuit and oblivious transfers, which are explained in detail in the following subsections.

2.1 Garbled Circuits

Garbled circuit construction is the basic idea behind Yao's protocol, which constructs a circuit in a way that only the output values of the circuit is revealed, not the inputs or any intermediate values. The term *garbled circuit* was introduced by Beaver, Micali, and Rogaway [16] in 1990. In any secure computation system with garbled circuits (GC), the constructor of the GC, constructs the GC and sends it to the evaluator for evaluation. The intermediate values of the gates produced are meaningless to the parties involved, but the output values of the circuit is intelligible and is guaranteed to be correct.

Bellare, Hong, and Rogaway [17] calls the garbled circuit construction as the garbling scheme and has denoted it as a five-tuple algorithm, $G = (Gb,En,De,Ev, ev)$. The garbling algorithm Gb transforms a six-tuple function $f = (n, m, q, A, B,G)$, $f: \{0,1\}^n \rightarrow \{0,1\}^m$ into a triple of functions $(F, e, d) \leftarrow Gb(f)$, where the encoding function e turns an initial input $x \geq \{0,1\}^n$ into a garbled input $X = e(x)$, the garbled function F produces the garbled output $Y = F(X)$, the decoding function d produces the final output $y = d(Y)$, which must coincide with $f(x)$. In the six-tuple function f, $n \geq 2$ is the number of inputs, $m \geq 1$ is the number of outputs, $q \geq 1$ is the number of gates, A is a function to identify each gate's first incoming wire, B is a function to identify the second

incoming wire, and G is the function to determine the functionality of the gates. Here, possession of e and x allows one to compute the encrypted garbled input $X = En(e, x)$, F and X allows to generate the garbled output $Y = Ev(F, X)$, and knowing d and F helps to recover the decrypted final output $y = De(d,Y)$, which must be equal to the evaluated value, $ev(f, x)$. The correctness condition can be defined as, $De(d, Ev(F, En(e, x))) = ev(f, x)$. The garbling scheme [17] also ensures correctness, privacy, authenticity, and obliviousness.

In the earlier generic approach protocols, the garbled inputs per gate used *double encryption* [3], which increases the complexity and hence reduces the efficiency. In garbling scheme [17], *dual-key cipher* is used, i.e., per gate only one encryption call is made, which results in faster circuit construction. Kolesnikov and Schruider [18] introduced the *free XOR technique*, i.e., the XOR gates are evaluated without the use of the associated garbled tables and the corresponding hashing or symmetric key operations. The concept of reducing the size of the garbled tables was pointed out by Naor, Pinkas, and Sumner [19] and was effectively used as the technique, *garbled row reduction* by Pinkas, Schneider, Smart and Williams [20]. The project *Might Be Evil* by Huang, Evans, Katz and Malka [21] uses free XOR technique, garbled row reduction, and oblivious-transfer extension optimizations; also the entire GC is never stored in their implementation. Topological sorting of the gates of the circuit and pipelining the process of circuit generation and evaluation significantly improves the overall efficiency of the 2PC protocol [21]. The *Just Garble* system developed by Bellare, Hoang, Sriram, and Rogaway [22] with the goal of optimized garbling is based on a fixed-key AES, making only one AES call per garbled gate evaluation. Here, the dual-key cipher's permutation is initiated by fixed-key AES, which considerably improves the efficiency of the system.

2.2 Oblivious Transfers

Oblivious transfers along with the garbled circuit forms the core of secure computation. In 2PC, the basic conditions are: (i) the first party who acts as the sender learns nothing of the second party's i.e. receiver's input value, and (ii) the receiver obtains only a single set of keys and so can compute the circuit on only a single value as required. These conditions holds under the assumption that the keys associated with the circuitinput wires are obtained in an oblivious manner that does not reveal the association with the parties' inputs. The first oblivious transfer (OT) protocol is the 1-*out-of*-2 *oblivious transfer* protocol developed by Even, Goldreich, and Lempel [23] in 1985. In 1-out-of-2 OT, a sender inputs two values k_0 and k_1, and the receiver inputs a selection bit b. The outcome of the protocol is that the receiver obtains k_b, but he learns nothing about $k_1 - b$, as well the sender learns nothing about the receivers selection bit b.

The *oblivious-transfer extension* introduced by Ishai, Kilian, Nissim, and Petrank [24] can achieve virtually unlimited number of oblivious transfers at the cost of the number of statistical security parameter executions of 1-out-of- 2 OT. The *cut-and-choose OT* protocols developed by Lindell and Pinkas [25, 26] are effective in case of malicious adversaries. The cut-and-choose OT works by first party constructing s copies of a garbled circuit and sending them to second party, who asks the first party to open half of them in order to verify that a majority of the unopened half are also correct, except with probability that is negligible in s. Huang, Katz, and Evans [27] have developed the *symmetric cut-andchoose OT*, in which both the parties generate garbled circuits equal to the number of statistical security parameter in case of malicious adversaries. They have proved that in this doublesided approach, even by reducing the number of garbled circuits generated by a factor of 3, were able to achieve the same level of statistical security, as of the previous single-sided approaches. Also, the symmetric cut-and-choose OT is three times faster than the single-sided approaches. Thus, the design of OT also contributes to the efficiency of secure computation protocols.

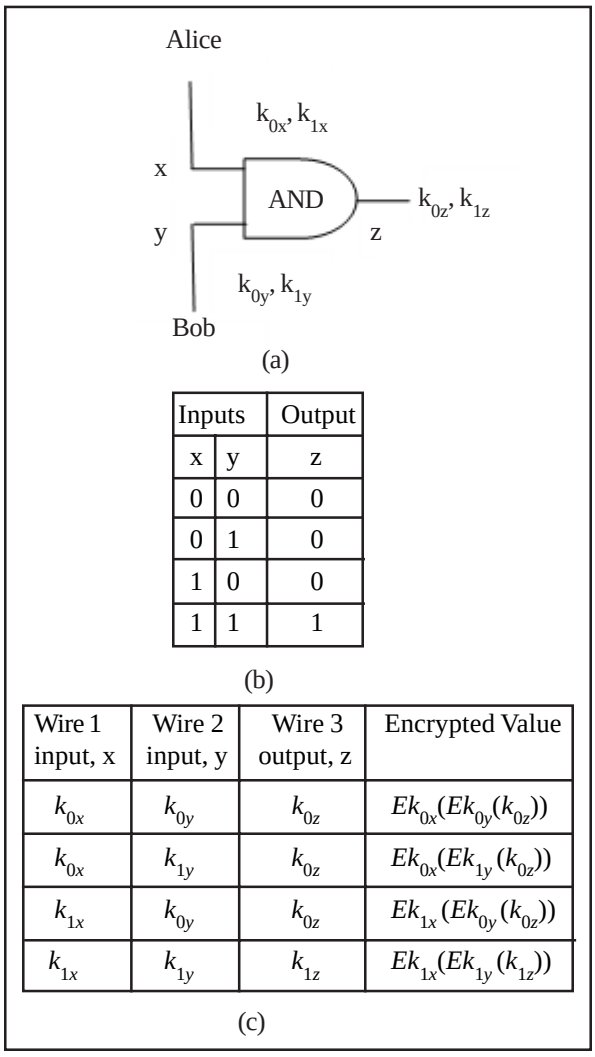
2.3 Steps for Garbled Circuit Construction

The steps for garbled circuit construction of Yao's protocol are described below with a simple example. Let Alice and Bob be the two parties involved in 2PC, and let f be any polynomial function that has to be used for secure computation. First, convert f into Boolean circuit, then evaluate a single gate and proceed for the whole circuit.

Step 1: Select Random Keys for Each Wire

Alice selects two random keys for each wire: one key corresponds to 0 and the other to 1. Total 6 keys are chosen for a gate with 2 input wires x, y and one output wire, z. The 6 corresponding keys selected are labeled as: (k_{0x}, k_{1x}), (k_{0y}, k_{1y}), *and* (k_{0z}, k_{1z}). For illustration purpose, considering a simple AND gate, let the six random keys with the key length of 8- bits be: (10011001, 11001100), (10000001, 11110000), and (00110011, 10101010).

Step 2: Encrypt Truth Table



(a)

Inputs		Output
x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

(b)

Wire 1 input, x	Wire 2 input, y	Wire 3 output, z	Encrypted Value
k_{0x}	k_{0y}	k_{0z}	$Ek_{0x}(Ek_{0y}(k_{0z}))$
k_{0x}	k_{1y}	k_{0z}	$Ek_{0x}(Ek_{1y}(k_{0z}))$
k_{1x}	k_{0y}	k_{0z}	$Ek_{1x}(Ek_{0y}(k_{0z}))$
k_{1x}	k_{1y}	k_{1z}	$Ek_{1x}(Ek_{1y}(k_{1z}))$

(c)

Figure 1. a) AND gate with input-output wires and keys corresponding to the wires, (b) Truth table of AND gate, and (c) Encrypted truth table with inputs and output as the keys correspond to the wires

Alice encrypts each row of the truth table by encrypting the output wire key with the corresponding pair of input wire keys. For an AND gate, the gate input-outputs, the truth table, and the encrypted truth table are shown in Figure 1.

Continuing with the illustration, let the encryption used be simple bitwise XOR. Table 1 shows the values of inputs and encrypted output values.

Wire 1 input, x	Wire 2 input, y	Wire 3 output, z	Encrypted Value
10011001	10000001	00110011	00101011
10011001	11110000	00110011	01011010
11001100	10000001	00110011	01111110
11001100	11110000	10101010	10010110

Table 1. Example of key inputs and encrypted output values for an AND gate using XOR as encryption function

Step 3: Send Garbled Truth Table

Alice randomly permutes/scrambles i.e. garbles encrypted truth table and sends it to Bob. Here, Bob does not know which row of garbled table corresponds to which row of original table. The encrypted and the corresponding garbled truth tables are shown in Figure 2, and Table 2 shows the values.

Row	Encrypted Value	Row	Encrypted Value
1	$Ek_{0x}(Ek_{0y}(k_{0z}))$	1	$Ek_{1x}(Ek_{0y}(k_{0z}))$
2	$Ek_{0x}(Ek_{1y}(k_{0z}))$	2	$Ek_{0x}(Ek_{0y}(k_{0z}))$
3	$Ek_{1x}(Ek_{0y}(k_{0z}))$	3	$Ek_{1x}(Ek_{1y}(k_{1z}))$
4	$Ek_{1x}(Ek_{1y}(k_{1z}))$	4	$Ek_{0x}(Ek_{1y}(k_{0z}))$

Figure 2. (a) Encrypted truth table, and (b) Garbled truth table

Row	**Encrypted value**	**Garbled value**
1	00101011	01111110
2	01011010	00101011
3	01111110	10010110
4	10010110	01011010

Table 2. Encrypted Values and the Garbled Values

Step 4: Send Keys for Alice's Input

Alice sends the key corresponding to her input bit. As the keys are of random nature, Bob does not learn which bit (0 or 1) the key correspond to. Suppose a (0 or 1) is the input bit of Alice, Bob will receive k_{ax}, from that he cannot retrieve the value of a. Suppose if $a = 1$, Alice will send $k_{1x} = 11001100$ to Bob.

Step 5: Use OT on Keys for Bob's Input

Now, to get the key corresponding to Bob's input, both Alice and Bob run the 1-out-of-2 oblivious transfer protocol. Alice's input to OT protocol is the two keys corresponding to Bob's input wire: k_{0y} and k_{1y}. Bob's input to OT protocol is simply his selection bit b, which can be 0 or 1. As the output of OT, Bob learns k_{by} which is the key corresponding to his input bit b. Here, Alice's input to OT protocol is the key pair, (10000001, 11110000). Suppose if Bob's input $b = 1$, he will receive $k_{1y} = 11110000$, and does not know anything about k_{0y}.

Step 6: Evaluate Garbled Gate

Using the two keys that he learned through Step 4 and Step 5 i.e., k_{ax} and k_{by}, Bob decrypts exactly one of the outputwire keys. But Bob does not know if the key he has obtained corresponds to 0 or 1. For example, if $a = 1$ and $b = 1$, then the values on his possession are k_{1x} and k_{1y}, and from Figure 2 (b) it is very clear that Bob can decrypt only the third row and will retrieve k_{1z}, but not the output bit. Here, when a and b are 1, Bob will have $k_{1x} = 11001100$, $k_{1y} = 11110000$ and from Table 2 garbled value column, only the 3rd row will be decrypted to achieve $k_{1z} = 10101010$.

Step 7: Evaluate Entire Garbled Circuit

Using Step 1 to Step 6, Bob evaluates the entire garbled circuit. For each wire in the circuit, Bob learns only one key and he does not know to which bit (0 or 1) it corresponds to. Hence Bob cannot learn the intermediate values. As well Alice will not know which will be Bob's input bit, as well which key Bob has obtained, hence she cannot try to retrieve the intermediate values. Bob after evaluating the entire garbled circuit will obtain a single value for the final output key, and which he will informs to Alice, in turn Alice tells Bob if it corresponds to 0 or 1.

This is how the Yao's protocol for millionaires' problem is being solved. Here, double-encryption is used per gate and 1-out-of-

2 OT is used for oblivious transfers. These steps can be customized using dual-key cipher or cut-and-choose oblivious transfer with respect to a specified application.

Algorithm/ Tool/ Project	**AES (per block)**	
	Online Time (s)	***Overall Time (s)***
Second Approach		
SHE scheme [13]	0.72	1.07
Generic Approach		
ROM-GRR [20]	5	7
TASTY [32]	0.4	3.3
Might Be Evil [21]	0.008	0.2
JustGarble [22]	0.004	0.005

The values of time corresponding to AES-128 are collected from the reference papers that are shown inside square brackets under Algorithm/Tool/Project column

Table 3. Comparison Of Aes-128 With Generic Approach and Second Approach Secure Computation Protocols

2.4 Second Approach

Exploiting specific properties of functionalities approach does not have a general structure. It can start with an arithmetic circuit and proceed with oblivious transfer to achieve secure computation, or it can only use oblivious transfer with some modifications, or it can depend only on some functionality to achieve the required result. Some examples of second approach protocols are given below.

Atallah and Du in their CERIAS tech report [5] have explained the two-party vector dominance protocol based on Cachin's protocol [6]. Ioannidis and Grama [7] uses secure dotproduct with oblivious transfer to solve millionaires' problem and online bidding problems. Luo, Huang, and Zhong [8] make use of homomorphic encryption to solve privacy-preserving computational geometry problems such as distance between two private points, and point-circle inclusion problem based on the distance protocol. Desmedt, Pieprzyk, Steinfeld, and Wang [9] considers a non-abelian group (G, .) and represents secure computation function, f_G as an arithmetic circuit over a finite ring R. Computations performed are; multiply, inverse, and random sampling on the group G, which acts as a black-box. Also they show how to construct f_G for reliable colouring of a planar graph. Bickson, Bezman, Dolev, and Pinkas [10] proposes an efficient framework for enabling secure numerical computations in a Peer-to-Peer network, considering only functions which are built using algebraic primitives of addition, subtraction, and multiplication. Homomorphic encryption is also used. I-Wang, Shen, Zhan, Hsu, Liau, and Da-Wang [11] replaces OTs with secure scalar-product and prove that they have completeness property and has the power of integer-based computing. Shi, Luo, and Zhang [12] solves multi-dimensional vector comparison problem using multiplication protocol and scalar products protocol.

3. Comparative Study

This paper compares the 2PC for AES-128 cipher in terms of efficiency measured in time complexity of the computations done by the protocol. Data are collected from different references and the results are tabulated in Table 3. AES-128 or simply AES is a symmetric block cipher developed by Daemen and Rijmen [28]. It uses a 128-bit key as well block size of 128-bits, and has variable number of rounds. The sub-keys generated have a variable key size (16, 24, or 32 bits) depending on the variable block sizes (16, 24, or 32 bit blocks). And the number of rounds also varies accordingly (10, 12 or 14). The round function is uniform and parallel function composed of 4 steps - *SubBytes* to introduce nonlinearity, *ShiftRows* for inter-column diffusion, *MixColumns* for interbyte diffusion within columns, and *AddRoundKey* to make round function key-dependent.

In 2PC of AES, first party holds the key k and the other party holds an input x, which will be encrypted as the result of protocol and the result $AESk$ (x) will be learnt by the second party.

Table 3 provides the online time or garbling time in seconds and the overall time or the garbling + evaluation time in seconds. The protocol considered for exploiting specific properties of functions approach is the *Somewhat Homomorphic Encryption* (SHE) scheme given by Damgard, Pastro, Smart, and Zakarias [13], which is the efficient known protocol of its kind. *Homomorphic Encryption* (HE) is a cryptosystem which allows specific types of computations to be carried out on ciphertext, and decryption results with plaintext with the same computations carried out on it, i.e., HE allows to perform arbitrary computation on encrypted data. The homomorphic encryption scheme can be considered as a quadruple algorithm denoted as, *HE* = (*Keygen, Enc, Dec, Eval*) with a security parameter, *n*. The key generation function, (*pk, evk, sk*) ← *Keygen* (1^n) outputs the public encryption key *pk*, public evaluation key *evk*, and a secret decryption key *sk*. Encryption function encrypts a single bit message $m \epsilon \{0,1\}$ into a ciphertext message *c*, decryption function does the reverse operation. The homomorphic evaluation function uses the evaluation key *evk*, and applies a function $f: \{0, 1\}^l \rightarrow \{0,1\}$ to ciphertexts $c_1, c_2, \ldots, c_l$ and outputs the final ciphertext c_f [28, 29, 30]. The SHE [8] uses a 5- tuple algorithm, *SHE* = (*ParamGen, KeyGen, KeyGen*, Enc, Dec*) with a security parameter *k*. Here, the *ParamGen* (1^k, *M*) function outputs an integer *N*, which will be used for encode/decode operations. The addition and multiplication operations on homomorphic encryption are denoted by + and •, and satisfy the following properties for all elements N_1, N_2 õ *M*, where *M* is the plaintext space; *decode* (*encode* (N_1) + *encode* (N_2)) = $N_1 + N_2$, *decode* (*encode* (N_1) • *encode* (N_2)) = $N_1 \bullet N_2$. *KeyGen* function generates the public-private key pair *pk* and *sk*. *KeyGen** is a randomized function which outputs a meaningless public key *pk'* such that *pk* and *pk'* are indistinguishable. It is used just to ensure security. Homomorphic encryption schemes use arithmetic circuits for processing and not the Boolean circuits as of the generic approach.

On generic approach, the ROM-GRR protocol (Random Oracle Model – Garbled Row Reduction protocol) [20] makes use of the free XOR technique and the garbled row reduction technique and is applied on semi-honest adversaries model. TASTY [32] is a framework and compiler for 2PC which outperforms the ROM-GRR protocol on generic approach as well any of the best protocols known on second (exploiting functions) approach. The 2PC protocol developed by the Might Be Evil project [21], introduced more control on the circuits than that of TASTY. Also introducing the pipelining process of circuit generation and without storing the garbled circuits, it improves the efficiency of the protocol. The JustGarble system [22] outperforms all of the 2PC systems known till now by the use of fixed-key AES using dual-key cipher (DKC). JustGarble system implements three protocols namely, Ga (only garbling), GaX (garbling and free XOR technique), and GaXR (garbling, free XOR technique, and garbled row reduction). Here gates are not considered as objects that communicate by sending messages, but they are indexed into an array. Gates are topologically ordered, so there is no queue of gates ready to be evaluated, hence one evaluates them in numerical order. This simplicity helps evaluating a GC an actual cryptographic work, not overhead related to procedure invocation, message passing, and etc. Also JustGarble makes use of AES-NI, which enhances the speed of execution of the AES encryptiondecryption routines. From Table 3 it is clear that the generic approach protocols have more efficiently emerged from the theoretical ones to practical ones which can be well suited for real life applications.

4. Conclusion

This paper summarizes the generic approach and second approach for secure 2PC, and compares the two approaches with secure computation of AES-128 application. The generic approach combined with best optimizations can help to obtain better protocol designs for secure 2PC. The JustGarble system which is the best known for semi-honest adversary model can be made more efficient by without storing the entire GC, which will lead to a considerable improvement in its time complexity. It can also be tried on malicious adversary model with the symmetric cut-and-choose oblivious transfer for better efficiency.

References

[1] Yao, A. (1982). Protocols for secure computation. (extended abstract), *In*: Proceedings of 21st Annual IEEE Symposium on Foundations of Computer Science, p. 1-5.

[2] Goldreich, O. (2004). Foundations of Cryptography: Volume 2 – Basic Applications, Cambridge University Press.

[3] Lindell, Y., Pinkas, B. (2009). A proof of security of Yao's protocol for two-party computation, J. Cryptology, 22, p. 161-188.

[4] Schoenmakers, B. (Ed.). (2008). Summary report on secure computation protocols, ECRYPT.

[5] Atallah, M., Du, W. (2001). A multi-dimensional Yao's millionaire protocol, CERIAS Tech Report 2001-09.

[6] Cachin, C. (1999). Efficient private bidding and auctions with an oblivious third party, Proceedings of the 6th ACM conference on Computer and communications security, p. 120–127, Singapore, November 1-4.

[7] Ioannidis, I., Grama, A. (2003). An efficient protocol for Yao's millionaires' problem, *In*: Proceedings of 36th IEEE Hawaii International Conference on System Sciences.

[8] Luo, Y., Huang, L., Zhong, H. (2007). Secure two-party point-circle inclusion problem, J. Comput. Sci. & Technol., Jan, 22 (1) 88-91.

[9] Desmedt, Y., Pieprzyk, J., Steinfeld, R., Wang, H. (2007). On secure multiparty computation in black-box groups, Advances in Cryptology – CRYPTO, LNCS, 4622, p. 591-612.

[10] Bickson, D., Bezman, G., Dolev, D., Pinkas, B. (2008). Peer-to-peer secure multi-party numerical computation, Appeared in the 8th IEEE Peer-to- Peer Computing (P2P), Aachen, Germany, Sept, arXiv:0810.1624.

[11] Wang, I., Shen, C., Zhan, J., Hsu, T., Liau, C., Wang, D. (2009). Toward empirical aspects of secure scalar product, IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, 39 (4), July, p. 440-447.

[12] Shi, L., Luo, Y., Zhang, C. (2009). Secure two-party multi-dimensional vector comparison protocol, IEEE International Conference on Management and service Science, p. 1-5.

[13] Damgard, I., Pastro, V., Smart, N., Zakarias, S. (2012). Multiparty computation from somewhat homomorphic encryption, Advances in Cryptology – CRYPTO, LNCS, 7417, p. 643-662.

[14] Yao, A. (1987). How to generate and exchange secrets, *In*: Proceedings of 27th IEEE FOCS, p. 162-167

[15] Goldreich, O., Micali, M., Wigderson, A. (1987). How to play any mental game (or) A completeness theorem for protocols with honest majority, *In*: Proceedings of 19th ACM Symposium on Theory of Computing, p. 218-229.

[16] Beaver, D., Micali, S., Rogaway, P. (1990). The round complexity of secure protocols, *In*: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, p. 503-513.

[17] Bellare, M., Hoang, V., Keelveedhi, S., Rogaway, P. (2012). Foundations of garbled circuits, *In*: Proceedings of the 2012 ACM Conference on Computer and Communications Security, p. 784-796. Full version as Cryptology ePrint Archive, Report 2012/265, Oct. 1.

[18] Kolesnikov, V., Schneider, T. (2008). Improved garbled circuit: Free XOR gates and applications, Automata, Languages and Programming, LNCS, 5126, p. 486-498.

[19] Naor, M., Pinkas, B., Sumner, R. (1999). Privacy preserving auctions and mechanism design, *In*: Proceedings of the 1st ACM Conference on Electronic commerce, p. 129-139.

[20] Pinkas, B., Schneider, T., Smart, N., Williams, S. (2009). Secure two-party computation is practical, Advances in Cryptology – ASIACRYPT, LNCS, 5912, p. 250-267.

[21] Huang, Y., Evans, D., Katz, J., Malka, L. (2011). Faster secure two-party computation using garbled circuits, *In*: Proceedings of the 20th USENIX Conference on Security, p. 35-49.

[22] Bellare, M., Hoang, V., Keelveedhi, S., Rogaway, P. (2013). Efficient garbling from a fixed-key blockcipher, IEEE Symposium on Security and Privacy, p. 478-492.

[23] Even, S., Goldreich, O., Lempel, I. A. (1985). A randomized protocol for signing contracts, communications of the ACM, June 28 (6) 637-647.

[24] Ishai, Y., Kilian, J., Nissam, K., Petrank, E. (2003). Extending oblivious transfers efficiently, CRYPTO, LNCS, 2729, p. 145-161.

[25] Lindell, Y., Pinkas, B. (2007). An efficient protocol for secure two-party computation in the presence of malicious adversaries, Advaces in Cryptology – EUROCRYPT, LNCS, 4515, p. 52-78.

[26] Lindell, Y., Pinkas, B. (2011). Secure two-party computation via cut-andchoose oblivious transfer, Theory of Cryptography, LNCS, 6597, p. 329-346. Full version as ePrint Archive, Report 2010/284, Sep. 18.

[27] Huang, Y., Katz, J., Evans, D. (2013). Efficient secure two-party computation using symmetric cut-and-choose, Advances in Cryptology – CRYPTO, LNCS, 8043, p. 18-35.

[28] FIPS PUB 197. (2001). Advanced Encryption Standard (AES), National Institute of Standards and Technology, U.S. Department of Commerce, November. http://csrc.nist.gov/publications/fips/fips197/fips- 197.pdf

[29] Gentry, C. (2010). Computing arbitrary functions of encrypted data, *Communications of the ACM*, March, 53 (3) 97-105.

[30] Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V. (2010). Fully homomorphic encryption over the integers, Advances in Cryptology – EUROCRYPT, p. 24-43.

[31] Brakerski, Z. (2012). Fully homomorphic encryption without modulus switching from classical GapSVP, Advances in Cryptology – CRYPTO, p. 868-886.

[32] Henecka, W., Kogl, S., Sadeghi, A., Schneider, T., Wehrenberg, I. (2010). TASTY: Tool for automating secure two-party computations, *In*: Proceedings of the 17th ACM Conference on Computer and Communications Security, p. 451-462