Int. J. Web and Grid Services, Vol. X, No. Y, XXXX

Reusable garbled gates for new fully homomorphic encryption service

Xu An Wang*

School of Telecommunications Engineering,
Xidian University,
Xi'an, China
and
Key Laboratory of Information and Network Security,
Engineering University of Chinese Armed Police Force,
Xi'an, China
Email: wangxazjd@163.com
*Corresponding author

Fatos Xhafa

Department of Computer Science,
Technical University of Catalonia,
Barcelona, Spain
Email: fatos@cs.upc.edu

Jianfeng Ma

School of Cyber Engineering,
Xidian University,
Shaanxi, China
Email: jfma@mail.xidian.edu.cn

Yunfei Cao and Dianhua Tang

Science and Technology on Communication Security Laboratory,
Chengdu, China
Email: cao_426@126.com
Email: 248848036@qq.com

Abstract: In this paper, we propose a novel way to provide a fully homomorphic encryption service, namely by using garbled circuits. From a high level perspective, Garbled circuits and fully homomorphic encryption, both aim at implementing complex computation on ciphertexts. We define a new cryptographic primitive named reusable garbled gate, which comes from the area of garbled circuits, then based on this new primitive we show that it is very easy to construct a fully homomorphic encryption. However, the instantiation of reusable garbled gates is rather difficult, in fact, we can only instantiate this new primitive based on indistinguishable obfuscation. Furthermore, reusable garbled gates can be a core component for constructing the reusable garbled circuits, which can reduce the communication complexity

of them from $O(n)$ to $O(1)$. We believe that reusable garbled gates promise a new way to provide fully homomorphic encryption and reusable garbled circuits service fast.

Keywords: fully homomorphic encryption service; reusable garbled gate; reusable garbled circuits service; indistinguishable obfuscation.

Reference to this paper should be made as follows: Wang, X.A., Fatos, X., Ma, J., Cao, Y. and Tang, D. (xxxx) 'Reusable garbled gates for fully homomorphic encryption service', *Int. J. Web and Grid Services*, Vol. x, No. x, pp.xx–xx.

Biographical notes: Xu An Wang is an Associate Professor in the Engineering University of Chinese Armed Police Force. His main research interests include public key cryptography and cloud security.

Fatos Xhafa is a Professor at Technical University of Catalonia. His main research interests include cloud computation and big data analysis. He has widely published in the field of grid, internet, cloud and big data computation.

Jianfeng Ma is a Professor in Xidian University. He is also a Yangtze River scholar in China. His main research interests include cyber engineering security and big data analysis. He has published widely in the field of public key cryptography and Cloud security.

Yunfei Cao is a principle senior researcher in Science and Technology on Communication Security Laboratory. His main research interests are cryptography and information security.

Dianhua Tang is a principle researcher in Science and Technology on Communication Security Laboratory. His main research interest is fully homomorphic encryption and cloud security.

1 Introduction

Nowadays Cloud computing platforms are offering computation services to individuals, institutions, organisation, enterprises, etc. widely. Cloud computation is a generalisation technique of grid computation, which has revolutionised the traditional distributed large scale computation paradigm. Traditionally, we heavily relied on the local computation equipments like personal computers and small clusters to implement computation tasks, but this paradigm has some limitations such as costly hardware management, poor scalability and only local accessibility, etc. Along with the rapid development of both Internet and hardware techniques, cluster and grid computation and later on cloud computing became a reality. Many computation devices can now be organised to support a very strong computation ability, many originally unimagined computation task can |now be completed by distributed or parallel grid/cloud computation techniques like searching life information from huge collected space signals. These years by utilising the advanced techniques such as virtual machines and fast/broad/anywhere/anytime internet accessibility, cloud computation has emerged. Cloud computation enables the traditional local storage/computation/accessibility now as a service; it can support various levels of service: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS), etc.

However, when data owners outsource their data to the cloud, they often worry about the privacy of their datum. Security is among the most concerned issues for cloud computation widely adoption (Chen et al., 2012; Chen et al., 2014; Dutu et al., 2014; Guo and Xu, 2015; Thabet et al., 2014; Wang et al., 2016; Zhu and Yang, 2015). A natural solution for the above issue is this: the data owners first encrypt their datum and then outsource them to the cloud. Although it solves the privacy issue, it poses another challenge to the cloud computation: in most cases data owners does not simply utilise the cloud to store their data but also utilise it to complete computation tasks, such as running data mining tasks on their data. However, when the outsourced data is in the form of encrypted ciphertext, how could the cloud run this data mining task efficiently? Fortunately we can solve this challenging issue by using several techniques developed from cryptography community. Fully homomorphic encryption (FHE) is the most promising one among them, it can enable arbitrary privacy preserving computation on the ciphertexts, just like directly running computation on the plaintexts but with the results also encrypted. To put is shortly, it can support blindly computation but getting the corrected encrypted result. An example of fully homomorphic encryption service for cloud computation can be seen in Figure 1.

1 Data owner Alice first encrypts her file by using her own encryption key and then outsource the encrypted files $E_{Alice}(file)$ to the cloud, she also outsource the computation task to the cloud, that is, an arbitrary function F, which can be seen as the data mining function or algorithm.

2 After obtaining the encrypted file and the data mining function F, the cloud runs the fully homomorphic encryption service on them and returns the encrypted result $E_{Alice}(F(file))$ to Alice. Note here the computation task is implemented in a privacy preserving way, the cloud does not know Alice's file or the finally computed result. Also note currently fully homomorphic encryption cannot be implemented in an efficient way, that is also why users prefer to utilise the cloud to implement this task.

3 After obtaining the encrypted computed result, Alice can decrypt the encrypted result and obtain what she wanted to compute.

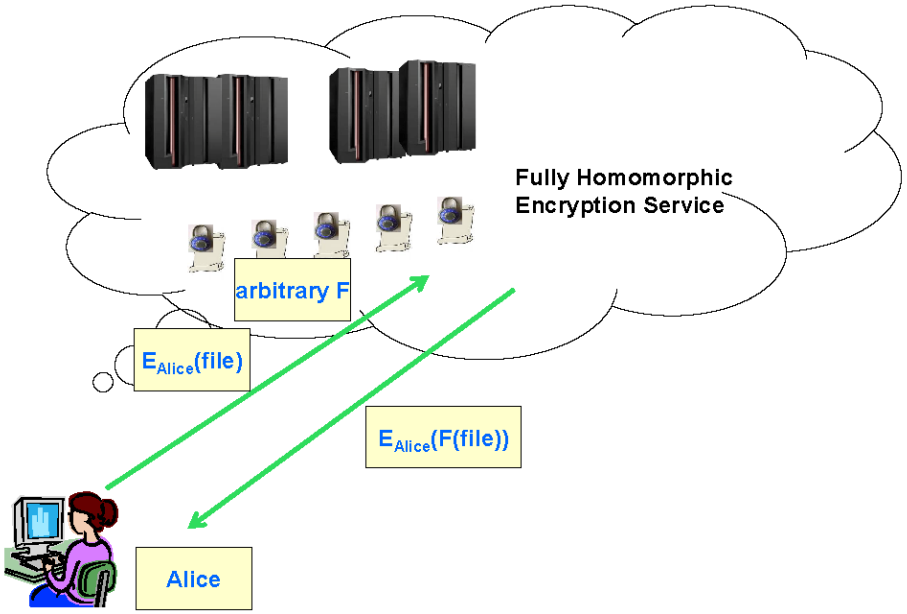
However there are two main challenges for the fully homomorphic encryption service deployment. The first one is how to design fully homomorphic encryption efficiently as currently most of them rely on the public cryptographic techniques, which are not efficient for many of such operations. Is it possible to construct fully homomorphic encryption based on symmetric cryptographic primitive such as AES or SHA3?

The second one is how to design various kinds of fully homomorphic encryption schemes, currently most of them rely on the cryptographic lattice tools. Is it possible to construct fully homomorphic encryption based on classic computation problems such as RSA or DL problem? In this paper, we approach the solution of these two problems.

Our key observation is that Yao's garbled circuits techniques can be utilised to construct fully homomorphic encryption. Garbled circuits enable (very fast) secure multi-party computation; fully homomorphic encryption is also the most promising technique to implement secure computation on ciphertexts. Roughly speaking, these two techniques are both aimed at implementing complex computation on ciphertexts. Somehow surprisingly until now little effort has been made on bridging the gap between these two

techniques. We give a first such attempt, namely, we define a new cryptographic primitive called reusable garbled gate, then based on this new primitive we show that it is very easy to construct fully homomorphic encryption. However, the instantiation of reusable garbled gates seems to be very difficult, we can only instantiate this new primitive based on indistinguishable obfuscation (iO), which until now has no efficient realisation. Below we first review Yao's garbled circuits and analysis the difficulty of construct fully homomorphic encryption based on Yao's garbled circuits, then we give our contribution and the detailed related work.

Figure 1 An example of fully homormorphic encryption service for cloud computation (see online version for colours)

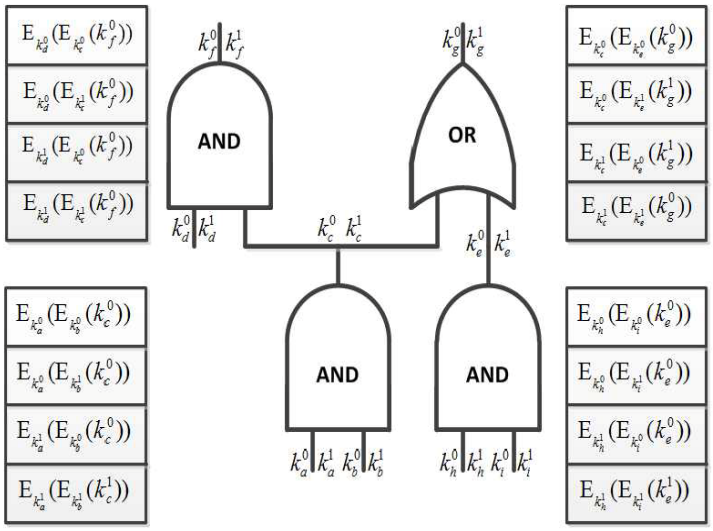


1.1 Yao's garbled circuits

The technique of garbled circuit was first proposed by Yao in 1980s (Yao, 1982; Yao, 1986) to implement the secure two-party computation mechanisms. The core idea of this technique is the following: in the first phase, the sender (or the generator) first simultaneously garbled the inputs and the computed function's circuit, and then send them to the receiver (or the evaluator), while the receiver runs the oblivious transfer protocol with the sender to get the garbled results corresponding to his inputs; in the second phase, the receiver evaluates the circuit on the received garbled inputs and gets the output garbled results; in the third phase, the sender recovers the final function's output from the garbled results. In this process, the receiver can correctly compute the function on both parties' inputs while he knows nothing about the sender's inputs, and the sender can ensure the receiver has computed the desired function correctly without knowing the receiver's inputs. Furthermore, the receiver cannot know the concrete

function he computed, thus achieving privacy preserving for the sender. This technique has some very interesting properties such as supporting arbitrary complex functions, being privacy preserving for the sender, being implementing very fast (garbled circuits can only use symmetric cryptographic primitives like hash functions and block ciphers), being very flexible. Thus it is among the most promising techniques to implement secure computation on large data sets like secure data mining, secure outsourcing computation task for mobile equipments, secure signal processing and even secure handling of big data.

Figure 2 A simple example of Yao's garbled circuit



Example 1.1: A concrete construction of Yao's garbled circuit can be seen in Figure 2. In the first phase, the sender garbles the input bits of $\{0,1\}^n$ to random keys (like DES's keys), and then constructs a table consisting of four ciphertexts for every gate in the circuits (consisting of AND, OR and XOR gates). These four ciphertexts are the encryption of the gate's output key under the input keys (by using double DES encryption). For there are four inputs (00,01,10,11) for every gate, there are four ciphertexts for one table (or one gate). In the second phase, the receiver evaluates the circuits from bottom to top as following: he first decrypts the four ciphertexts at the bottom level to obtain the correct output key, and then uses these evaluated output keys to continuously decrypt the up level gates' four ciphertexts to again obtain the correct output key, thus finally gets the output keys of the output gates. In the third phase, the receiver returns these output keys to the sender, and the sender recovers the corresponding bits by using the bits-keys map he set up in the first phase.

Yao's garbled circuit has important applications in cryptography: on one hand, it is one of the foundational techniques for secure two-party or multi-party computation; on the other hand, by using only symmetric primitives like DES or AES block cipher, it can achieve very high efficiency, furthermore by using various optimisation techniques, its efficiency can be very impressive, thus it is a very popular basic tool for privacy-preserving computation on large data sets and secure outsourcing computation for resource-constrained devices such as mobile phones.

But Yao's garbled circuit has its own restrictions, for instance, it is not reusable. The sender needs to garble again the circuit and the inputs when the inputs are changed. The main reasons are the followings: if the receiver obtains the same two same in two different evaluations but for the same circuit, the receiver can easily decide the bits corresponding to these two evaluations are the same. Thus it is very important to realise reusable garbled circuits. Goldwasser et al. (2013) were the first to make a step forward in this direction, they constructed the first reusable garbled circuits based on functional encryption with fully homomorphic encryption, thus it is not efficient. Recently, Boneh et al. (2014) proposed a new cryptographic primitive named fully key homomorphic encryption and showed a concrete construction based on lattice, they also showed their primitive can be used to construct reusable garbled circuits with better size, but their result is also not efficient. Thus, an important research open problem is how to realise reusable garbled circuits efficiently.

1.2 Toward basing fully homomorphic encryption on Yao's garbled circuit

Gentry (2009) first constructed a fully homomorphic encryption based on the ideal lattice, which is a breakthrough result in cryptography. Since then, many wonderful results have been achieved. To give a clear picture on how fully homomorphic encryption works, here we give a very high level view on this primitive. Roughly speaking, a fully homomorphic encryption has the following three requirements: first, anyone can generate the cipheretexts corresponding to any plaintext; second, anyone (the evaluator) can compute any function on the ciphertexts; third, the evaluator has no idea on the contents of the ciphertext and the evaluated results.

It is a well-known fact that any computation function can be converted to a circuit. thus here we review some basic facts on circuits for computation of any function. Consider the computation of any function on bits $(x_1, x_2, \ldots, x_n)$, that is, functions that can be described as $f(x_1, x_2, \cdots, x_n)$, where f is any function. Now the big problem for FHE construction is the following: how to privately evaluate the circuits? Note that an essential difference between garbled circuit and FHE is that the former can only be implemented on one function one time, while the latter can support arbitrary function on the same ciphertexts. Thus the biggest issue we need to solve is how to extend Yao's garbled circuit to handle any function.

1.3 Our contribution

1 For the first time we give a try to bridge the gap between fully homomorphic encryption technique and reusable garbled circuit technique.

2 We first introduce the concept of reusable garbled gate, give its somewhat formal definition and properties.

3 We show how to construct fully homomorphic encryption and reusable garbled circuits based on reusable garbled gate.

4 We also give a concrete construction for reusable garbled gate AND, XOR and OR, and point out some interesting open problems which deserving further exploring.

1.4 Organisation

We organise our paper as following. In Section 2, we review the related work on garbled circuits and fully homomorphic encryption in detail. In Section 3, we introduce a new cryptographic concept, the reusable garbled gate. We also give its somewhat formal definition and its security properties. In Section 4, we show how to construct reusable garbled circuit and fully homomorphic encryption based on reusable garbled gate, also point out some promising result which maybe can be achieved. In Section 5, we instantiate the new cryptographic concept- reusable garbled gate based on indistinguishable obfuscation (iO) and roughly analysis its security. We conclude our paper in Section 6 with some interesting open problems.

2 Related work

Research results on the security of garbled circuits. Although the technique of Yao's garbled circuit has been put forward in 1980s, but its rigorous security proof only was given by Lindell and Pinkas (2009) a few years ago. In 2012, Bellare et al. gave a foundational work on garbled circuit, their formal analysis of the properties of garbled circuits need to satisfy, that is, authentication, secrecy and privacy. They pointed out, although there are existing some relation among these three properties, they are different. Furthermore, they proved Yao's garbled circuits are all satisfying these three properties. Yao's garbled circuit can be only proved secure in the semi-honest model, and not secure in the malicious model. By using "cut and choose" technique, Yao's garbled circuit can be strengthened to be secure in the malicious model. For Yao's garbled circuit is very efficient in practice, even using some optimised "cut and choose" technique, the strengthened Yao's garbled circuits are still efficient for many applications (Lindell, 2013).

Research on the implementation of Yao's garbled circuits. After Yao proposed the garbled circuit, many researchers thought they are impractical for many AND/OR/XOR gates in circuits, which mean many encryption/decryption times, although this technique gave a firm foundation for secure two-party or multi-party computation. But in 2004, Malkhi et al. gave a breakthrough result, namely, they implemented a very practical two-party secure computation system called Fairplay. The high efficiency owns to the only using of block ciphers/hash functions in garbled circuit. After that, many optimised techniques have been proposed, such as the free-XOR technique (Kolesnikov and Schneider, 2008), the Garbled Row Reduction technique (Pinkas et al., 2009), the flexible free-XOR technique (Kolesnikov, 2014), etc. Huang et al. (2011) implemented a very impressive garbled circuit system by utilising the parallel technique when generating the garbled circuits. In 2013, Bellare et al. designed a new cipher tailored for implementing garbled circuits and thus greatly improved the efficiency. As of today, we can safely say garbled circuits are among the most practical techniques to implement complex and natural secure computation task, such as data mining, signal processing, etc.

Research on the garbled circuits used for outsourcing computation. Gennaro et al. (2010) first explored how to use garbled circuits for verifiable outsourcing computation. A lot of work has been done after that, in recent years garbled circuits have been used for implementing secure outsourcing computation for mobile phones, practical secure multi-party computation, privacy-preserving data mining etc. In 2013, Carter et al.

implemented a very practical secure outsourcing computation system on mobile phone based on garbled circuits, since then, they also improve their work in several aspects.

Research on the reusable garbled circuits. Although the efficiency of Yao's garbled circuits is very high, but it cannot be reusable, and this heavily restricts its application. Goldwasser et al. (2013) first solved the open problem of constructing reusable garbled circuits, which was left open for almost 30 years, they relied on fully homomorphic encryption and functional encryption to solve the problem. Boneh et al. (2014) proposed a new concept named fully key homomorphic encryption and constructed more efficient reusable garbled circuits with small size. However, these two works are more of theoretical interest than practical implementation. Mood et al. (2014) discussed how to reuse part of the garbled circuit generation result to improve the practical efficiency of garbled circuit, this work is very interesting from a practical point of view, but it is not fully reusable.

Research on the fully homomorphic encryption. In 1978, Rivest proposed the concept of "private bank", which can be seen as a dream about fully homomorphic encryption and its application. Gentry (2009) first constructed a concrete fully homomorphic encryption based on ideal lattice. In his proposal, Gentry proposed a new way called Gentry's blueprint to construct FHE scheme: Bootstrapping + Squashing. Van Dijk et al. (2010) proposed a new fully homomorphic encryption scheme based on integers, establishing its hardness based on LWE problem and subset sum hard problem, this scheme is denoted as DGHV11, but this scheme has been proved to be insecure in 2014. Brakerski and Vaikuntanathan (2011) based on only LWE assumption constructed a FHE scheme called BV11, in this work he introduced two new techniques: the re-linearisation technique and the dimension-modulus reduction technique. Later, Brakerski et al. (2012) proposed the BGV scheme, in this scheme before every gate evaluation, they first use the modulus reduction technique, thus reduce the noise amplification from exponential to linear. Furthermore, this scheme uses many other optimisation techniques and establish its security on the ring LWE hard problem, thus it is a highly efficient scheme. Later Gentry et al. (2012) proposed a new scheme named GHS, which can have a relatively large plaintext space. But as the BGV scheme, GHS scheme also needs to setup a maximal depth of the computed circuits when running. Gentry et al. (2013) gave the first identity or attribute based fully homomorphic encryption. Halevi and Shoup (2014) developed a software library named Helib to implement FHE schemes based on NTL library. A notable work on connecting FHE and GC is Gentry's i-hop homomorphic encryption (Gentry et al., 2010), but this work has no intention to construct FHE on GC.

3 Reusable garbled gate

3.1 Reusable garbled gate for AND

By carefully observing the running of Yao's garbled circuits, we find that the garbled gate play an important role for garbled circuit generation. Instead of aiming at reusing the garbled circuit, what results we can achieve if the garbled gates can be reused? Surprisingly, if we can construct reusable garbled gates, then many wonderful goals can be achieved, such as fully homomorphic encryption and reusable garbled circuits.

Our idea on reusable garbled gate can be seen from Figures 3 and 4. Figure 3 describes a garbled gate for AND. In Figure 2, $\{A, B\}$ and $\{C, D\}$ represent the random keys for the two input wires, $\{X, Y\}$ represent the random keys for the output wire. $\{A, C,$

$X\}$ is coloured with green, which can be seen as bit 0; $\{B, D, Y\}$ is coloured with blue, which can be seen as bit 1. $\{A, B\}$, $\{C, D\}$, $\{X, Y\}$ are all indistinguishable for the evaluators (receivers), in other words, the evaluators cannot deduce the underlying bit content from $\{A, B\}$, $\{C, D\}$, $\{X, Y\}$. The garbled table consists of four boxes, all the four boxes are locked by two lockers, and the contents in the boxes are the random keys for the output wire. In Figure 3, the boxes from up to down are locked by [*blue locker (type-1 locker), blue locker (type-1 locker)*], *[green locker (type-0 locker), blue locker (type-1 locker)], [blue locker (type-1 locker), green locker (type-0 locker)*], *[green locker (type-0 locker), green locker (type-0 locker)*]. If and only if the colours of the random keys for the two input wires are the same with the colours of lockers, the lockers on the box can be simultaneously opened and output the correct random key for the output wire. Also if and only if the types of the random keys for the two input wires are the same with the types of lockers, the lockers on the box can be simultaneously opened and output the correct the random key for the output wire.

Figure 3 Yao's garbled gate for AND (see online version for colours)

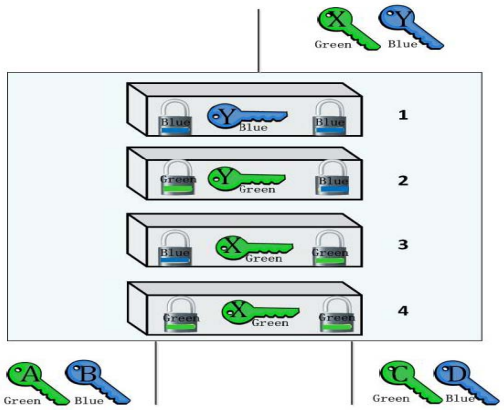
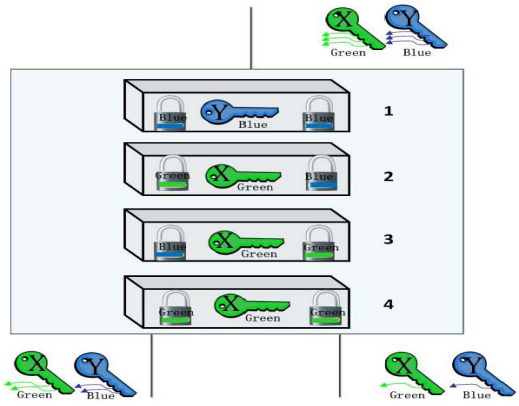


Figure 4 Reusable garbled gate for AND (see online version for colours)



Example 3.1: For example, when the input random keys are $\{A, D\}$, only boxes 2,3 can be opened, and the output random key is X. For another example, when the input random keys are $\{B, D\}$, only boxes 1 can be opened, and the output random key is Y.

Figure 4 describes a reusable garbled gate for AND. The main difference between Figure 4 and Figure 3 is the following. In Yao's garbled gate, the random keys corresponding to the two input wires are $\{A, B\}$, $\{C, D\}$, and the random keys corresponding to the output wire are $\{X, Y\}$, the secrets locked in the boxs are $\{X, Y\}$. In our reusable garbled gate, the random keys corresponding to the two input wires are $\{X1, Y1\}$, $\{X2, Y2\}$, and the random keys corresponding to the output wire are $\{X3, Y3\}$, while the secrets locked in the boxs are $\{X, Y\}$. $\{X1, X2, X3\}$ are the randomisation of X which correspond to bit 0, and $\{Y1, Y2, Y3\}$ are the randomisation of Y which correspond to bit 1. Note here can $\{X1, X2, X3\}$ act as X, and it can unlock the locker locked by X; also $\{Y1, Y2, Y3\}$ can act as Y, and it can unlock the locker locked by Y. Also note here the output $\{X3, Y3\}$ are generated freshly and differently every time run the reusable garbled gate, only in this way we can ensure the input random keys are fresh and leak no information about its underlying bits. As in the Yao's garbled circuits, all the $\{X, X1, X2, X3, \cdots, Y, Y1, Y2, Y3, \cdots, \}$ are indistinguishable to the evaluators. Also the garbled table consists of four boxes, all the four boxes are locked by two lockers, and the contents in the boxes are the *source* random keys for the output wire. In Figure 4, the boxes from up to down are locked by *[blue locker (type-1 locker), blue locker (type-1 locker)*], *[green locker (type-0 locker), blue locker (type-1 locker)], [blue locker (type-1 locker), green locker (type-0 locker)*], *[green locker (type-0 locker), green locker (type-0 locker)]*. If and only if the colours of the random keys for the two input wires are the same with the colours of lockers, the lockers on the box can be simultaneously opened and output the *randomisation* of the correct key for the output wire. Also if and only if the types of the random keys for the two input wires are the same with the types of lockers, the lockers on the box can be simultaneously opened and output the *randomisation* of the correct key for the output wire.

Example 3.2: For example, when the input random keys are $\{X2, Y1\}$, only boxes 2,3 can be opened, and the output is the randomisation of the correct key X, which is $X3$. For another example, when the input random keys are $\{Y2, Y1\}$, only boxes 1 can be opened, and the output is the randomisation of the correct key Y, which is $Y3$. Note here when $\{X3, Y3\}$ is then used as the input random keys, only boxes 2,3 can be opened, and the output is a new randomisation of the correct key X, which can be denoted as $X4$.

Figure 4 can be taken as reusable garbled AND gate, similarly there can exist reusable garbled OR gate and XOR gate, these three kinds of gate are enough to describe any function, for XOR/AND/OR can be combined to express any function.

3.2 A somewhat formal definition of reusable garbled gate for AND

Definition 3.3: We first define a Rand algorithm, which takes as inputs (inputbit, Trapdoor), and then maps the input bit to be a random key by using Trapdoor, it can be described as following: $0 \rightarrow$ *Rand(0) and* $1 \rightarrow$ *Rand(1). Note every time we invoke the Rand algorithm, the output random key is afresh one, that is, the output random keys are all different. Then we give a somewhat formal definition of Reusable Garbled Gate (RGG) for AND, it consists of the following algorithms:*

1 Generation (*AND, Input*l, *Input*2, *Trapdoor*). *This algorithm running by the sender (or data owner) generates the garbled table for AND gate and the garbled inputs for input wire 1 and input wire 2. Like Yao's garbled gate, the sender (or data owner) first map the input bits* {00,01,10,11} *to be four random key pairs by using the Trapdoor. Then the sender constructs four locked boxes: each box is locked by two lockers and hides a secret key which is a source key for the output wire (this process can be seen as encrypting the secret output source key with two input random keys by the sender). And the order of them from up to down can be arbitrarily mixed (this process can be seen as mixing the ciphertexts). Finally the sender sends the garbled gate for AND and garbled inputs to the evaluator.*

2 Evaluation *(AND, InputKey1, InputKey2). After obtaining the random key pairs and the table consists of four ciphertexts for garbled gate AND, the evaluator runs this algorithm to obtain the random output key for this gate. The running results can be seen in Table 1. Note in this algorithm the evaluation process can be decrypting and randomising the table consists of four ciphertexts. Finally this algorithm returns the OutputKey to the sender (or data owner).*

3. Recover (*AND, OutputKey, Trapdoor). After obtaining the Outputkey, the sender recovers the output bit by using the Trapdoor. This process can be described as following: Rand*(0) $\rightarrow$ *0 and Rand*(1) $\rightarrow$ 1.

Table 1 Reusable garbled gate for AND

InputKey1	*InputKey2*	*OutputKey*
Rand(1)	Rand(1)	Rand(1)
Rand(1)	Rand(0)	Rand(0)
Rand(0)	Rand(1)	Rand(0)
Rand(0)	Rand(0)	Rand(0)

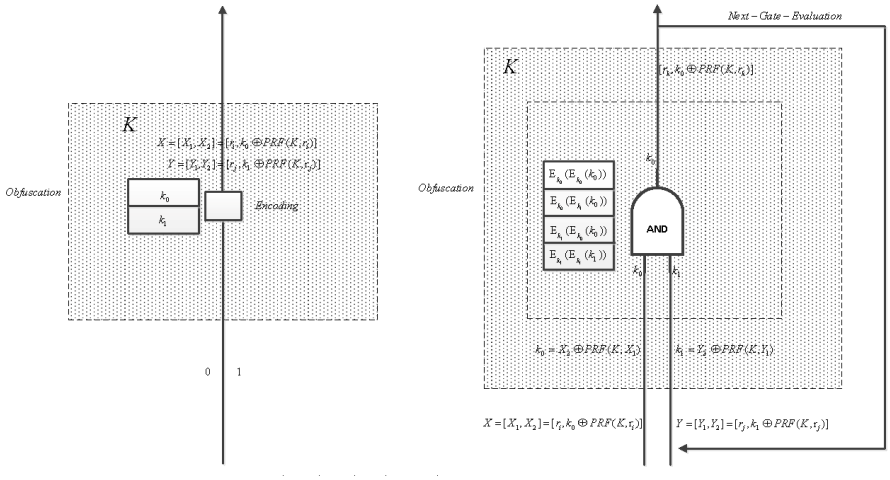
Similarly, we can give a somewhat formal definition of Reusable Garbled Gate (RGG) for XOR and OR.

Remark 3.4: Here we emphasise that this definition is just the "gold" goal we want to achieve, indicating no way on how to construct such "gold" reusable garbled gates. If such "gold" reusable garbled gates can be easily achieve, then it is also easy to construct fully homomorphic encryption, while we know this is not the fact until now. So such "gold" reusable garbled gates are difficult to be instantiated. But we also emphasis here that such concept of "gold" reusable garbled gates is already valuable, which can guide us to a new way on constructing fully homomorphic encryption.

Remark 3.5: This table of four ciphertexts can be omitted if the data owners and evaluator can directly simulate the process of Generation, Evaluation and Recover. For example, proxy re-encryption maybe can be used to simulate the process of Evaluation without any operation of Decryption. In our point of view, searchable encryption with probabilistic tokens or ORAM techniques maybe can be used to construct such "gold" reusable garbled gates. Generally we can view these "gold" reusable garbled gates as the encoding-evaluating-decoding process. We also note here for supporting arbitrary computation, the outputs of the *Rand* algorithm must support AND, OR, XOR operation simultaneously. It is no use if this encoding process only support one type of operation,

and the key challenge is supporting AND, OR, XOR operation simultaneously for this encoding-evaluating-decoding process.

Figure 5 An example of construction based on the idea of obfuscation



Example 3.6: Here we try to give an example of construction based on the idea of obfuscation, which can be seen in the Figure 5. We describe a very simple evaluation of an AND gate by the evaluator.

1 First the data owner constructs an obfuscated program of input encoding or input transformation (the left figure), and an obfuscated program of gate evaluation or decryption of reusable garbled gates (the right figure), which embedded a key K for pseudorandom function PRF. Then he publishes these obfuscated program as the public keys. For example, data owner's input bits are 0 and 1, he can encode them as $[r_i, k_0 \oplus PRF(K, r_i)]$ and $[r_j, k_1 \oplus PRF(K, r_j)]$, and outsource them to the evaluator.

2 After obtaining the input keys, the evaluator can invoke the obfuscated program of gate evaluation to get the output key $[r_k, k_0 \oplus PRF(K, r_k)]$. This output key then is returned to the data owner (Also note here the output key can be used as an input for next-gate-evaluation if the evaluated function is more complex than a simple AND gate).

3 After obtaining the output key, the data owner can recover the output bit by first computing $k_0 = k_0 \oplus PRF(K, r_k) \oplus PRFPRK(K, r_k)$ and then mapping this secret key k_0 to bit 0.

Remark 3.7: Roughly speaking, the security of this construction heavily relies on the security of obfuscation, that is, the evaluator can implement the evaluating without knowing the underlying secret key K, k_0, k_1 by just invoking the obfuscated program. Fortunately, now we can have some positive and concrete construction on indistinguishable obfuscation, such as in Miles et al. (2016).

Definition 3.8: *Here we give the properties which RGG should satisfy. Roughly speaking, a RGG for AND needs to satisfy the following properties:*

1 *First, keys representing 0 (we assume they are keys contain X in Figure 4) and keys representing 1 (we assume they are keys contain Y in Figure 4) should be indistinguishable for the evaluator;*

2 *Second, keys representing 0 should be indistinguishable among themselves, keys representing 1 should also be indistinguishable among themselves;*

3 *Third, keys representing 0 should be able to open the lock locked by keys representing 0, keys representing 1 should be able to open the lock locked by keys representing 1;*

4 *Fourth, keys representing 1 should be unable to open the lock locked by keys representing 0, keys representing 0 should be unable to open the lock locked by keys representing 1;*

5 *Fifth, the lock is circular-secure, if we think the locked box is a ciphertext, then the encryption scheme should be circular-secure;*

6 *Sixth, the two locks should have no order. For example, a locked box which has been locked by 0 and 1, can be opened by two keys representing 0 and 1, also can be opened by two keys representing 1 and 0;*

7 *Seventh, there should be existing many probabilistic keys all representing 0, also there should be many probabilistic keys all representing 1;*

8 *Eighth, if we think the locked box is a ciphertext, then the decryption algorithm should be probabilistic, that is, any two successful decryption to keys representing 0 should output two indistinguishable different keys representing 0, and the same holds for decryption results representing 1.*

Similarly, it is easy to derive the properties the RGG for OR, XOR should satisfy, here we omit it.

4 FHE and RGC based on RGG

4.1 FHE based on RGG

Here we describe how to construct FHE based on RGG, which can be seen in Figure 6.

1 First, the data owner constructs three kinds of reusable garbled gates: AND, OR and XOR and publish them as public parameters.

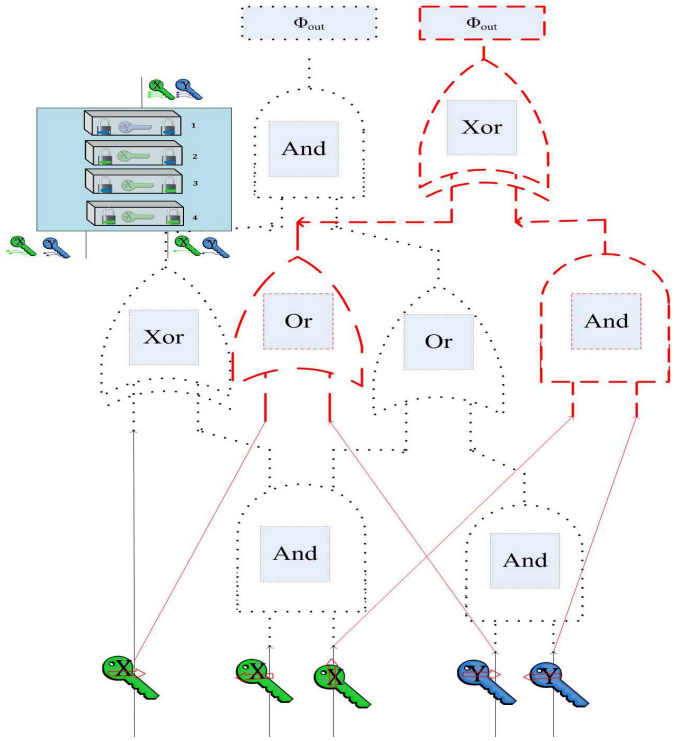
2 Second, he transforms his plaintext bits $(x_1, x_2, \cdots, x_n)$ to be the keys corresponding to $(x_1, x_2, \cdots, x_n)$, which is actually the ciphertexts. He also outsource the input ciphertexts to the evaluator (can be the cloud).

3 Third, the evaluator chooses an arbitrary function *f* and then transform it to be a circuit.

4 Fourth, the evaluator reconstruct the Yao-style circuit from reusable garbled gates. After obtaining the input bits' corresponding input ciphertexts, it runs the reusable garbled gates and thus can get the final output. Finally it returns this output to the data owner.

5 Fifth, the final output are also keys representing 0 or 1, which are not known by the evaluator, but the data owner can recover the results by using trapdoor which maps the keys to 0 or 1.

Figure 6 FHE based on RGG (see online version for colours)



4.2 RGC based on RGG

Here we describe how to construct RGC based on RGG, which can be seen in Figure 7.

1 First, the data owner constructs three kinds of reusable garbled gates: AND, OR and XOR and publish them as public parameters.

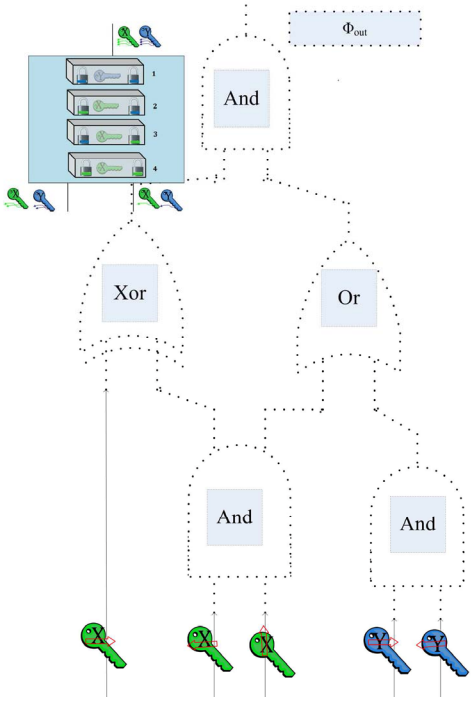
2 Second, he also transforms his plaintext bits $(x_1, x_2, \cdots, x_n)$ to be the keys corresponding to $(x_1, x_2, \cdots, x_n)$, which is actually the ciphertexts. Assume the function corresponding to the reusable garbled circuits is f, he outsources the input ciphertexts and f to the evaluator (can be the cloud).

3 Third, the evaluator transforms function f to be a circuit.

4 Fourth, the evaluator reconstruct the Yao-style reusable garbled circuit from reusable garbled gates and the input ciphertexts, it runs the reusable garbled gates and thus can get the final output. Finally it returns this output to the data owner.

5 Fifth, the final output are also keys representing 0 or 1, which are not known by the evaluator, but the data owner can recover the results by using trapdoor which map the keys to 0 or 1.

Figure 7 RGC based on RGG (see online version for colours)



5 A concrete construction for CSPLE Based on iO

5.1 Construction

In this section we give a concrete construction for CSPLE based on indistinguishable obfuscation (iO) (Sahai and Waters, 2014). Let PRG be a pseudo-random generator that maps $\{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$. Let *PRF* be a puncturable PRF that takes inputs of 2λ bits and outputs l bits. We describe our CSPLE schemes as following:

1 Setup (1^{λ}): The setup algorithm first chooses a puncturable key K_0 for *PRF*, a PRF key K_1^0 for *PRF*, a PRF key K_1^1 for *PRF*. Next, it creates the reusable garbled gates for AND, OR and XOR gates in Figure 10, it creates an obfuscation of the Program Input Transformation of Figure 8, it creates an obfuscation of the Program Decryption of Reusable Garbled Gates of Figure 9. Note here the size of Program Input Transformation of 8 is padded to the maximum of itself and Program Input Transformation* of Figure 12, and the size of Program Decryption of Reusable

Garbled Gates of Figure 9 is padded to the maximum of itself and Program Decryption of Reusable Garbled Gates* of Figure 13. The public key, PK, is the obfuscated programs in Figure 8, Figure 9 and the reusable garbled gates in Figure 10. The secret key *SK* is $K_0,\ K_1^0,\ K_1^1$.

2 Input Transformation: On input message $M=(m_1,m_2,\ldots,m_n),$ for every m_i the user (including data owner and any others) runs the obfuscated program of Input Transformation in Figure 8 with every fresh *r* chosen randomly from $\{0,1\}^{\lambda}$. Finally he outputs $Keys=(X^1,X^2,\ldots,X^n)$ which representing the ciphertexts for his messages.

3 Decryption of Reusable Garbled Gates: On input ciphertext *C* and garbled keys $X=(X_1,X_2),Y=(Y_1,Y_2),$ the evaluator runs the obfuscated program of Decryption of Reusable Garbled Gates in Figure 9 with every fresh *r* chosen randomly from $\{0,1\}^{\lambda}$. Finally he outputs $Key=X_{Out}$ as the output.

4 Evaluation of Any Function: By repeated run Decryption of Reusable Garbled Gates algorithm with any two inputs for AND, OR, XOR gates, the evaluator can run arbitrary function on the ciphertexts and get the corrected result.

5 Recover Result: By using trapdoors $K_0,K_1^0,K_1^1,$ the user can map the keys (the ciphertexts) to the plaintext bits, thus he can recover the correct computation result.

Remark 5.1: Note here we although simulate Yao-style garbled circuit, but our construction has an important difference with them, that is, our protocol can not achieve the authenticity property for computation result (Bellare et al., 2012). But we point out fully homomorphic encryption does not include authenticity property as its basic requirement, that is, Gentry's FHE also do not guarantee authenticity property for computation result (Gentry, 2009).

5.2 Correctness and security analysis

Correctness. As an example, we derive the correctness for AND gate as Figure 11. Similarly we can verify the correctness for other gates OR, XOR, here we omit it. By iteratively invoking the gates for AND/OR/XOR corresponding to the structure of the needed computed function's circuit, we can easily obtain the final correct computation result.

Figure 8 Program input transformation

Input Transformation

Constants: Punctured PRF Key K_0, PRF Key K_1^0, K_1^1.
Input: message $m \in \{0,1\}$.

(a) Select a randomness $r \in \{0,1\}^{\lambda}$, and let $t = PRG(r)$;

(b) if $m = 0$, output garbled keys $X = (X_1, X_2) = (t, PRF(K_0, t) \oplus K_1^0)$;

(c) otherwise $m = 1$, output garbled keys $X = (X_1, X_2) = (t, PRF(K_0, t) \oplus K_1^1)$;

Figure 9 Program decryption of reusable garbled gates

Decryption of Reusable Garbled Gates

Constants: Punctured PRF Key K_0, PRF Key K_1^0, K_1^1.
Input: Ciphertext $C = (C_1, C_2, C_3)$, garbled keys $X = (X_1, X_2), Y = (Y_1, Y_2)$.

(a) Compute $K_1^X = PRF(K_0, X_1) \oplus X_2$;

(b) Compute $K_1^Y = PRF(K_0, Y_1) \oplus Y_2$;

(c) Compute $K_1^{Out} = PRF(K_1^X, C_1) \oplus PRF(K_1^Y, C_2) \oplus C_3$ or $K_1^{Out} = PRF(K_1^X, C_2) \oplus PRF(K_1^Y, C_1) \oplus C_3$ (One of the results will be correct for all AND, OR and XOR gates, the two input bits can be exchanged without changing the output result);

(d) Select a randomness $r \in \{0,1\}^\lambda$, and compute $t = PRG(r)$;

(e) Output garbled keys $X_{Out} = (X_{Out1}, X_{Out2}) = (t, PRF(K_0, t) \oplus K_1^{Out})$;

Figure 10 Reusable garbled gates

Reusable Garbled Gates Generation: Here we describe four ciphertexts representing reusable garbled gates AND:

$$C_{00}^0 = (C_{00}^{0a}, C_{00}^{0b}, C_{00}^{0c}) = (t_1, t_1', PRF(K_1^0, t_1) \oplus PRF(K_1^0, t_1') \oplus K_1^0)$$
$$C_{01}^0 = (C_{01}^{0a}, C_{01}^{0b}, C_{01}^{0c}) = (t_2, t_2', PRF(K_1^0, t_2) \oplus PRF(K_1^1, t_2') \oplus K_1^0)$$
$$C_{10}^0 = (C_{10}^{0a}, C_{10}^{0b}, C_{10}^{0c}) = (t_3, t_3', PRF(K_1^1, t_3) \oplus PRF(K_1^0, t_3') \oplus K_1^0)$$
$$C_{11}^1 = (C_{11}^{1a}, C_{11}^{1b}, C_{11}^{1c}) = (t_4, t_4', PRF(K_1^1, t_4) \oplus PRF(K_1^1, t_4') \oplus K_1^1)$$

where $t_1, t_1', t_2, t_2', t_3, t_3', t_4, t_4'$ are generated by $PRG(r)$ for every fresh r chosen randomly from $\{0,1\}^\lambda$. The Data Owner also permutates this four ciphertexts' position randomly and publishes them as public key.
Similarly we can construct four ciphertexts representing reusable garbled gates OR:

$$D_{00}^0 = (D_{00}^{0a}, D_{00}^{0b}, D_{00}^{0c}) = (s_1, s_1', PRF(K_1^0, s_1) \oplus PRF(K_1^0, s_1') \oplus K_1^0)$$
$$D_{01}^1 = (D_{01}^{1a}, D_{01}^{1b}, D_{01}^{1c}) = (s_2, s_2', PRF(K_1^0, s_2) \oplus PRF(K_1^1, s_2') \oplus K_1^1)$$
$$D_{10}^1 = (D_{10}^{1a}, D_{10}^{1b}, D_{10}^{1c}) = (s_3, s_3', PRF(K_1^1, s_3) \oplus PRF(K_1^0, s_3') \oplus K_1^1)$$
$$D_{11}^1 = (D_{11}^{1a}, D_{11}^{1b}, D_{11}^{1c}) = (s_4, s_4', PRF(K_1^1, s_4) \oplus PRF(K_1^1, s_4') \oplus K_1^1)$$

where $s_1, s_1', s_2, s_2', s_3, s_3', s_4, s_4'$ are generated by $PRG(r)$ for every fresh r chosen randomly from $\{0,1\}^\lambda$. The Data Owner also permutates this four ciphertexts' position randomly and publishes them as public key.
Similarly we can construct four ciphertexts representing reusable garbled gates XOR:

$$E_{00}^0 = (E_{00}^{0a}, E_{00}^{0b}, E_{00}^{0c}) = (u_1, u_1', PRF(K_1^0, u_1) \oplus PRF(K_1^0, u_1') \oplus K_1^0)$$
$$E_{01}^1 = (E_{01}^{1a}, E_{01}^{1b}, E_{01}^{1c}) = (u_2, u_2', PRF(K_1^0, u_2) \oplus PRF(K_1^1, u_2') \oplus K_1^1)$$
$$E_{10}^1 = (E_{10}^{1a}, E_{10}^{1b}, E_{10}^{1c}) = (u_3, u_3', PRF(K_1^1, u_3) \oplus PRF(K_1^0, u_3') \oplus K_1^1)$$
$$E_{11}^0 = (E_{11}^{0a}, E_{11}^{0b}, E_{11}^{0c}) = (u_4, u_4', PRF(K_1^1, u_4) \oplus PRF(K_1^1, u_4') \oplus K_1^0)$$

where $u_1, u_1', u_2, u_2', u_3, u_3', u_4, u_4'$ are generated by $PRG(r)$ for every fresh r chosen randomly from $\{0,1\}^\lambda$. The Data Owner also permutates this four ciphertexts' position randomly and publishes them as public key.

Figure 11 An example of correctness for AND gate

An example of correctness for AND gate

Input: An pair of message corresponding to the two wires input bits 01.

1. First by invoking "Program Input Transformation" with fresh randomness r_1, r_2, the data owner can get the pair of InputKeys:
$$X = (X_1, X_2) = (t_1, PRF(K_0, t_1) \oplus K_1^0), t_1 = PRG(r_1)$$
$$Y = (Y_1, Y_2) = (t_2, PRF(K_0, t_2) \oplus K_1^1), t_1 = PRG(r_2)$$
then the data owner outsources X, Y to the evaluator.
2. After obtaining (X, Y), the evaluator can first use the WIRE SIGNAL technique to find the correct ciphertext. This technique can be seen in [23], which can be used to find the correct ciphertext in the ciphertext table, but without leaking the underlying input bits.
$$C_{01}^0 = (C_{01}^{0a}, C_{01}^{0b}, C_{01}^{0c}) = (t_2, t_2', PRF(K_1^0, t_2) \oplus PRF(K_1^1, t_2') \oplus K_1^0)$$
Denote $(C_{01}^{0a}, C_{01}^{0b}, C_{01}^{0c}) = (C_1, C_2, C_3)$
3. Then he runs the "Program Decryption of Reusable Garbled Gates" with fresh randomness r_3, and the process is the following:
   (a) Compute $K_1^0 = PRF(K_0, X_1) \oplus X_2$;
   (b) Compute $K_1^1 = PRF(K_0, Y_1) \oplus Y_2$;
   (c) Compute
$$\begin{aligned} K_1^0 &= PRF(K_1^0, C_1) \oplus PRF(K_1^1, C_2) \oplus C_3 \\ &= PRF(K_1^0, t_2) \oplus PRF(K_1^1, t_2') \oplus PRF(K_1^0, t_2) \oplus PRF(K_1^1, t_2') \oplus K_1^0; \end{aligned}$$
   (d) Compute $t_3 = PRG(r_3)$;
   (e) Output garbled keys $X_{Out} = (X_{Out1}, X_{Out2}) = (t_3, PRF(K_0, t_3) \oplus K_1^0)$;
   he returns X_{Out} to the data owner.
4. Finally the data owner recovers the output bit by using punctured key K_0 as following:
$$K_1^0 = PRF(K_0, X_{Out1}) \oplus X_{Out2}$$
which corresponding to bit 0.

Output: an output bit 0.

Figure 12 Program input transformation*

Input Transformation*

Constants: Punctured PRF Key $K_0(\{t^*\})$, PRF Key K_1^0, K_1^1.
Input: message $m \in \{0,1\}$.

1. Select a randomness $r \in \{0,1\}^\lambda$ and let $t = PRG(r)$;
2. if $m = 0$, output garbled keys $X = (X_1, X_2) = (t, PRF(K_0, t) \oplus K_1^0)$;
3. otherwise $m = 1$, output garbled keys $X = (X_1, X_2) = (t, PRF(K_0, t) \oplus K_1^1)$;

Figure 13 Program decryption of reusable garbled gates*

Decryption of Reusable Garbled Gates*

Constants: Punctured PRF Key $K_0(\{t^*\})$, PRF Key K_1^0, K_1^1.
Input: Ciphertext $C = (C_1, C_2, C_3)$, garbled keys $X = (X_1, X_2), Y = (Y_1, Y_2)$.

1. Compute $K_1^X = PRF(K_0, X_1) \oplus X_2$;
2. Compute $K_1^Y = PRF(K_0, Y_1) \oplus Y_2$;
3. Compute $K_1^{Out} = PRF(K_1^X, C_1) \oplus PRF(K_1^Y, C_2) \oplus C_3$ or $K_1^{Out} = PRF(K_1^X, C_2) \oplus PRF(K_1^Y, C_1) \oplus C_3$ (One of the results will be correct for all AND, OR and XOR gates, the two input bits can be exchanged without changing the output result);
4. Select a randomness $r \in \{0,1\}^\lambda$ and Compute $t = PRG(r)$;
5. Output garbled keys $X_{Out} = (X_{Out1}, X_{Out2}) = (t, PRF(K_0, t) \oplus K_1^{Out})$;

Security. Indistinguishable obfuscation has been proved to be secure if two obfuscated programs have the same functionality, while our design is satisfied. The puncturable pseudo random function is a key object for proving security for indistinguishable obfuscation, which we also use in our design. Concretely we describe the twin (functionally equivalent) obfuscated programs of Program Input Transformation and Program Decryption of Reusable Garbled Gates in Figures 12 and 13, which we denote as Program Input Transformation* and Program Decryption of Reusable Garbled Gates*. First we prove Program Input Transformation's security, here we can regard this transformation as a public key encryption algorithm.

Lemma 5.2: *The Program Input Transformation is IND-CPA secure if our obfuscation scheme is indistinguishably secure, PRG is a secure pseudorandom generator, and PRF is a secure punctured PRF*.

Proof. We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original IND-CPA security game. We prove that the attacker's advantage must be the negligibly close between each successive hybrid and that the attacker has zero advantage in the final experiment.

- Hyb_0 : In the first hybrid the following game is played.

  1 $r^* \in \{0,1\}$ is chosen at random and $t^* = PRG(r^*)$.

  2 K is chosen as a key for the puncturable PRF.

  3 The public key given out is an obfuscation of the Program Input Transformation.

  4 The attacker receives PK and then gives m_0, $m_1 \in \{0,1\}$ to the challenger.

  5 The challenge ciphertext (output garbled key) is $X = (X_1, X_2) = (t, PRF(K_0, t) \oplus K_1^b)$ where $b \in \{0,1\}$ is chosen randomly.

- Hyb_1: Is the same as Hyb_0 with the exception that t^* is chosen randomly in $\{0,1\}^{2\lambda}$. Note that r is no longer in the attacker's view and does not need to be generated.

- Hyb_2: Is the same as Hyb_1 except that the public key is created as an obfuscation of the Program Input Transformation of Figure 12. Note that with all but negligible probability t^* is not in the image of the PRG.
- Hyb_3: Is the same as Hyb_2 except the challenge ciphertext is given as $X = (X_1, X_2) = (t^*, z^*)$ for random z^*

We first argue that the advantage of any poly-time attacker in guessing the bit b in Hyb_1 must be negligibly close to the attacker's advantage in Hyb_0. Otherwise, we can easily create a reduction algorithm B that breaks the security of the pseudorandom generator. This conclusion can be easily derived from observing the difference between Hyb_1 and Hyb_0, we omit the details here.

Next, we argue that the advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_2 and Hyb_1. We first observe that with all but negligible probability that the input/output behaviour of Program Input Transformation and Program Input Transformation* are identical when t^* is chosen at random. The reason is that with probability $1-\frac{1}{2^\lambda}, t^*$ is not in the image on the PRG. Thus, with high probability for all inputs neither program can call on $PRF(K_0, t^*)$. Therefore, puncturing t^* out from the key K will not effect input/output behaviour. Therefore, if there is a difference in advantage, we can create an algorithm B that breaks indistinguishability security for obfuscation.

We now argue that the advantage for any poly-time attacker in guessing the bit b must be negligibly close in hybrids Hyb_3 and Hyb_2. Otherwise, we can create a reduction algorithm B that breaks the selective security of the constrained pseudorandom function at the punctured points. This conclusion can be easily derived from observing the difference between Hyb_3 and Hyb_2 and the property of punctured pseudorandom function, we omit the details here.

Finally, we observe that any attacker's advantage in Hyb_3 must be 0, since it conveys no information about b. Since the advantage of all poly-time attacker's are negligibly close in each successive hybrid, this proves IND-CPA security of Program Input Transformation.

Next we prove Program Decryption of Reusable Garbled Gates's security, here we can regard this transformation as a public key decryption-then-randomised-encryption algorithm.

Lemma 5.3: *The Program Decryption of Reusable Garbled Gates is IND-CPA secure if our obfuscation scheme is indistinguishably secure, PRG is a secure pseudorandom generator, and PRF is a secure punctured PRF.*

Proof. We describe a proof as a sequence of hybrid experiments where the first hybrid corresponds to the original IND-CPA security game. We prove that the attacker's advantage must be the negligibly close between each successive hybrid and that the attacker has zero advantage in the final experiment.

- Hyb_0 : In the first hybrid the following game is played.

  1 $r^* \in \{0,1\}$ is chosen at random and $t^* = PRG(r^*)$.

  2 K is chosen as a key for the puncturable PRF.

3 The public key given out is an obfuscation of the Program Decryption of Reusable Garbled Gates.

4 The attacker receives *PK* from the challenger.

5 The challenge ciphertext (output garbled key) is $X_{Out} = (X_{Out1}, X_{Out2}) = (t, PRF(K_0, t) \oplus K_1^b)$ where $b \in \{0,1\}$ is chosen randomly.

- Hyb_1: Is the same as Hyb_0 with the exception that t^* is chosen randomly in $\{0,1\}^{2\lambda}$. Note that r is no longer in the attacker's view and does not need to be generated.
- Hyb_3: Is the same as Hyb_2 except the challenge ciphertext is given as $X = (X_1, X_2) = (t^*, z^*)$ for random z^*

Similarly as the above lemma, we can obtain the following results: any attacker's advantage in Hyb_3 must be 0, since it conveys no information about b. Since the advantage of all poly-time attacker's are negligibly close in each successive hybrid, thus Program Decryption of Reusable Garbled Gates is IND-CPA secure.

Theorem 5.4: *If the underlying indistinguishable obfuscation of "Program Input Transformation" and "Program Decryption of Reusable Garbled Gates" are secure and the puncturable pseudo random function is secure, then our construction of RGG for AND, OR, XOR is secure*.

Proof. Here we give a very roughly security proof. The idea of our construction is following. First we use the "Program Input Transformation" to transform the input bit to be a random InputKey, the data owner can directly implement this transformation without relying on indistinguishable obfuscation. However others besides the data owner can not complete this transformation, thus we use the indistinguishable obfuscation of "Program Input Transformation" to support anyone's ability of encoding input bits. Second the data owner constructs three tables of four ciphertexts as the gate for AND/OR/XOR, just like the garbled tables in Yao's garbled circuits. Third we use indistinguishable obfuscation of "Program Decryption of Reusable Garbled Gates" to complete the decrypt-then-randomise operation, which is the key feature for RGG. If we just use the decryption algorithm, then the output key cannot be random which does not satisfy the RGG's properties. By combining the above two lemmas, we can conclude our design is secure if the underlying indistinguishable obfuscation of "Program Input Transformation" and "Program Decryption of Reusable Garbled Gates" are secure and the puncturable pseudo random function is secure.

Remark 5.5: In Program Decryption of Reusable Garbled Gates, $K_1^{Out} = PRF(K_1^X, C_1) \oplus PRF(K_1^Y, C_2) \oplus C_3$ or $K_1^{Out} = PRF(K_1^X, C_2) \oplus PRF(K_1^Y, C_1) \oplus C_3$. These results maybe not always correct for the invalid input $X = (X_1, X_2) Y = (Y_1, Y_2)$, but we can use other authentication techniques to ensure this output's correctness. Note traditional fully homomorphic encryption schemes also has no this authentication property either.

Remark 5.6: One may argue that the evaluator can know the intermediated computation result when running Program Decryption of Reusable Garbled Gates, such as $K_1^X = PRF(K_0, K_1) \oplus X_2$ and $K_1^Y = PRF(K_0, Y_1) \oplus Y_2$. But we emphasis here this is not possible for the indistinguishable obfuscation is as strong as the best possible obfuscation

(Goldwasser and Rothblum, 2007), which could hide the intermediate computation results. Otherwise it cannot hide the secret constant embedded in the obfuscated program either, while almost all concrete constructions of indistinguishable obfuscation having this property.

6 Conclusion

In this paper, for the first time we try to bridge the gap between fully homomorphic encryption and Yao's garbled circuit. For this purpose, we propose an interesting new primitive: reusable garbled gate and show how to easily provide fully homomorphic encryption and reusable garbled circuit service based on it. However, the instantiation of this primitive is very difficult. We can only give an instantiation based on the recently new concept of indistinguishable obfuscation, which is not efficient until now. There are many interesting open problems deserved further exploring such as:

1 The most important open problem is how to instantiate reusable garbled gate for AND, XOR, OR efficiently? Can it be efficient instantiated by symmetric primitives like DES or AES?

2 Can the indistinguishable obfuscation just be used in the Setup phase instead of using in every running of reusable garbled gate? If we can achieve this, then the computation cost for running fully homomorphic encryption and reusable garbled circuit will be reduced greatly.

3 Can we rely on some semi-trusted party like cloud to implement the primitive of reusable garbled gate for AND, XOR, OR with few rounds of interaction with data owners, while keeping this party not knowing the underlying bits? Or can FHE with any relaxation model be efficiently instantiated by using the primitive of reusable garbled gate, such as relying on two un-colluding servers for implementing the FHE operation? We think these are also deserved research topics.

4 Can we instantiate the reusable garbled gate by leveraging the techniques such as proxy re-encryption, two-to- one encoding, searchable encryption with probabilistic tokens and ORAM? Any such instantiation will bring us the hope of construction of fully homomorphic encryption based on various mathematica tools besides lattice.

Acknowledgements

The authors would like to express their gratitude thanks to Prof. Mingwu Zhang, Prof. Jian Weng and Dr. Baodong Qin for many helpful comments. This work is supported by Natural Science Foundation of Shaanxi Province (Grant No. 2014JM8300, 2014JQ8358, 2014JQ8307), the Changjiang Scholars and Innovation Research Team in University (Grant No. IRT 1078), the Key Project of NFSC-Guangdong Union Foundation (Grant No. U1135002), the Major Nature Science Foundation of China (Grant No. 61370078), China 863 project (Grant No. 2015AA016007), the Fundamental Research Funds for the Center Universities (Grant No. JY10000903001), Cross-Straits Science Foundation (Grant No. U1405255), Nature Science Foundation of China (Grant No. 61572721, 61572390), China 111 project (B08038).

References

Bellare, M., Hoang, V.T. and Rogaway, P. (2012) 'Foundations of garbled circuits', in Yu, T., Danezis, G. and Gligor, V.D. (Eds): *ACM CCS 12*, Raleigh, NC, USA, 16–18 October, ACM Press, pp.784–796.

Bellare, M., Hoang, V.T., Keelveedhi, S. and Rogaway, P. (2013) 'Efficient garbling from a fixed-key blockcipher', *2013 IEEE Symposium on Security and Privacy*, Berkeley, California, USA, 19–22 May, IEEE Computer Society Press, pp.478–492.

Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V. and Vinayagamurthy, D. (2014) 'Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits', in Nguyen, P.Q. and Oswald, E. (Eds): *EUROCRYPT2014,* volume 8441 of *LNCS*, Copenhagen, Denmark, 11–15 May, Springer, Berlin, Germany, pp.533–556.

Brakerski, Z. and Vaikuntanathan, V. (2011) 'Efficient fully homomorphic encryption from (standard) LWE', in Ostrovsky, R. (Ed.): *52nd FOCS*, Palm Springs, California, USA, 22–25 October, pp.97–106.

Brakerski, Z., Gentry, C. and Vaikuntanathan, V. (2012) '(Leveled) fully homomorphic encryption without bootstrapping', *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pp.309–325.

Carter, H., Mood, B., Traynor, P. and Butler, K.R.B. (2013) 'Secure outsourced garbled circuit evaluation for mobile devices', *USENIX Security Symposium*.

Chen, X., Li, J., Ma, J., Tang, Q. and Lou, W. (2012) 'New algorithms for secure outsourcing of modular exponentiations', in Foresti, S., Yung, M. and Martinelli, F. (Eds): *ESORICS 2012, volume 7459 of LNCS*, Pisa, Italy, 10–12 September, pp.541–556.

Chen, X., Li, J., Weng, J., Ma, J. and Lou, W. (2014) 'Verifiable computation over large database with incremental updates', *ESORICS*, pp.148–162.

Dutu, C., Apostol, E., Leordeanu, C. and Cristea, V. (2014) 'A solution for the management of multimedia sessions in hybrid clouds', *International Journal of Space-Based and Situated Computing*, Vol. 4, No. 2, pp.77–87.

Gennaro, R., Gentry, C. and Parno, B. (2010) 'Non-interactive verifiable computing: outsourcing computation to untrusted workers', in Rabin, T. (Ed.): *CRYPTO 2010, volume 6223 of LNCS*, Santa Barbara, CA, USA, 15–19 August, pp.465–482.

Gentry, C. (2009) 'Fully homomorphic encryption using ideal lattices', in Mitzenmacher, M. (Ed.): *41st ACM STOC*, Bethesda, Maryland, USA, May 31 – June 2, pp.169–178.

Gentry, C., Halevi, S. and Smart, N.P. (2012) 'Fully homomorphic encryption with polylog overhead', in Pointcheval, D. and Johansson, T. (Eds): *EUROCRYPT 2012, volume 7237 of LNCS*, Cambridge, UK, 15–19 April, pp.465–482.

Gentry, C., Halevi, S. and Vaikuntanathan, V. (2010) 'i-Hop homomorphic encryption and rerandomizable Yao circuits', in Rabin, T. (Ed.): *CRYPTO 2010, volume 6223 of LNCS*, Santa Barbara, CA, USA, 15–19 August, pp.155–172.

Gentry, C., Sahai, A. and Waters, B. (2013) 'Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based', in Canetti, R. and Garay, J.A. (Eds): *CRYPTO 2013, Part I, volume 8042 of LNCS*, Santa Barbara, CA, USA, 18–22 August, pp.75–92.

Goldwasser, S. and Rothblum, G.N. (2007) 'On best-possible obfuscation', in Vadhan, S.P. (Ed.): *TCC 2007, volume 4392 of LNCS*, Amsterdam, The Netherlands, 21–24 February, pp.194–213.

Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V. and Zeldovich, N. (2013) 'Reusable garbled circuits and succinct functional encryption', in Boneh, D., Roughgarden, T. and Feigenbaum, J. (Eds): *45th ACM STOC*, Palo Alto, CA, USA, 1–4 June, pp.555–564.

Guo, S. and Xu, H. (2015) 'A secure delegation scheme of large polynomial computation in multi-party cloud', *International Journal of Grid and Utility Computing*, Vol. 6, No. 2, pp.1–7.

Halevi, S. and Shoup, V. (2014) 'Algorithms in HElib', *Cryptology ePrint Archive*, Report 2014/106, 2014. Available online at: http://eprint.iacr.org/2014/106

Huang, Y., Evans, D., Katz, J. and Malka, L. (2011) 'Faster secure two-party computation using garbled circuits', *USENIX Security Symposium*.

Kolesnikov, V. and Schneider, T. (2008) 'Improved garbled circuit: Free XOR gates and applications', in Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A. and Walukiewicz, I. (Eds): *ICALP 2008, Part II, volume 5126 of LNCS*, Reykjavik, Iceland, 7–11 July, pp.486–498.

Kolesnikov, V., Mohassel, P. and Rosulek, M. (2014) 'FleXOR: Flexible garbling for XOR gates that beats free-XOR', in Garay, J.A. and Gennaro, R. (Eds): *CRYPTO 2014, Part II, volume 8617 of LNCS*, Santa Barbara, CA, USA, 17–21 August, pp.440–457.

Lindell, Y. (2013) 'Techniques for efficient secure computation based on Yao's protocol', in Kurosawa, K. and Hanaoka, G. (Eds): *PKC 2013, volume 7778 of LNCS*, Nara, Japan, February 26 – March 1, p.253.

Lindell, Y. and Pinkas, B. (2009) 'A proof of security of Yao's protocol for two-party computation', *Journal of Cryptology*, Vol. 22, No. 2, pp.161–188.

Malkhi, D., Nisan, N., Pinkas, B. and Sella, Y. (2004) Fairplay – secure two-party computation system', *USENIX Security Symposium*, pp.287–302.

Miles, E., Sahai, A. and Zhandry, M. (2016) 'Secure obfuscation in a weak mutilinear map model: a simple construction secure against all known attacks', *Cryptology ePrint Archive*, Report 2016/588. Available online at: http://eprint.iacr.org/2016/588

Mood, B., Gupt, D., Butler, K.R.B. and Feigenbaum, J. (2014) 'Reuse it or lose it: more efficient secure computation through reuse of encrypted values', *ACM CCS*, pp.582–596.

Pinkas, B., Schneider, T., Smart, N.P. and Williams, S.C. (2009) 'Secure two-party computation is practical', in Matsui, M. (Ed.): *ASIACRYPT 2009, volume 5912 of LNCS*, Tokyo, Japan, 6–10 December, pp.250–267.

Sahai, A. and Waters, B. (2014) 'How to use indistinguishability obfuscation: deniable encryption, and more', in Shmoys, D.B. (Ed.): *46th ACM STOC*, New York, NY, USA, May 31 – June 3, pp.475–484.

Thabet, M., Boufaida, M. and Kordon, F. (2014) 'An approach for developing an interoperability mechanism between cloud providers', *International Journal of Space-Based and Situated Computing*, Vol. 4, No. 2, pp.88–99.

van Dijk, M., Gentry, C., Halevi, S. and Vaikuntanathan, V. (2010) 'Fully homomorphic encryption over the integers', in Gilbert, H. (Ed.): *EUROCRYPT 2010, volume 6110 of LNCS*, French Riviera, May 30 – June 3, pp.24–43.

Wang, Y., Du, J., Cheng, X., Liu, Z. and Lin, K. (2016) 'Degradation and encryption for outsourced png images in cloud storage', *International Journal of Grid and Utility Computing*, Vol. 7, No. 1, pp.22–28.

Yao, A.C-C. (1982) 'Theory and applications of trapdoor functions (extended abstract)', *23rd FOCS*, Chicago, Illinois, 3–5 November, pp.80–91.

Yao, A.C-C. (1986) 'How to generate and exchange secrets (extended abstract)', *27th FOCS*, Toronto, Ontario, Canada, 27–29 October, pp.162–167.

Zhu, S. and Yang, X. (2015) 'Protecting data in cloud environment with attribute-based encryption', *International Journal of Grid and Utility Computing*, Vol. 6, No. 2, pp.91–97.