*future internet*

MDPI

*Article*

# Supporting Privacy of Computations in Mobile Big Data Systems †

**Sriram Nandha Premnath [1] and Zygmunt J. Haas [2,3,*]**

[1] Qualcomm Research, Santa Clara, CA 95051, USA; sriramnp@qti.qualcomm.com
[2] School of Electrical and Computer Engineering, Cornell University, Ithaca, NY 14853, USA
[3] Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, USA
* Correspondence: haas@ece.cornell.edu; Tel.: +1-607-592-0467

† This paper is an extended version of our work published in the 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14), Niagara Falls, ON, Canada, 17–20 August 2014.

Academic Editors: Eduardo Fernández-Medina Patón and David G. Rosado
Received: 16 February 2016 ; Accepted: 28 April 2016 ; Published: 10 May 2016

**Abstract:** Cloud computing systems enable clients to rent and share computing resources of third party platforms, and have gained widespread use in recent years. Numerous varieties of mobile, small-scale devices such as smartphones, red e-health devices, *etc.*, across users, are connected to one another through the massive internetwork of vastly powerful servers on the cloud. While mobile devices store "private information" of users such as location, payment, health data, *etc.*, they may also contribute "semi-public information" (which may include crowdsourced data such as transit, traffic, nearby points of interests, *etc.*) for data analytics. In such a scenario, a mobile device may seek to obtain the result of a computation, which may depend on its private inputs, crowdsourced data from other mobile devices, and/or any "public inputs" from other servers on the Internet. We demonstrate a new method of delegating real-world computations of resource-constrained mobile clients using an encrypted program known as the garbled circuit. Using the garbled version of a mobile client's inputs, a server in the cloud executes the garbled circuit and returns the resulting garbled outputs. Our system assures privacy of the mobile client's input data and output of the computation, and also enables the client to verify that the evaluator actually performed the computation. We analyze the complexity of our system. We measure the time taken to construct the garbled circuit as well as evaluate it for varying number of servers. Using real-world data, we evaluate our system for a practical, privacy preserving search application that locates the nearest point of interest for the mobile client to demonstrate feasibility.

**Keywords:** secure cloud computing; privacy preserving search; garbled circuits; secure multiparty computation

## 1. Introduction

Cloud computing systems enable clients to rent and share computing resources of third party platforms such as Amazon Elastic Cloud, Microsoft Azure, *etc.*, and have gained increased attention in recent years. Availability of a large pool of hardware and software resources allows clients of cloud computing systems to perform computations on a vast amount of data without setting up their own infrastructure [1]. Numerous varieties of mobile, small-scale, IoT (Internet-of-Things)-devices (such as smartphones, e-health devices, *etc.*), across users, are connected to one another through the massive internetwork of vastly powerful servers on the cloud. These individual devices on their own are significantly enhancing our everyday lives; additionally, big data analytics on the cloud enables cross-correlation of numerous datasets to learn large-scale patterns across users to further enhance the user experience.

While mobile devices store "private information" of users such as location, payment, health data, *etc.*, they may also contribute "semi-public information" (which may include crowdsourced data such as transit, traffic, nearby points of interests, *etc.*) for data analytics. In such a scenario, a mobile device may seek to obtain the result of a computation, which may depend on its private inputs, crowdsourced data from other mobile devices, and/or any "public inputs" from other servers on the Internet. However, the mobile device may not perform the computation on its own: (i) due to resource constraints (in terms of computing, memory, network, power, *etc.*); and/or (ii) due to the restrictions (privacy, legal requirements, *etc.*) on the servers to share data from one user with others. Alternatively, exchange of data in " plain text form" between the mobile client and the cloud service provider to carry out the computations will also result in complete loss of data privacy.

Homomorphic encryption [2] is one of the approaches to address the problem of preserving data privacy. It can allow the cloud service providers to perform specific computations directly on the encrypted client data, without requiring private decryption keys. To perform any arbitrary computation on encrypted data, fully homomorphic encryption (FHE) schemes (e.g., [3–5]) have been proposed recently. However, FHE schemes are currently not suitable for mobile cloud computing or big data applications due to extremely large cipher text size and computational requirements beyond the capabilities of mobile devices today. Therefore, there exists a need for a more efficient alternative suitable for mobile systems.

We consider Yao's garbled circuits approach [6,7] in our work as a potential alternative to FHE schemes to drastically reduce the ciphertext size. In general, any computation can be represented using a Boolean circuit, for which, there exists a corresponding garbled circuit [6–9]. Each gate in a garbled circuit can be unlocked using a pair of input "wire keys" to reveal an output wire key, which, in turn, serves as an input wire key for unlocking the subsequent gate in the next level of the circuit, and so on. Each wire key corresponds to the underlying plaintext bit for the relevant wire in the circuit, and the association between the wire keys and the plaintext bits is kept secret from the cloud server that performs the computation. Thus, "oblivious evaluation" of any arbitrary function, expressible as a Boolean circuit, is possible using garbled circuits on any third-party server in the cloud.

While garbled circuits can preserve the privacy of client data, they are, however, one time programs—using the same version of the circuit more than once can reveal to an adversarial evaluator the mapping between the garbled values and the plaintext bits. Since creating a garbled circuit is at least as expensive as evaluating the underlying Boolean circuit, expecting the client to create a new version of the garbled circuit for each evaluation, however, is an unreasonable solution. Therefore, unlike FHE schemes which can directly delegate the desired computation to the cloud servers, a scheme using garbled circuits, presents the additional challenge of efficiently delegating to the cloud servers the creation of garbled circuit [10].

We propose a new method, in which to perform any computation on behalf of the client, a number of cloud servers collaborate to create a new version of the garbled circuit in a distributed manner. Using unique seed value from the client, each server generates a set of private input bits, and interacts with all the other servers to create a new garbled circuit, which is a function of the private input bits of all the servers. Essentially, the servers construct the desired garbled circuit without revealing their private inputs to one another using a secure multiparty computation protocol (e.g., Goldreich *et al.* [8,9]). Once a new version of the garbled circuit is created using multiple servers, the client utilizes an arbitrary server in the cloud for evaluation. Even if the evaluating server chooses to collude with any strict-subset of servers that participated in the creation of the garbled circuit, the resulting version of the garbled circuit, the garbled inputs that can unlock the circuit, and the corresponding garbled outputs, remain unrecognizable to the evaluator.

While the mobile clients are resource constrained, the cloud servers, however, are sufficiently provisioned to perform numerous intensive computation and communication tasks. Therefore, our proposed system is designed to readily exploit the real-world asymmetry that exists between typical mobile clients and cloud servers. Beyond the generation and exchange of compact cipher

text messages, our system requires very little computation and communication involvement from the mobile client to achieve secure and verifiable computing capability. However, the cloud servers can efficiently generate and exchange a large volume of random bits necessary for carrying out the delegated computation due to the significantly large amount of resources available to them. Therefore, our proposed scheme is very suitable for mobile environments (While our proposed system is especially beneficial for clients in a mobile environment, due to compact cipher text messages, it is also suitable for clients in other environments that need to delegate their computations to the cloud servers in a secure manner).

We employ the garbled circuit design of Beaver, Micali, Rogaway (BMR [11,12]) and the secure multiparty computation protocol of Goldreich *et al.* [8,9] for the purpose of building a secure cloud computing system. To facilitate the construction of the garbled circuit in the cloud, and also to enable the client to efficiently recover and verify the result of the computation, our method incorporates the novel use of the cryptographically secure Blum, Blum, Shub pseudo random number generator [13,14], whose strength relies on the computational difficulty of factorizing large numbers into primes. Using our proposed system, the client can efficiently verify that the evaluator actually and fully performed the requested computation.

Our major contributions are as follows. First, we design a secure mobile cloud computing system that uses multiple servers to perform any arbitrary computation on behalf of the client. Second, even if the evaluating server colludes with all but one of the servers that created the garbled circuit, our system assures the privacy of the client input and the result of the computation. Third, using our system the client can efficiently recover the result of the computation and verify whether the evaluator actually performed the computation. Fourth, we present an analysis of the complexity of our proposed scheme from the perspective of both the client and the server. Our scheme uses very small cipher text messages suitable for mobile clients in comparison to Gentry's FHE scheme [3]. Fifth, we evaluate our system for a privacy preserving search application that locates the nearest point of interest (bank/ATM) from the mobile client. Our method shows how publicly available location information from different sources (e.g., from Wells Fargo, Chase) may be used in conjunction with private location information of a mobile device in a privacy preserving computation. Finally, to demonstrate the feasibility of our system, we measure the time taken to construct and evaluate the garbled circuit for varying number of servers.

The rest of the paper is structured as follows. Section 2 presents a high-level overview, our adversary model and the main characteristics of our system. In Section 3, we present the background material. We present our proposed secure and verifiable cloud computing system in Section 4. We analyze the complexity of our system in Section 5. Section 6 presents our case study of a privacy preserving search application for computing the nearest point of interest from the mobile client. We discuss the related work in Section 7, and present our concluding remarks in Section 8.

## 2. A High-Level Overview of Our System

In our proposed system, the client utilizes a set of $(n+2)$ servers, $\{p_1, p_2, \ldots, p_n, p_c, p_e\}$, in the cloud. To each server $p_i$ $(1 \leq i \leq n)$, the client initially sends a description of the desired computation (simple examples of which could be addition of two numbers, computation of hamming distance between two bit sequences, *etc.*), and a unique seed value $s_i$ . Each of the $n$ servers first creates (or retrieves from its repository, if available already) a corresponding Boolean circuit $(B)$ for the requested computation. Each server $p_i$ $(1 \leq i \leq n)$, uses the unique seed value $s_i$ to generate a private pseudorandom bit sequence whose length is proportional to the total number of wires in the Boolean circuit $(B)$. Then, these $n$ servers use their private pseudorandom bit sequences and the Boolean circuit $(B)$ as inputs, communicate with one another, as well as perform some local computations, according to a secure multiparty computation protocol, to create their shares $(GC_i, (1 \leq i \leq n))$ for an one-time program called a "garbled circuit".

Once the shares for the garbled circuit are created, the client instructs all of the $n$ servers, $p_i$ $(1 \leq i \leq n)$, to send their shares, $GC_i$, to the server $p_c$. Applying an *XOR* operation on these shares, the server $p_c$ creates the desired circuit, $GC = GC_1 \oplus GC_2 \oplus \ldots \oplus GC_n$. Subsequently, the client instructs the server $p_c$ to send the garbled circuit $GC$ to another server $p_e$ for evaluation, either for immediate or future use. Now, the client generates its own garbled input values for each input wire in the circuit using the unique seed values $s_i$ $(1 \leq i \leq n)$, and sends them to the server $p_e$ for evaluation. Using these garbled inputs, the server $p_e$ unlocks each gate on the first level of the circuit to obtain the corresponding garbled outputs, which, in turn, can unlock each gate on the second level of the circuit, and so on. In this manner, the server $p_e$ unlocks every gate in the circuit, to obtain the garbled outputs of the circuit, and returns them to the client. To recover the result of the desired computation, the client now translates these garbled output values into plaintext bits.

Figure 1 depicts an overview of our proposed secure cloud computing system with $(n+2) = 5$ servers.

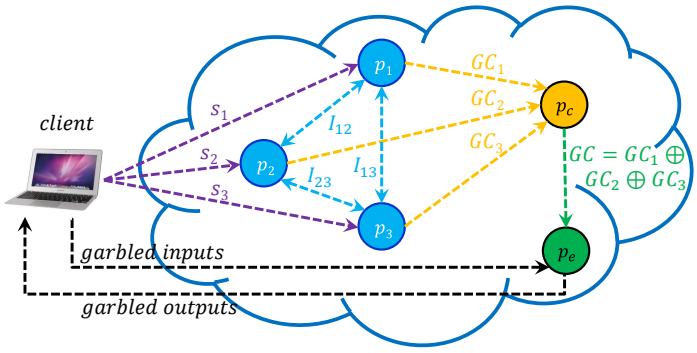


**Figure 1.** Our secure cloud computing model with $(n+2) = 5$ servers. (1) Client sends unique seed value, $s_i$, to each $p_i$ $(1 \leq i \leq 3)$; (2) $p_1$, $p_2$, $p_3$ interact ($I_{ij}, 1 \leq i < j \leq 3$) to construct shares of the garbled circuit $GC$; (3) Each $p_i$ sends its share ($GC_i$) to $p_c$; (4) $p_c$ computes $GC = GC_1 \oplus GC_2 \oplus GC_3$, and sends it to $p_e$; (5) Client generates garbled inputs, and sends them to $p_e$ for evaluation; (6) $p_e$ evaluates $GC$, and returns the garbled outputs to the client.

### *2.1. Our Adversarial Model*

We assume that there exists a secure communication channel between the client and each of the $(n+2)$ servers, $\{p_1, p_2, \ldots, p_n, p_c, p_e\}$, to send the unique seed values for pseudorandom bit generation, identity of the other servers, *etc.* We assume that every pair of communicating servers authenticate one another (Note that without authentication, an external adversary can masquerade any of the servers to disrupt the construction of the garbled circuit in an unpredictable manner. Additionally, if the communication between the client and the $(n+2)$ servers is compromised by an external adversary, then it can determine the semantics of garbled inputs/outputs based on the seed values. Therefore, both authenticity and confidentiality of communication are essential for the construction of garbled circuits). We assume a very capable adversary, where the evaluator $p_e$ will remain unable to determine the plain text values from the garbled values that it can observe during evaluation even if it individually colludes with any proper subset of the $n$ servers, $\{p_1, p_2, \ldots, p_n\}$. Our adversary model captures a very realistic scenario — where the client may be certain that some (however, not all) of the parties are corrupt, however, it is uncertain which of the parties are corrupt. If any adversarial party eavesdrops and analyzes all the messages between different parties, and also analyze all the messages that it has legitimately received from the other parties, it still cannot determine the shares of the other parties, or the plain text values of the garbled value pairs that are assigned to each wire in the circuit. Furthermore, the client can efficiently detect a cheating evaluator $p_e$ which returns arbitrary numbers as output to the client without performing any computation. A new garbled circuit is created for every evaluation in our model. Thus, the set of inputs and outputs that have changed or remained the same between different evaluations remains unknown to the evaluator.

### *2.2. Main Characteristics of Our System*

We highlight some of the main features of our secure cloud computing system in this subsection.

1. Offloaded Computation: The client delegates the intensive computational tasks to the cloud servers of creating and evaluating the garbled circuit. The client only chooses the cloud servers, provides them with unique seed values, generates garbled inputs during evaluation, and interprets garbled outputs returned by the evaluator.
2. Compact Cipher Text: While Gentry's scheme has an extremely large cipher text size, the cipher text size can be as small as a few hundred bits with our scheme (Section 5.6). Thus, our proposed method is far more practical for cloud computing in mobile systems in comparison to FHE schemes.
3. Decoupling: The process of creating the garbled circuit is decoupled from the process of evaluating the garbled circuit. While the servers, $p_i$ $(1 \leq i \leq n)$, interact with one another for creating the garbled circuit, the server $p_e$ evaluates the garbled circuit, independently.
4. Advance Construction: Since evaluation of the garbled circuit requires no interaction among the servers, if several versions of the garbled circuit for a given computation are precomputed and stored at the evaluator, in advance, then it can readily carry out the requested computation. Thus, the client will only incur the relatively short time taken to evaluate the garbled circuit. In other words, precomputation will drastically improve the response time for the client.
5. Collusion Resistance: To interpret any garbled value, the evaluator, $p_e$, must collude with each of the $n$ servers, $p_i$ $(1 \leq i \leq n)$. Thus, even if $(n-1)$ out of the $n$ servers are corrupt and collude with the evaluator, the privacy of the client's inputs and the result of the computation are still preserved.
6. Verification of Outputs: The client has the ability to verify that the evaluator actually carried out the requested computation.

## 3. Background

We briefly describe the construction and evaluation of Yao's garbled circuits [6,7], as well as the oblivious transfer protocols of Naor and Pinkas [15,16].

### *3.1. Yao's Garbled Circuit*

Any computation (simple examples of which could be addition of two numbers, computation of hamming distance between two bit sequences, *etc.*) can be represented as a Boolean circuit. Such a Boolean circuit can be expressed using only $\{AND, XOR\}$ gates, which corresponds to a "functionally-complete" set of operators. Therefore, any mechanism capable of performing $AND$ and $XOR$ operations, homomorphically, can enable secure delegation of any arbitrary computation. In our work, we achieve the homomorphic computation of $AND$ and $XOR$ gates using garbled circuits [6–9].

Each wire in the circuit is associated with a pair of keys known as "garbled values" that correspond to the underlying binary values. For example, the circuit in Figure 2 has seven wires, $\omega_i$ $(0 \leq i \leq 6)$, and three gates, $P, Q, R$, denoting $OR$, $AND$, $XOR$ gates respectively. Keys $\omega_i^0, \omega_i^1$ represent the garbled values corresponding to binary values 0, 1 respectively on the wire $\omega_i$.

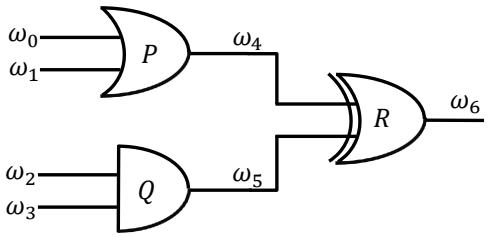


**Figure 2.** A circuit with three gates, $P, Q, R$, and seven wires, $\omega_i$ $(0 \leq i \leq 6)$.

Each gate in the circuit, is associated with a list of four values, in a random order, known as "garbled table". Table 1 shows the garbled tables for the gates $P, Q, R$ of Figure 2. Let $x, y \in \{0,1\}$. Let $E_k[v]$ denote the encryption of $v$ using $k$ as the key. Then, each entry in the garbled table for $P$ is of the form, $E_{\omega_0^x}[E_{\omega_1^y}[\omega_4^{x|y}]]$. Similarly, each entry in the garbled table for $Q$ and $R$ are of the forms, $E_{\omega_2^x}[E_{\omega_3^y}[\omega_5^{x.y}]]$ and $E_{\omega_4^x}[E_{\omega_5^y}[\omega_6^{x \oplus y}]]$, respectively.

**Table 1.** Garbled Tables for Gates $P, Q, R$.

| P | Q | R |
|---|---|---|
| $E_{\omega_0^0}[E_{\omega_1^1}[\omega_4^1]]$ | $E_{\omega_2^0}[E_{\omega_3^1}[\omega_5^0]]$ | $E_{\omega_4^1}[E_{\omega_5^1}[\omega_6^0]]$ |
| $E_{\omega_0^0}[E_{\omega_1^0}[\omega_4^0]]$ | $E_{\omega_2^1}[E_{\omega_3^0}[\omega_5^0]]$ | $E_{\omega_4^0}[E_{\omega_5^0}[\omega_6^0]]$ |
| $E_{\omega_0^1}[E_{\omega_1^1}[\omega_4^1]]$ | $E_{\omega_2^0}[E_{\omega_3^0}[\omega_5^0]]$ | $E_{\omega_4^1}[E_{\omega_5^0}[\omega_6^1]]$ |
| $E_{\omega_0^1}[E_{\omega_1^0}[\omega_4^1]]$ | $E_{\omega_2^1}[E_{\omega_3^1}[\omega_5^1]]$ | $E_{\omega_4^0}[E_{\omega_5^1}[\omega_6^1]]$ |

Suppose that the client wishes to delegate the computation of Figure 2, *i.e.*, $((a|b) \oplus (c.d))$, to a server in the cloud. The server is provided with a description of the circuit (Figure 2) along with the set of the garbled tables (Table 1), which together represents a "garbled circuit". However, the client keeps the mapping between the garbled values and the underlying binary values as secret. For example, to evaluate the circuit with inputs $a = 1,\ b = 0,\ c = 0,\ d = 1$, the client provides the set of garbled inputs, $\omega_0^1,\ \omega_1^0,\ \omega_2^0,\ \omega_3^1$, to the cloud server.

Now, assume that there exists a mechanism to determine whether a value is decrypted correctly; for example, through zero-padding. Using $\omega_0^1,\ \omega_1^0$ as keys, the server attempts to decrypt all the four entries in the garbled table for gate $P$; however, only the fourth entry will decrypt correctly to reveal the garbled output $\omega_4^1$. Similarly, using $\omega_2^0,\ \omega_3^1$ as keys, the first entry in the garbled table for gate $Q$ reveals the garbled output $\omega_5^0$. Finally, on using $\omega_4^1,\ \omega_5^0$ as keys, the third entry in the garbled table for gate $R$ reveals the garbled output $\omega_6^1$. Thus, the server can perform an "oblivious evaluation" of the garbled circuit and return the result of the computation $\omega_6^1$ to the client. Using the secret mapping, the client can determine that the garbled value $\omega_6^1$ corresponds to the binary value 1.

In our work, we use an alternative garbled circuit design from Beaver, Micali, Rogaway (BMR [11,12]), and adapt it, as we describe in Section 4, for the purpose of building a secure cloud computing system.

### *3.2. 1-Out-Of-2 Oblivious Transfer*

In the 1-out-of-2 oblivious transfer (OT) protocol, there are two parties: a sender and a chooser. There are two private messages at the sender: $M_0, M_1$, and the chooser holds a private choice bit, $\sigma \in \{0,1\}$. At the end of the 1-out-of-2 OT protocol, the sender learns nothing, and the chooser learns $M_\sigma$ only.

Let $p = 2q + 1$ denote a safe prime number; *i.e.*, $q$ is also a prime number. Let $Z_p^* = \{1, 2, 3, 4, \ldots, (p-1)\}$, which denotes the set of integers that are relatively prime to $p$. Let $G$ denote a subgroup of $Z_p^*$, where $|G| = q$. Let $g$ denote the generator for $G$.

The sender randomly chooses an element, $C \in G$, and sends it to the chooser. Note that the discrete logarithm of $C$ is unknown to the chooser. The chooser randomly selects an integer, $k$ $(1 \leq k \leq q)$, and sets $PK_\sigma = g^k \ mod\ p$, and $PK_{1-\sigma} = C \times (PK_\sigma)^{-1} \ mod\ p$. The chooser sends $PK_0$ to the sender. Note that $PK_0$ does not reveal the choice bit $\sigma$ to the sender.

The sender calculates $PK_1 = C \times (PK_0)^{-1} \ mod\ p$ on its own, and randomly chooses two elements, $r_0, r_1 \in G$. Let $h(x)$ denote the output of the hash function (e.g., SHA, which stands for secure hash algorithm) on input $x$. Let $E_i$ denote the encryption of $M_i$, $\forall i \in \{0,1\}$. Then, the sender calculates $E_i = [(g^{r_i} \ mod\ p), (h(PK_i^{r_i} \ mod\ p) \oplus M_i)]$, and sends both $E_0$, $E_1$ to the chooser.

The chooser decrypts $E_\sigma$ to obtain $M_\sigma$ as follows. Let $l_1 = g^{r_\sigma} \bmod p$ and $l_2 = h(PK_\sigma^{r_\sigma} \bmod p) \oplus M_\sigma$ denote the first and second numbers respectively in $E_\sigma$. The chooser calculates $M_\sigma$ using the relation, $M_\sigma = h(l_1^k \bmod p) \oplus l_2$. Note that since the discrete logarithm of $C$, and hence $PK_{1-\sigma}$, is unknown to the chooser, it cannot retrieve $M_{1-\sigma}$ from $E_{1-\sigma}$.

### *3.3. 1-Out-Of-4 Oblivious Transfer*

In the 1-out-of-4 oblivious transfer (OT) protocol, there are two parties: a sender and a chooser. There are four private messages at the sender: $M_{00}$, $M_{01}$, $M_{10}$, $M_{11}$, and the chooser holds two private choice bits, $\sigma_1$, $\sigma_2$. At the end of the 1-out-of-4 OT protocol, the sender learns nothing, and the chooser learns $M_{\sigma_1\sigma_2}$ only.

The sender randomly generates two pairs of keys, $(L_0, L_1)$, $(R_0, R_1)$, and computes the encryptions of $M_{00}$, $M_{01}$, $M_{10}$, $M_{11}$ as follows. Let $F_k(x)$ denote the output of a pseudorandom function such as AES-128, that is keyed using $k$ on the input $x$. Let $E_{ij}$ denote the encryption of $M_{ij}$, $\forall i, j \in \{0,1\}$. Then, $E_{ij} = M_{ij} \oplus F_{L_i}(2i + j + 1) \oplus F_{R_j}(2i + j + 1)$.

The sender and the chooser engage in 1-out-of-2 OT twice. In the first 1-out-of-2 OT, the sender holds two messages, $L_0$, $L_1$, and the chooser holds the choice bit, $\sigma_1$; at the end of this OT, the chooser obtains $L_{\sigma_1}$. In the second 1-out-of-2 OT, the sender holds two messages, $R_0$, $R_1$, and the chooser holds the choice bit, $\sigma_2$; at the end of this OT, the chooser obtains $R_{\sigma_2}$.

Now, the sender sends all the four encryptions, $E_{00}$, $E_{01}$, $E_{10}$, $E_{11}$, to the chooser. Using $L_{\sigma_1}$, $R_{\sigma_2}$, the chooser decrypts $E_{\sigma_1\sigma_2}$ to obtain $M_{\sigma_1\sigma_2}$, as $M_{\sigma_1\sigma_2} = E_{\sigma_1\sigma_2} \oplus F_{L_{\sigma_1}}(2\sigma_1 + \sigma_2 + 1) \oplus F_{R_{\sigma_2}}(2\sigma_1 + \sigma_2 + 1)$.

## 4. Secure and Verifiable Cloud Computing for Mobile Systems

In Section 4.1, we present the construction of BMR garbled circuit [11,12] using $n$ servers through the secure multiparty computation protocol of Goldreich *et al.* [8,9]. In Section 4.2, we highlight how we adapt the protocol of Goldreich and the garbled circuit design of BMR, in order to suit them for our secure cloud computing model. In our model, each server $p_i$ $(1 \leq i \leq n)$, generates shares of garbled values using cryptographically secure pseudorandom number generation method of Blum, Blum, Shub [13,14]. In Section 4.3, we present our method of how the client efficiently recovers the result of the delegated computation, as well as how the client verifies that the evaluator in fact carried out the computation. We summarize our secure cloud computing model in Section 4.4.

### *4.1. Construction and Evaluation of Garbled Circuits*

#### 4.1.1. Construction of the Garbled Circuit, *GC*

Garbled Value Pairs: In order to denote the underlying plaintext bits 0 and 1, each wire in the circuit is associated with a pair of garbled values. Let $x, y$ denote two input wires, and $z$ denote the output wire of a specific gate $A$ in the circuit. Assume that $(\alpha_0, \alpha_1)$, $(\beta_0, \beta_1)$ and $(\gamma_0, \gamma_1)$ are the pair of garbled values associated with the wires $x$, $y$, and $z$, respectively. Let $LSB(v)$ denote the least significant bit of the number $v$. Then, $LSB(\alpha_0) = LSB(\beta_0) = LSB(\gamma_0) = 0$ and $LSB(\alpha_1) = LSB(\beta_1) = LSB(\gamma_1) = 1$.

Let $n$ denote the number of servers and $k$ denote the security parameter. Then, each garbled value is $(nk + 1)$ bits long, and is a concatenation of shares from the $n$ servers. Let $a, b, c \in \{0,1\}$. Then the garbled values for gate $A$ are as follows: $\alpha_a = \alpha_{a1}||\alpha_{a2}||\alpha_{a3}||\ldots||\alpha_{an}||a$; $\beta_b = \beta_{b1}||\beta_{b2}||\beta_{b3}||\ldots||\beta_{bn}||b$; $\gamma_c = \gamma_{c1}||\gamma_{c2}||\gamma_{c3}||\ldots||\gamma_{cn}||c$; here $\alpha_{ai}$, $\beta_{bi}$, $\gamma_{ci}$ are shares of server $p_i$ $(1 \leq i \leq n)$.

$\lambda$ Value: A 1-bit $\lambda$ value is associated with each wire in the circuit, which determines the semantics for the pair of garbled values associated with that wire. Specifically, for a garbled value $v$ whose $LSB(v) = b$, the underlying plaintext bit is $(b \oplus \lambda)$.

Collusion Resistance: Let $\lambda_x, \lambda_y, \lambda_z$ denote the $\lambda$ values for the three wires $x, y, z$ respectively. Then, $\lambda_x = \bigoplus_{i=1}^{n} \lambda_{xi}$, $\lambda_y = \bigoplus_{i=1}^{n} \lambda_{yi}$, $\lambda_z = \bigoplus_{i=1}^{n} \lambda_{zi}$, where $\lambda_{xi}, \lambda_{yi}, \lambda_{zi}$ are shares of server $p_i$ $(1 \leq i \leq n)$.

Since the $\lambda$ value of each wire is unknown to any individual server, the evaluator of the garbled circuit must collude with each of the $n$ servers to interpret the garbled values.

Garbled Table: Each gate, $A$, in the circuit, is associated with an ordered list of four values, $[A_{00}, A_{01}, A_{10}, A_{11}]$, representing the garbled table for gate $A$. Note that the $\lambda$ values of the input wires, which are random, and also unknown to any of the $n$ servers, determine the order of the entries in the garbled table. Let $\otimes$ denote the binary operation of gate $A$, where $\otimes \in \{XOR, AND\}$. Then, the value of one specific entry is expressed as, $A_{ab} = \gamma_{[((\lambda_x \oplus a) \otimes (\lambda_y \oplus b)) \oplus \lambda_z]} \oplus [G_b(\alpha_{a1}) \oplus G_b(\alpha_{a2}) \oplus \ldots \oplus G_b(\alpha_{an})] \oplus [G_a(\beta_{b1}) \oplus G_a(\beta_{b2}) \oplus \ldots \oplus G_a(\beta_{bn})]$, where $G_a$ and $G_b$ are pseudorandom functions that expand $k$ bits into $(nk+1)$ bits. In the expression for $A_{ab}$ , $G_0(s)$ and $G_1(s)$ denote the first and last $(nk+1)$ bits of $G(s)$ respectively, where $G$ denotes a pseudorandom generator, which on providing a $k$-bit input seed, outputs a sequence of $(2nk+2)$ bits, *i.e.*, if $|s| = k$, then $|G(s)| = (2nk+2)$. $G$ may represent the output of AES block cipher in output feedback mode, for example.

For each gate in the circuit, to compute each garbled table entry $A_{ab}$, the $n$ servers use the secure multiparty computation protocol of Goldreich [8,9] (Section 4.1.2), where $f(x_1, x_2, \ldots, x_n) = A_{ab}$, and for each server, $p_i, (1 \leq i \leq n)$, its private input, $x_i = [\lambda_{xi}, \lambda_{yi}, \lambda_{zi}, G_b(\alpha_{ai}), G_a(\beta_{bi}), \gamma_{0i}, \gamma_{1i}]$ is a vector of length $m = (3 + 2(nk+1) + 2k)$ bits.

#### 4.1.2. Secure Multiparty Computation of an Entry, $A_{ab}$

Assume that $n$ parties need to compute the value of an arbitrary function of their private inputs, namely $f(x_1, x_2, \ldots, x_n)$ without revealing their private inputs to one another. Assume that the function $f(x_1, x_2, \ldots, x_n)$ is expressed as a Boolean circuit ($B'$) (Boolean circuit $B'$ is different from Boolean circuit $B$. While $B$ is a circuit that corresponds to the computation requested by the client (e.g., addition of two numbers), $B'$ is a circuit that creates the entries such as $A_{ab}$ in the garbled tables of the garbled circuit $GC$) using *XOR* and *AND* gates only.

We briefly describe the secure multiparty computation protocol of Goldreich [8,9] as follows. For each wire in the Boolean circuit, the actual binary value which corresponds to the *XOR*-sum of shares is distributed among the $n$ parties.

Evaluation of each *XOR* gate in the circuit, requires no communication—it is carried out locally; each party merely performs an *XOR* operation over its shares for the two input wires to obtain its share for the output wire.

Evaluation of each *AND* gate in the circuit, however, requires communication between all pairs of parties. For the two inputs wires of the *AND* gate, let $a_i$, $b_i$ denote the shares of party $p_i$; and let $a_j$, $b_j$ denote the shares of party $p_j$. Then, the output of the *AND* gate is expressed as follows:

$(\bigoplus_{i=1}^{n} a_i).(\bigoplus_{i=1}^{n} b_i) = [\bigoplus_{1 \leq i < j \leq n}((a_i \oplus a_j).(b_i \oplus b_j))] \bigoplus_{i=1}^{n}((a_i.b_i).I)$, where $I = n \bmod 2$.

Each party $p_i$ computes $((a_i.b_i).I)$ locally; and the computation of each *partial-product*, $((a_i \oplus a_j).(b_i \oplus b_j))$, is realized using 1-out-of-4 oblivious transfer (OT [15,16]) between $p_i$ and $p_j$, such that no party reveals its shares to the other party [8,9].

Following the above procedure, the $n$ parties evaluate each gate in the circuit. Thus, in the end, for the BMR protocol, as we have described above, each server $p_i$ $(1 \leq i \leq n)$, obtains the share, $(f(x_1, x_2, \ldots, x_n))_i = (A_{ab})_i$, such that $f(x_1, x_2, \ldots, x_n) = A_{ab} = \bigoplus_{i=1}^{n}(A_{ab})_i$.

#### 4.1.3. Evaluation of the Garbled Circuit, $GC$

The garbled table for each gate, $A$, in the circuit is an ordered list of four values, $[A_{00}, A_{01}, A_{10}, A_{11}]$.

During evaluation, let $\alpha$, $\beta$ denote the garbled values for the two input wires of gate $A$. Let $a = LSB(\alpha)$, $b = LSB(\beta)$. Then, the garbled value for the output wire, $\gamma$, is computed using $\alpha$, $\beta$, $A_{ab}$, as described in the two steps below:

1. split the most significant $nk$ bits of $\alpha$ into $n$ parts, $\alpha_1$, $\alpha_2$, $\alpha_3$, $\ldots$, $\alpha_n$; each part has $k$ bits; similarly, split the most significant $nk$ bits of $\beta$ into $n$ parts, $\beta_1$, $\beta_2$, $\beta_3$, $\ldots$, $\beta_n$; each part has $k$ bits; *i.e.*, $|\alpha_i| = |\beta_i| = k$, where $1 \leq i \leq n$.

2. compute $\gamma = [G_b(\alpha_1) \oplus G_b(\alpha_2) \oplus \ldots \oplus G_b(\alpha_n)] \oplus [G_a(\beta_1) \oplus G_a(\beta_2) \oplus \ldots \oplus G_a(\beta_n)] \oplus A_{ab}$.

In this manner, the garbled output for any gate in the circuit can be computed using the garbled table and the two garbled inputs to the gate. Note that while the construction of the garbled circuit requires interaction among all the $n$ parties, $p_i$ $(1 \leq i \leq n)$, the server $p_e$ can perform the evaluation of the garbled circuit independently.

### *4.2. Secure and Verifiable Cloud Computing through Secure Multiparty Computation*

In a secure multiparty computation protocol, multiple parties hold private inputs, and receive the result of the computation. However, in our proposed secure cloud computing system, while multiple parties participate in the creation of garbled circuits, only the client holds private inputs and obtains the result of the computation in garbled form. Therefore, we adapt the protocols of Goldreich and BMR in a number of ways, as we discuss in this section, to build an efficient, secure cloud computing system, that also enables the client to easily verify the outputs of the computation.

First, note that in the protocol of Goldreich [8,9], each party $p_i$ sends its share $(f(x_1, x_2, \ldots, x_n))_i$ to all the other parties. Using these shares, each party computes $f(x_1, x_2, \ldots, x_n)$ as $\bigoplus_{i=1}^{n}(f(x_1, x_2, \ldots, x_n))_i$. In our secure cloud computing system, however, we require each server $p_i$ $(1 \leq i \leq n)$, to send its share $(f(x_1, x_2, \ldots, x_n))_i = (A_{ab})_i$ to only one server, $p_c$, which combines the received shares using the *XOR* operation to produce entries such as $A_{ab}$ for each garbled table in the garbled circuit, *GC*.

Second, in the BMR protocol [11,12], which is also a secure multiparty computation protocol, in addition to creating the garbled circuit, for evaluation, the $n$ parties also create garbled inputs using secure multiparty computation. Then, each of these $n$ parties evaluates the garbled circuit and obtains the result of the computation. In our system model, since only the client holds the inputs for the computation, it generates the corresponding garbled input for each input wire on its own using the seed values it sends to each server, $p_i$ $(1 \leq i \leq n)$. Then, it sends these garbled values to the server $p_e$ for evaluating the garbled circuit and obtains the result in garbled form. Note that in our model, only the server $p_e$ evaluates the garbled circuit, and that $p_e$ cannot interpret any garbled value, unless it colludes with all the $n$ servers, $p_i$ $(1 \leq i \leq n)$.

Third, in the BMR protocol [11,12], the $\lambda$ value is set to zero for each output wire in the Boolean circuit. Therefore, each party evaluating the garbled circuit obtains the result of the computation in plaintext form from the LSB of the garbled output for each output wire in the circuit. In our system model, however, the $\lambda$ value for each output wire is also determined using the *XOR*-sum of the shares from all the $n$ servers, $p_i$ $(1 \leq i \leq n)$. As a consequence, result of the computation in plaintext form remains as secret for the evaluator $p_e$.

Fourth, in the protocol of Goldreich [8,9], each party splits and shares each of its private input bits with all the other parties over pairwise secure communication channels. In our system, we eliminate this communication using a unique seed value $s_{ik}$ that the client shares with all pairs of parties, $(p_i, p_k)$, $(1 \leq i, k \leq n)$. To split and share each of its $m$ private input bits $x_{ij}$ $(1 \leq j \leq m)$, party $p_i$ uses the seed value $s_{ik}$ to generate $r_{kj}$ $(\forall k \neq i)$. Specifically, party $p_i$ sets its own share as $x_{ij} \bigoplus_{k=1,k\neq i}^{n} r_{kj}$, where $r_{kj} = R(s_{ik}, j, gate_{id}, entry_{id})$ corresponds to the output of the pseudorandom bit generator using the seed value $s_{ik}$ for the $j^{th}$ private input bit of party $p_i$, for a specific garbled table entry ($entry_{id}$) of one of the gates ($gate_{id}$) in the Boolean circuit. Similarly, party $p_k$ sets its own share as $r_{kj} = R(s_{ik}, j, gate_{id}, entry_{id})$. The total number of pseudorandom bits generated by each party for the protocol of Goldreich equals $2(n-1)m \times 4N_g = 8(n-1)m \times N_g$, where $m = (3 + 2(nk+1) + 2k)$, and $N_g$ denotes the total number of gates in the circuit. In other words, our approach eliminates the exchange of a very large number of bits ($O(n^3 k N_g)$ bits) between the $n$ parties.

Fifth, our novel use of the Blum, Blum, Shub pseudorandom number generator for generating garbled value shares enables the client to efficiently recover and verify the outputs of the computation. The client can detect a cheating evaluator, if it returns arbitrary values as output. We present this in a greater detail in Section 4.3.

### *4.3. Recovery and Verification of Outputs*

We address the following questions in this subsection:

1. How does the client efficiently recover the result of the computation without carrying out the delegated computations by itself?
2. How does the client verify that the evaluator, in fact, evaluated the garbled circuit? Stated differently, is it possible for the client to determine whether the evaluator returned arbitrary numbers without carrying out any computation at all, instead of the actual garbled output for each output wire?

We can enable the client to efficiently retrieve and verify the outputs received from the evaluator, $p_e$. To accomplish this, each of the $n$ parties that participates in the creation of the garbled circuit uses the cryptographically secure Blum, Blum, Shub pseudorandom number generator [13,14], as we have described below.

Let $N = p \times q$ denote the product of two large prime numbers such that $p$ and $q$ are congruent to 3 *mod* 4. The client chooses a set of $n$ seed values, $\{s_1, s_2, \ldots, s_n\}$; each seed value $s_i$ belongs to $Z_N^*$, the set of integers relatively prime to $N$. Over a secure communication channel, the client sends the modulus value $N$ and a unique seed value $s_i$ to each party $p_i$ $(1 \leq i \leq n)$. However, the client retains the prime factors, $p$, $q$, of $N$ as secret.

Let $b_{i,j} = LSB(x_{i,j})$ denote the $j^{th}$ bit generated by the party $p_i$; here $x_{i,j} = x_{i,(j-1)}^2 \ mod \ N$, and $x_{i,0} = s_i$.

In the Boolean circuit, each wire $\omega$ is associated with $(\omega_0, \omega_1)$, a pair of garbled values, and $\lambda_\omega$, a 1-bit value. These are expressed as follows:

$\omega_0 = \omega_{01}||\omega_{02}||\omega_{03}||\ldots||\omega_{0n}||0$,
$\omega_1 = \omega_{11}||\omega_{12}||\omega_{13}||\ldots||\omega_{1n}||1$, and
$\lambda_\omega = \lambda_{\omega 1} \oplus \lambda_{\omega 2} \oplus \lambda_{\omega 3} \oplus \ldots \oplus \lambda_{\omega n}$.

In the above expressions: $\omega_{0i}$, $\omega_{1i}$ and $\lambda_{\omega i}$ are shares of the party $p_i$ $(1 \leq i \leq n)$, and $|\omega_{0i}| = |\omega_{1i}| = k$, and $|\lambda_{\omega i}| = 1$.

Let $W$ denote the number of wires in the Boolean circuit. For each wire $\omega$ $(0 \leq \omega \leq W-1)$, in the Boolean circuit, each party, $p_i$ $(1 \leq i \leq n)$, generates $(2k+1)$ pseudo random bits. Therefore, each party, $p_i$, generates a total of $(W(2k+1))$ pseudorandom bits.

To generate its shares $\omega_{0i}, \omega_{1i}$, and $\lambda_{\omega i}$ for wire $\omega$, party $p_i$ concatenates the $b_{i,j}$ values, where the indices $j$ belong to the range: $[(\omega(2k+1)+1), (\omega+1)(2k+1)]$. Let $\Omega_{\omega k} = \omega(2k+1)$ for concise notation. Then,

$\omega_{0i} = b_{i,(\Omega_{\omega k}+1)}||b_{i,(\Omega_{\omega k}+2)}||\ldots||b_{i,(\Omega_{\omega k}+k)}$,
$\omega_{1i} = b_{i,(\Omega_{\omega k}+k+1)}||b_{i,(\Omega_{\omega k}+k+2)}||\ldots||b_{i,(\Omega_{\omega k}+2k)}$,
$\lambda_{\omega i} = b_{i,(\Omega_{\omega k}+2k+1)}$.

Short-Cut: Note that each server $p_i$ must compute all the previous $(j-1)$ bits before it can compute the $j^{th}$ bit. However, using its secret knowledge of the prime factors of $N$, *i.e.*, $p$, $q$, the client can directly calculate any $x_{i,j}$ (hence, the bit $b_{i,j}$) using the relation: $x_{i,j} = x_{i,0}^{2^j \ mod \ C(N)} \ mod \ N$; here $C(N)$ denotes the "Carmichael function", which equals the least common multiple of $(p-1)$ and $(q-1)$.

Therefore, without having to compute $\omega_0$, $\omega_1$, and $\lambda_\omega$ for any intermediate wire in the circuit, the client can readily compute $\omega_0$, $\omega_1$, and $\lambda_\omega$ for any output wire $\omega$ in the circuit using the secret values $p$, $q$. The client can translate each of the garbled values from the evaluator $p_e$ into a plaintext bit using the $\lambda_\omega$ values of the output wires in order to recover the result of the requested computation. Only if the garbled output from the evaluator matches with either $\omega_0$ or $\omega_1$, for each output wire $\omega$ of the circuit, the client declares that the output verification is successful.

Collusion Resistance: Notice that the evaluator can return one of the two expected garbled outputs for every output wire in the circuit without performing any computation at all, if and only if: (i) it colludes with all the $n$ servers, $\{p_1, p_2, \ldots, p_n\}$, that participated in creating the garbled circuit; or (ii) it factorizes $N$ into the prime factors, $p$ and $q$, which is infeasible.

Furthermore, note that in order to cheat successfully without any collusion, the evaluator would have to correctly guess one of the two expected garbled outputs for each output wire in the circuit. Each garbled output value is $(nk+1)$ bits long. Therefore, for any reasonable value of the number of servers $(n \geq 2)$ and security parameter $(k = 128$, for example$)$, the evaluator would have to correctly guess hundreds of bits for each output wire in the circuit. Hence, it is very highly unlikely for the evaluator to be successful.

Unpredictability: Even with the knowledge of all the previous/future bits, one cannot predict the next/previous bit output from the Blum, Blum, Shub pseudorandom number generator [13,14]. Therefore, through the observation of garbled values during evaluation, the evaluating server, $p_e$, cannot predict the preceding or subsequent garbled values, or the $\lambda$ values for every wire in the circuit.

### *4.4. Summary of Our Proposed System*

We summarize our secure cloud computing model in this subsection.

1. The client chooses $\{p_1, p_2, \ldots, p_n, p_c, p_e\}$, a set of $(n+2)$ servers in the cloud. Then, to each server $p_i$ $(1 \leq i \leq n)$, the client provides a unique seed value $s_i$, and a description of the desired computation. To each pair of servers, $(p_i, p_k), (1 \leq i, k \leq n)$, the client also provides another seed value $s_{ik}$.
2. For the requested computation, each server, $p_i$ $(1 \leq i \leq n)$, creates (or retrieves from its repository, if available already) a corresponding Boolean circuit $(B)$.
3. Using the pseudo random generator of Blum, Blum, Shub, each server, $p_i$ $(1 \leq i \leq n)$, uses the seed value $s_i$ to generate its shares for the pair of garbled values and a $\lambda$ value for each wire in the circuit $(B)$. Each server, $p_i$, generates a pseudorandom bit sequence of length $W(2k+1)$ bits using seed value $s_i$; here $W$ denotes the number of wires in the Boolean circuit $(B)$.
4. The client instructs the $n$ servers $p_i$ $(1 \leq i \leq n)$ to construct the shares $(GC_i)$ of a BMR garbled circuit, $GC$, using their shares as private inputs for the secure multiparty computation protocol of Goldreich. Each pair of servers, $(p_i, p_k), (1 \leq i, k \leq n)$, generates pseudorandom bits using pairwise seed values $s_{ik}$ while using the protocol of Goldreich. Let $A_i = (A_{00})_i || (A_{01})_i || (A_{10})_i || (A_{11})_i$ denote the shares of server $p_i$ corresponding to the four garbled table entries of gate $A$. Then, $GC_i$ equals the concatenation of all bit strings of the form $A_i$, where the concatenation operation is carried out over all the gates in the circuit.
5. The client instructs each of the $n$ servers, $p_i$ $(1 \leq i \leq n)$ to send their shares $GC_i$ to the server $p_c$. Performing only $XOR$ operations, the server $p_c$ creates the desired circuit, $GC = GC_1 \oplus GC_2 \oplus \ldots \oplus GC_n$, which is sent to server $p_e$ for evaluation.
6. The client generates garbled input values for each input wire in the circuit using the unique seed values $s_i$ $(1 \leq i \leq n)$, and sends them to the server $p_e$ for evaluation. The client also generates the $\lambda$ values and the two possible garbled values for each output wire in the circuit using these seed values, and keeps the $\lambda$ values as secret.
7. The server $p_e$ evaluates $GC$ using the garbled inputs to obtain the garbled output for every output wire in the circuit and sends them to the client. The client now translates these garbled values into plaintext bits, using the $\lambda$ values, to recover the result of the requested computation.
8. For each output wire in the circuit, the client checks whether the garbled output that is received from the evaluator, $p_e$, matches with one of the two expected garbled values that it computed on its own. If the client determines that there is a match for all output wires, it declares that the evaluator in fact carried out the computation.

## 5. Complexity

### *5.1. Circuit Size of One Garbled Table Entry*

We analyze the size of the Boolean circuit $(B')$ for computing one specific entry $(A_{ab})$ in the garbled table. Assume that each gate produces one output bit from two input bits. As we have shown in Section 4.1.1,

$A_{ab} = \gamma_{[((\lambda_x \oplus a) \otimes (\lambda_y \oplus b)) \oplus \lambda_z]} \oplus [G_b(\alpha_{a1}) \oplus G_b(\alpha_{a2}) \oplus \ldots \oplus G_b(\alpha_{an})] \oplus [G_a(\beta_{b1}) \oplus G_a(\beta_{b2}) \oplus \ldots \oplus G_a(\beta_{bn})]$; here $\otimes \in \{XOR, AND\}$ denotes the binary operation of gate $A$.

Let $s$ denote the expression: $((\lambda_x \oplus a) \otimes (\lambda_y \oplus b)) \oplus \lambda_z$. Since $\lambda_x = \bigoplus_{i=1}^{n} \lambda_{xi}$, $\lambda_y = \bigoplus_{i=1}^{n} \lambda_{yi}$, $\lambda_z = \bigoplus_{i=1}^{n} \lambda_{zi}$, where $\lambda_{xi}, \lambda_{yi}, \lambda_{zi}$ are shares of server $p_i, (1 \leq i \leq n)$, computing $s$ requires a total of $3 + 3(n-1) = 3n$ *XOR* gates and 1 $\otimes$ gate.

Boolean circuit $B'$ includes a multiplexer that chooses $\gamma_s$, where $\gamma_s = ((\gamma_0 \oplus \gamma_1).s) \oplus \gamma_0$, which is composed of 2 *XOR* gates and 1 *AND* gate. The multiplexing is performed on the most significant $nk$ bits since $|\gamma_0| = |\gamma_1| = nk + 1$, and $LSB(\gamma_s) = s$. This multiplexer is composed of a total of $2nk$ *XOR* gates and $nk$ *AND* gates.

Now, consider the expression: $\gamma_s \oplus [G_b(\alpha_{a1}) \oplus G_b(\alpha_{a2}) \oplus \ldots \oplus G_b(\alpha_{an})] \oplus [G_a(\beta_{b1}) \oplus G_a(\beta_{b2}) \oplus \ldots \oplus G_a(\beta_{bn})]$; it has $(2n + 1)$ terms, which are combined using $2n$ *XOR* operations. Computing this expression requires a total of $2n(nk + 1)$ *XOR* gates, since each term has a length of $(nk + 1)$ bits.

Therefore, the Boolean circuit ($B'$) that computes one specific garbled table entry ($A_{ab}$) has a total of $(3n + 2nk + 2n(nk + 1))$ *XOR* gates, $nk$ *AND* gates, and 1 $\otimes$ gate.

Figure 3 shows the total number of gates in the circuit that computes $A_{ab}$, when $A$ is an *AND* gate, as a function of $n$ for a fixed value of $k = 128$ bits. Notice the relatively small number of *AND* gates in the circuit. For example, when $n = 6$, the circuit that computes $A_{ab}$ has a total of 10782 *XOR* and 769 *AND* gates. To summarize, while the number of *XOR* gates increases quadratically with $n$, the number of *AND* gates increases only linearly with $n$.

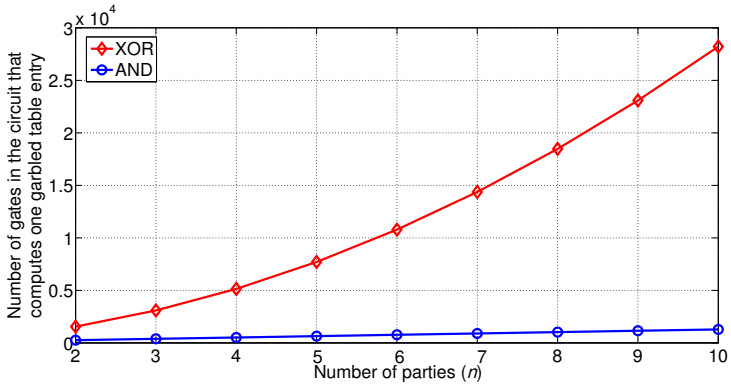


**Figure 3.** Number of gates in the circuit that computes one garbled table entry as a function of $n$.

Let $B$ denote the Boolean circuit that corresponds to the desired computation such as addition of two numbers. Then, in order to create the garbled circuit $GC$ for $B$, the $n$ parties use the Boolean circuit $B'$ for the protocol of Goldreich to compute each one of the four garbled table entries of the form $A_{ab}$ for each gate $A$ in the circuit $B$.

### *5.2. Communication Cost to Compute One Garbled Table Entry*

In general, a 1-out-of-2 OT [15] exchange between two parties involves the exchange of: (i) a random element $C$ from the prime order subgroup, $G$, of $Z_p^*$; (ii) a public key, $PK_0$; and (iii) the encryptions, $E_0$, $E_1$, of the plaintext messages $M_0$, $M_1$. Let $k$ denote the security parameter, which equals the size of the plaintext messages, $M_0$, $M_1$. Let $s_{1:2}$ denote the total number of bits that are exchanged during a 1-out-of-2 OT. Then, $s_{1:2} = |C| + |PK_0| + |E_0| + |E_1| = |p| + |p| + (|p| + k) + (|p| + k) = 4|p| + 2k$.

A 1-out-of-4 OT [16] exchange between two parties includes: (i) two 1-out-of-2 OTs, and (ii) four encryptions, $E_{00}$, $E_{01}$, $E_{10}$, $E_{11}$. Let $s_{1:4}$ denote the total number of bits exchanged during a 1-out-of-4 OT. Then, $s_{1:4} = 2(s_{1:2}) + 4k = 8(|p| + k)$; here $|p|$ and $k$ are public and symmetric key security parameters, respectively. For example, $|p| = 3072$ achieves the equivalent of $k = 128$-bit security [17]; in this case, the sum of the sizes of all messages exchanged during a 1-out-of-4 OT is $s_{1:4} = 3200$ bytes.

For each *AND* gate in the circuit $B'$, each pair of servers, $(p_i, p_j), 1 \leq i < j \leq n$, engage in a 1-out-of-4 OT, and there are a total of $n(n-1)/2$ combinations of $(p_i, p_j)$. The total number of 1-out-of-4 OTs is at most $t_{1:4} = (nk+1) \times n(n-1)/2$ since the number of *AND* gates in the circuit $B'$ is at most $(nk+1)$.

To create the desired garbled table entry, $A_{ab}$, at the completion of the secure multiparty computation protocol of Goldreich, each server, $p_i$ $(1 \leq i \leq n)$, sends its share $(A_{ab})_i$ to another server, $p_c$, which receives a total of $s^\star = n(nk+1)$ bits from the other $n$ servers since $|(A_{ab})_i| = nk+1$.

To summarize, in order to create one garbled table entry, $A_{ab}$, the total amount of network traffic, $T = (t_{1:4} \times s_{1:4}) + s^\star = (nk+1)[(4(|p|+k) \times n(n-1)) + n]$. The network traffic is a cubic function of $n$ when the security parameters, $k$ and $|p|$, are fixed.

Figure 4 shows the network traffic to create one garbled table entry as a function of $n$. For example, when $n = 5$, the cloud servers exchange 19.56 MB of data in order to create a single entry in the garbled table.

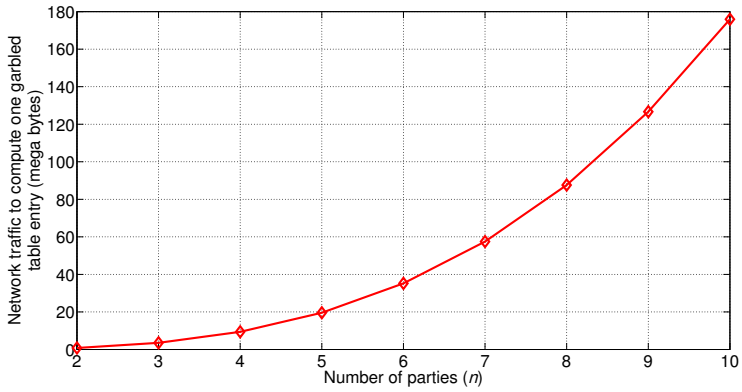


**Figure 4.** Amount of network traffic to compute one garbled table entry as a function of $n$.

In the process of creating the garbled circuit, $GC$, the total amount of network traffic equals $4N_g \times T$, where $N_g$ denotes the total number of gates in the circuit $B$ that corresponds to the desired computation.

### *5.3. Computation Cost of Creating the Garbled Circuit*

Let $W$ denote the total number of wires in the circuit $B$. For each wire, each server, $p_i$ $(1 \leq i \leq n)$, generates $(2k+1)$ bits using the Blum, Blum, Shub (BBS) pseudorandom number generator (PRNG) [13,14] for its share of garbled values and the $\lambda$ value. Therefore, the $n$ servers collectively generate a total of $b_1 = n(2k+1)W$ bits using the BBS PRNG. Let $N$ denote the modulus value in BBS PRNG (note: $|N| = 3072$ achieves 128-bit security [17]). Then, $n(2k+1)W$ modular multiplication operations in $Z_N^*$ are necessary to generate bits using BBS PRNG.

Let $W_o$ denote the number of output wires in the circuit $B$. Let $G$ denote the PRNG, which we have described in Section 4.1.3, that outputs a sequence of $(2nk+2)$ bits on providing a $k$-bit input seed. Each server uses the PRNG $G$, on its share of each garbled value for every non-output wire in the circuit $B$. Then, while creating the garbled circuit, the $n$ servers collectively use the PRNG $G$, $2n(W - W_o)$ times to generate a total of $b_2 = 4n(nk+1)(W - W_o)$ bits.

Let $N_g$ denote the total number of gates in the circuit $B$. For protocol of Goldreich, the total number of pseudorandom bits generated by each party using the PRNG $R$, equals $8(n-1)m \times N_g$, where $m = (3 + 2(nk+1) + 2k)$ (Section 4.2). Thus, the $n$ parties collectively generate a total of $b_3 = 8n(n-1)(3 + 2(nk+1) + 2k) \times N_g$ bits using the PRNG $R$.

Note that both the PRNG $G$ and PRNG $R$ can be realized using a block cipher such as AES (which stands for advanced encryption standard) operating in output feedback mode.

Let $t_{1:4}$ denote the number of 1-out-of-4 OTs to create one garbled table entry (Section 5.2). Then, the total number of 1-out-of-4 OTs to create the complete garbled circuit $GC$ is at most $4N_g \times t_{1:4} = 4N_g \times (nk+1) \times n(n-1)/2$.

During a 1-out-of-2 oblivious transfer, the sender and chooser generate a total of $|C| + |k| + |r_0| + |r_1| = (4|p| - 1)$ bits. Each 1-out-of-4 oblivious transfer involves the cost of two 1-out-of-2 oblivious transfers, in addition to generating a total of $|L_0| + |L_1| + |R_0| + |R_1| = 4k$ bits. A very small constant number of modular arithmetic operations, AES and SHA crypto operations are carried out during each OT. While creating the garbled circuit $GC$, these sets of operations are performed $4N_g \times (nk+1) \times n(n-1)/2$ times; and a total of $b_4 = 4N_g \times (nk+1) \times (n(n-1)/2) \times (8|p| + 4k - 2)$ bits are generated during the OTs.

To summarize, the total number of bits that are generated by the $n$ parties while creating the garbled circuit is $b = b_1 + b_2 + b_3 + b_4 = (n(2k+1)W) + (4n(nk+1)(W - W_o)) + (8n(n-1)(3 + 2(nk+1) + 2k) \times N_g) + (4N_g \times (nk+1) \times (n(n-1)/2) \times (8|p| + 4k - 2))$. Thus, for any given Boolean circuit, when the security parameters $k$ and $|p|$ are fixed, the total number of bits generated randomly is a cubic function of $n$.

For example, consider the construction of a garbled circuit for adding two 32-bit numbers. The corresponding Boolean circuit has a total of $W = 439$ wires, $W_o = 33$ output wires, and $N_g = 375$ gates (The 32-bit adder circuit in [18] has 127 *AND* gates, 61 *XOR* gates and 187 *NOT* gates. Note that a *NOT* gate is equivalent to an *XOR* gate, since $NOT(x) = (1 \oplus x)$. As we have mentioned in Section 4.1, we consider only two different types of gates, namely, $\{AND, XOR\}$ in our work. Alternatively, note that more efficient constructions are possible; for example, an *AND* gate with a *NOT* gate on its output wire may be replaced with a single *NAND* gate, if the output wire of the *AND* gate is not an input wire for any other gate in the circuit).

Figure 5 shows the total number of Mbits that are generated randomly while creating one garbled table entry (*i.e.*, $b/(4N_g \times 2^{20})$) for the 32-bit adder. As an example, when $n = 5$, these parties collectively generate a total of 153.41 Mbits to create one garbled table entry.

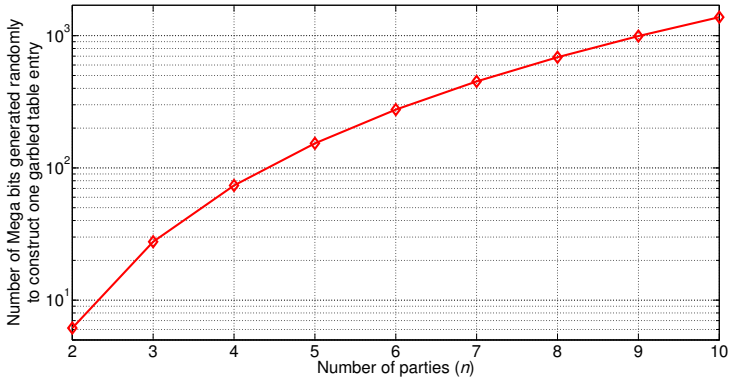


**Figure 5.** Total number of Mbits generated randomly while creating one garbled table entry for the 32-bit adder.

### *5.4. Cost of Evaluating the Garbled Circuit*

In order to perform the requested computation, the server $p_e$ obtains the garbled circuit, $GC$, from the server $p_c$. Let $N_g$ denote the total number of gates in the circuit $B$. Each entry in the garbled table has a length of $(nk+1)$ bits. Therefore, the size of the garbled circuit equals $4N_g \times (nk+1)$ bits.

Figure 6 shows the size of the garbled circuit in Kbits for the 32-bit adder. This circuit has $N_g = 375$ gates. The security parameter $k = 128$.

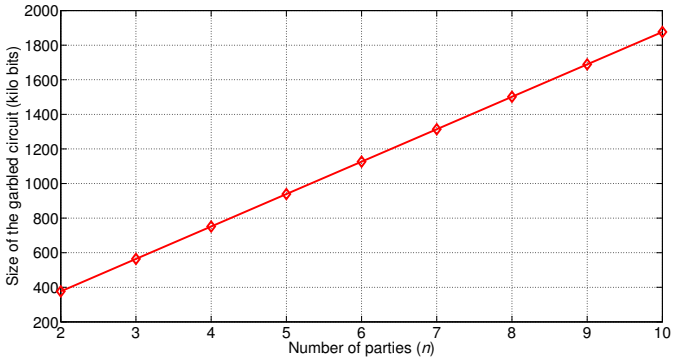


**Figure 6.** Size of the garbled circuit in Kbits for the 32-bit adder.

Let $W$ and $W_o$ denote the total number of wires and output wires, respectively, in the Boolean circuit $B$. During evaluation, for each non-output wire of the circuit, the server $p_e$ uses the PRNG $G$ $n$ times. Therefore, $G$ is used for a total of $(W - W_o)n$ times.

### *5.5. Cost for the Client*

In order to create the garbled circuit, the client provides: (i) a unique seed value, $s_i$, to each server $p_i$ $(1 \leq i \leq n)$; and (ii) a seed value, $s_{ik}$, to each pair of servers $(p_i, p_k)$, $(1 \leq i, k \leq n)$.

The length of each seed value, $|s_i| = |N|$ for the BBS PRNG. The length of each seed $|s_{ik}| = k$ for the PRNG $R$, which can be implemented using a block cipher such as AES in output feedback mode. Let $b_s$ denote the total number of bits that the client exchanges for the seed values. Then, $b_s = n|N| + n(n-1)k = n(|N| + (n-1)k)$.

For each plaintext input bit to the circuit, the client is required to generate the garbled input. Each garbled value is $(nk + 1)$ bits long, whose least significant bit depends on the $\lambda$ value, and the $\lambda$ value, in turn, depends on the 1-bit shares for the $n$ parties. Let $b_i$ denote the number of bits that the client needs to generate for each input wire. Then, $b_i = (nk + n)$.

The client needs to generate both possible garbled outputs for each output wire to enable the verification of outputs. Let $b_o$ denote the number of bits that the client needs to generate for each output wire. Then, $b_o = (2nk + n)$.

Let $W_i$ and $W_o$ denote the number of input and output wires, respectively, in the Boolean circuit $B$. Then, the client generates/exchanges a total of $b_s + W_i \times b_i + W_o \times b_o = n[|N| + (n-1)k + W_i(k+1) + W_o(2k+1)]$ bits.

Figure 7 shows the total number of Kbits that the client generates in order to enable the servers to construct and evaluate the garbled circuit, as well as for the verification of outputs for the 32-bit adder. This circuit has $W_i = 64$ input wires and $W_o = 33$ output wires. The security parameters are $k = 128$ and $|N| = 3072$.

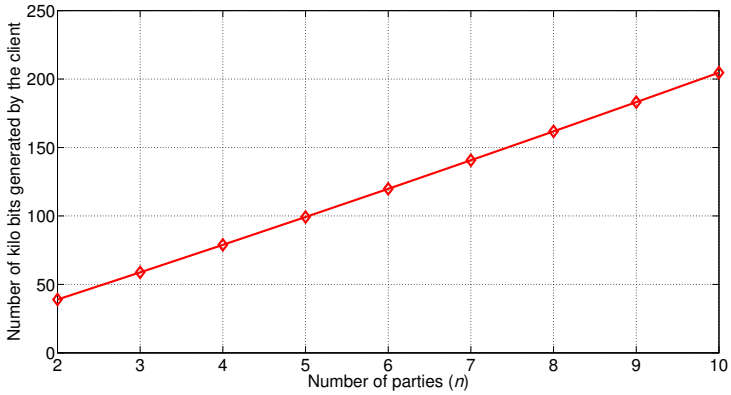


**Figure 7.** Total number of Kbits that the client generates to delegate the construction and evaluation of the garbled circuit, and to verify the outputs for the 32-bit adder.

For example, when $n = 10$, the client generates about 209 Kbits, if the security parameters are $k = 128$ and $|N| = 3072$. Furthermore, note that even when the security parameters are increased to $k = 256$ and $|N| = 15,360$, the client generates only 510 Kbits (The security parameters $k$ and $|N|$ are related; Barker *et al.* [17] provide a tabular listing of recommended combination of value-pairs $(k, |N|)$).

Comparing Figure 7 with Figures 4 and 5, we notice that while the servers generate and exchange Gigabytes of information to create the garbled circuit, the resource constrained mobile client, on the other hand, generates and exchanges only Kbytes of information with the evaluator and the other servers in the cloud.

While the network traffic among the servers is a cubic function of $n$ (Section 5.2), our result in Figure 7 shows that it is feasible to perform privacy preserving computations on behalf of resource-constrained clients, and that our system is well-suited for such clients, since they generate and exchange only compact cipher text messages with the servers - that are many orders of magnitude smaller than the amount of network traffic between the servers.

### *5.6. Comparison of Our Scheme with Gentry's FHE Scheme*

While Gentry's FHE scheme [3] requires a single server only, it, however, expects the client to exchange $O(k^5)$ bits with the evaluating server, for each input and output wire of the circuit. In our secure cloud computing system, for each input and output wire, the client exchanges $O(nk)$ bits only with the server $p_e$ since each garbled value has a length of $(nk + 1)$ bits. For example, the size of each encrypted plain text bit equals several Gigabits with Gentry's FHE scheme, while it equals a mere 641 bits in our approach with $n = 5$ and $k = 128$. Therefore, in comparison to FHE schemes, our approach is far more practical for cloud computing in mobile systems.

### *5.7. Construction and Evaluation Time*

We used BIGNUM routines and crypto functions from the OpenSSL library to implement our secure cloud computing system. We built our system as a collection of modules, and our implementation uses TCP (which stands for Transmission Control Protocol) sockets for communication among the servers. Using a server with Intel Xeon 2.53 GHz processor (Intel, Santa Clara, CA, USA), with 6 cores and 32 GB RAM we evaluated our system. Figure 8 shows the time taken to construct one garbled table for varying number of servers. We note that in order to significantly reduce the construction time, the garbled tables for any number of gates in the circuit can be constructed in parallel.

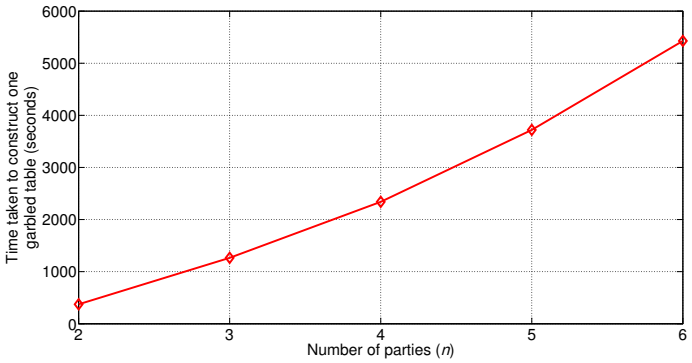


**Figure 8.** Time taken to construct one garbled table.

Figure 9 shows the time taken to evaluate one garbled gate for varying number of servers. Notice that evaluation is significantly faster than construction, and that the latter can be done offline. The evaluating server can readily carry out the requested computation, and therefore drastically reduce the response time for the mobile client, if the garbled circuits are pre-computed and made available to the evaluator, in advance.

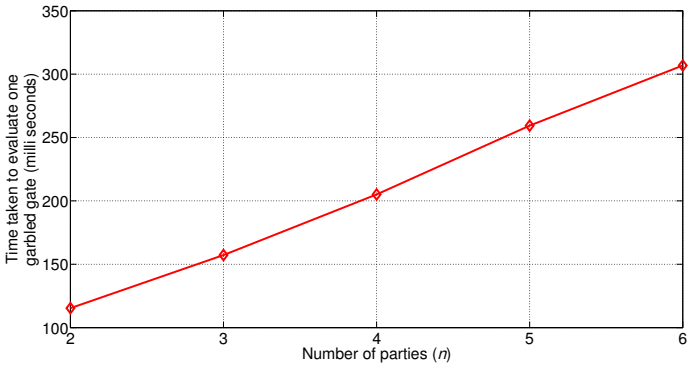


**Figure 9.** Time taken to evaluate one garbled gate.

While our prototype is implemented entirely in software, we note that it is possible to achieve significant speedup through special dedicated hardware blocks. For example, our method relies on the computation of pseudorandom functions such as AES, which can be expressed as a Boolean circuit with only *AND* gates and *XOR* gates [18]. Therefore, using hardware blocks for realizing the computation of pseudorandom functions that are used very frequently in our system can dramatically improve the overall time taken for performing privacy-preserving computations on the cloud on behalf of the client.

## 6. Privacy Preserving Search for the Nearest Bank/ATM

In this section, we consider the special scenario of a mobile device seeking to obtain the result of a computation that depends on its own "private inputs", and also on "public inputs" obtained from other servers on the Internet. More specifically, we examine the following privacy preserving application. A mobile client, which is located at the intersection of two streets, needs to determine the location of the nearest Chase or Wells Fargo banks or an ATM machine in a privacy-preserving manner. We evaluate our application using real-world data available for Salt Lake City, UT, whose streets are arranged in a "grid pattern". Our application assures the privacy of the following: (i) the mobile client's input location; (ii) the computed bank/ATM location nearest to the client; and (iii) the computed distance to the nearest ATM. Note that these secrets are revealed to the evaluator only if it colludes with all the $n$ servers that participate in the creation of the garbled circuit.

We consider an area of Salt Lake City, UT that lies between the Main Street (which represents the 0 East Street), the 1300 East Street, the South Temple Street (which represents the 0 South Street), and the 800 South Street. This area consists of $L = 10$ ATM locations that are shown in Table 2.

**Table 2.** Locations of Banks and ATMs in Salt Lake City, UT *(source: Chase [19], Wells Fargo [20]).*

| Bank/ATM | Location |
|---|---|
| Chase | 201 South 0 East |
| Chase | 185 South 100 East |
| Chase | 376 East 400 South |
| Chase | 531 East 400 South |
| Wells Fargo | 299 South 0 East |
| Wells Fargo | 381 East 300 South |
| Wells Fargo | 79 South 0 East |
| Wells Fargo | 778 South 0 East |
| Wells Fargo | 570 South 700 East |
| Wells Fargo | 235 South 1300 East |

Each East/South coordinate in this area is an $l = max(\lceil log_2 1300 \rceil, \lceil log_2 800 \rceil) = 11$-bit unsigned number. Therefore, the location of the mobile client at an intersection, or any bank/ATM in this area can be identified using $L_{ind} = 2l = 22$ bits.

### *6.1. Circuit for Computing Manhattan Distance*

Let $(x_a, y_a)$ represent the coordinates of the mobile client at an intersection. Similarly, let $(x_b, y_b)$ represent the coordinates of a bank/ATM. Since the streets are arranged in a grid pattern, the shortest distance ($D$) between $(x_a, y_a)$ and $(x_b, y_b)$ equals the sum of the absolute differences between the respective coordinates: $D = |x_a - x_b| + |y_a - y_b|$. This distance metric is more commonly referred to as the Manhattan distance.

We design a Boolean circuit for computing the Manhattan distance between two points, $(x_a, y_a)$ and $(x_b, y_b)$. Assume that each coordinate is an $l$ bit unsigned number. Figures 10 and 11 show the block diagram of our circuit. We use the *SUB* and *ADD* blocks of Kolesnikov *et al.* [21].

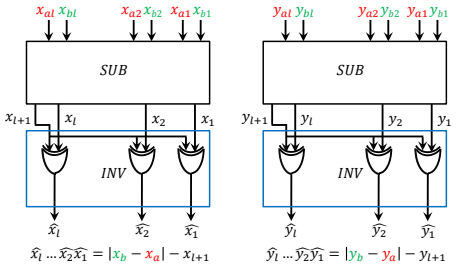


**Figure 10.** Absolute difference between the $X$ and $Y$ coordinates.

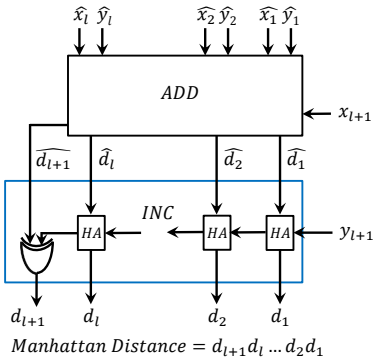


**Figure 11.** Manhattan Distance Computation.

Each $SUB/ADD$ block is composed of $l$ 1-bit subtractors/adders; each 1-bit subtractor/adder, in turn, is composed of 4 $XOR$ gates, 1 $AND$ gate. Note that $(l+1)^{th}$ output bit of $SUB$ block equals the complement of carry-out bit from the $l^{th}$ 1-bit subtractor.

If $x_a \geq x_b$, then the output of $SUB$ block equals the absolute difference, $|x_a - x_b|$. Otherwise, the output of $SUB$ block equals the negative value, $-|x_a - x_b|$, in $2's$ complement form. Since $x_{l+1} = 1$ for negative values, we use $x_{l+1}$ as one of the inputs for the $XOR$ gates in the $INV$ block to compute the $1's$ complement of the absolute difference. Then, we subsequently use $x_{l+1}$ as a carry-in input bit for the $ADD$ block. Thus, the output of the $ADD$ block accounts for both $x_a \geq x_b$ and $x_a < x_b$ cases.

Similarly, for the $Y$ coordinates we use the $SUB$ and $INV$ blocks to compute the absolute difference in $1's$ complement form. To account for the case when $y_{l+1} = 1$, we use an $INC$ block that adds the value of the bit $y_{l+1}$ to the output of the $ADD$ block. The $INC$ block is composed of $l$ half-adder ($HA$) blocks, where each $HA$ block, in turn, is composed of 1 $XOR$ and 1 $AND$ gate.

The output of the $INC$ block represents the $(l+1)$-bit Manhattan distance between $(x_a, y_a)$ and $(x_b, y_b)$. Using the circuit design of Figures 10 and 11, a list of distances between the location of the mobile client and any number ($L$) of ATM locations can be computed. Table 3 shows that our Boolean circuit for computing the Manhattan distance between two points has a total of $(15l+1)$ $XOR$ gates and $4l$ $AND$ gates.

**Table 3.** Circuit size for Manhattan distance calculation.

| Block | #*XOR* Gates | #*AND* Gates |
|---|---|---|
| $2 \times SUB$ | $2 \times 4l$ | $2 \times l$ |
| $2 \times INV$ | $2 \times l$ | 0 |
| $ADD$ | $4l$ | $l$ |
| $INC$ | $l+1$ | $l$ |
| $Total$ | $15l+1$ | $4l$ |

Note: In a more direct alternative to compute Manhattan distance, we may first find $min(x_a, x_b)$ and $max(x_a, x_b)$, and always subtract $min(x_a, x_b)$ from $max(x_a, x_b)$ (similarly for the $Y$ coordinates). While this approach would eliminate $INV$ and $INC$ blocks, it however would require the use of two comparator and conditional swap blocks [22], which together introduce $12l$ new $XOR$ gates and $4l$ $AND$ gates. Consequently, this alternative approach to compute Manhattan distance would require a total of $24l$ $XOR$ gates and $7l$ $AND$ gates. Thus, in comparison to this more direct alternative, our design shown above in Figure 10 and Figure 11 requires a significantly smaller number of $(15l+1)$ $XOR$ and $4l$ $AND$ gates.

### 6.2. Circuit for Computing the Nearest ATM

Following the computation of distance between the mobile client and $L$ ATM locations, it is necessary to find the nearest ATM, along with its distance. We use the approach of Kolesnikov *et al.* [21] to find the minimum value and its index, given a list of values. Kolesnikov *et al.* [21] have designed a $MIN$ block to find the minimum of two input values – it uses the result of a comparator to multiplex the minimum value, as well as the corresponding index as shown in Figure 12. Table 4 shows the size of the circuit that computes the minimum of two values, along with its index. In our privacy preserving application, each distance is an $(l+1) = 11+1 = 12$-bit number, and each index that identifies an ATM using its East and South coordinates is an $L_{ind} = 2l = 22$-bit number.

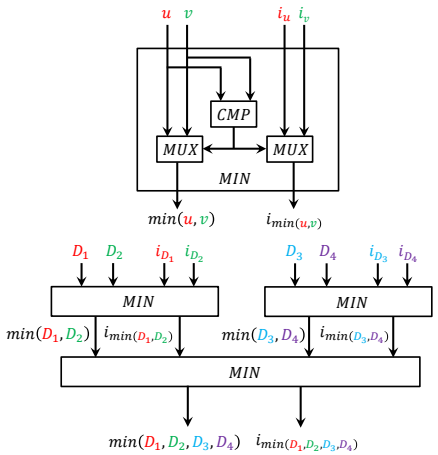


**Figure 12.** Finding minimum value and index.

**Table 4.** Circuit size of 1 Min block.

| Block | #*XOR* Gates | #*AND* Gates |
|---|---|---|
| $CMP$ | $3(l+1)$ | $l+1$ |
| $MUX_{min}$ | $2(l+1)$ | $l+1$ |
| $MUX_{index}$ | $2L_{ind}$ | $L_{ind}$ |
| $Total$ | $5(l+1)+2L_{ind}$ | $2(l+1)+L_{ind}$ |

Given $L$ values and their indices, using $(L-1)$ $MIN$ blocks organized as a tree, the minimum value and the corresponding index are propagated from the leaves to the root. Figure 12 shows the computation of the minimum of 4 input values, $D_1$, $D_2$, $D_3$, $D_4$, and the corresponding index, $i_{min(D_1,D_2,D_3,D_4)}$, as an example.

Table 5 shows the number of $XOR$ and $AND$ gates in the complete circuit that computes the nearest ATM location. It shows that in our privacy preserving application of finding the nearest Chase or Wells Fargo ATM in Salt Lake City, the circuit has a total of 2596 $XOR$ and 854 $AND$ gates.

**Table 5.** Complete circuit size for nearest ATM. $DIST(L)$ denotes distance to $L$ locations. $T(MIN)$ denotes the tree of *MIN* blocks.

| Block | #*XOR* Gates | #*AND* Gates |
|---|---|---|
| $DIST(L)$ | $v_1 = (15l+1)L$ | $v_3 = 4lL$ |
| $T(MIN)$ | $v_2 = [5(l+1)+2L_{ind}] \times (L-1)$ | $v_4 = [2(l+1)+L_{ind}] \times (L-1)$ |
| $Total$ | $v_1 + v_2$ | $v_3 + v_4$ |
| $Application$ | 2596 | 854 |

### *6.3. Server-Side and Client-Side Cost*

Figure 13 shows the network traffic as a function of the number of servers ($n$) involved in the creation of the garbled circuit that can compute the nearest ATM and the corresponding distance in a privacy preserving manner. For example, with $n = 4$ servers, the servers exchange a total of 126 $GB$ of information to create the garbled circuit. This result demonstrates the feasibility of our approach for performing real-world privacy-preserving computations.

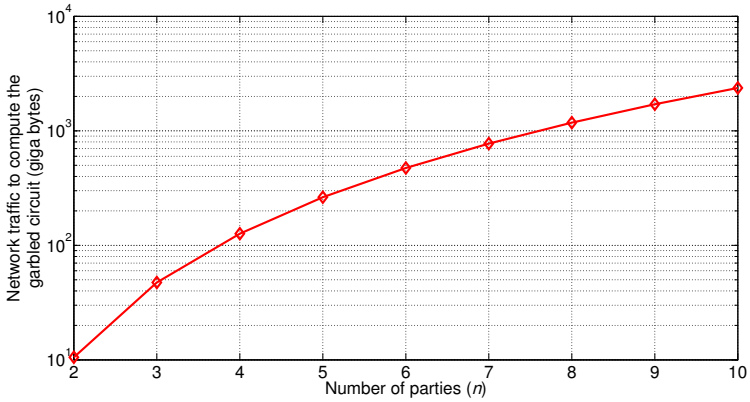


**Figure 13.** Server-side network traffic to construct the garbled circuit for determining the nearest ATM.

In order to facilitate the creation of the garbled circuit, and for the evaluation, the mobile client sends: (i) the seed values to the $n$ servers; and (ii) the garbled values representing the coordinates

of the input location to the evaluator. Since the ATM locations are publicly known, they are assumed to be hard-coded in the garbled circuit. *i.e.*, the client is not required to transmit the ATM locations to the servers.

Figure 14 shows the total number of bits generated by the client to delegate the privacy preserving computation of finding the nearest ATM and the distance from the client. To preserve location privacy, the client exchanges a very small amount of information with the servers—less than 60 Kbits, with $n = 4$ servers, for example. From Figures 13 and 14, we note that in comparison to the server-side cost, the client-side cost grows much slowly with the number of servers.

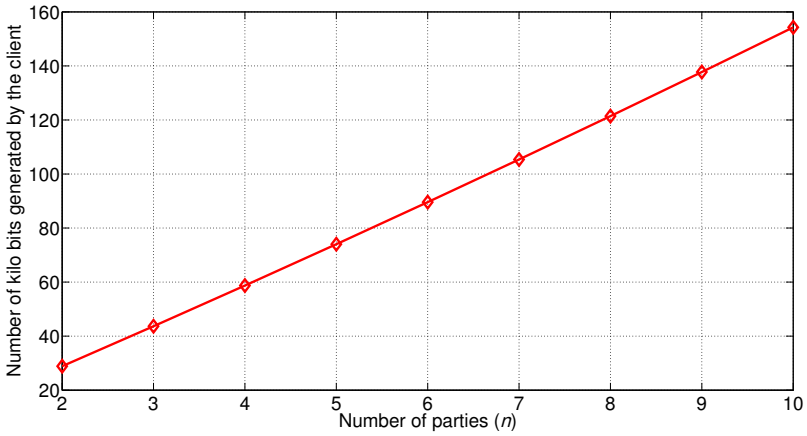


**Figure 14.** Client-side cost to determine the nearest ATM.

## 7. Related Work

Homomorphic encryption methods enable computations directly on the encrypted data, without requiring private decryption keys. For example, the product of two ciphertext messages produces a ciphertext corresponding to the product of the underlying plain text messages in the RSA public key system [2]. Domingo-Ferrer *et al.* [23] present an additive and multiplicative homomorphic scheme in which polynomials are used to represent ciphertext; however, multiplication operations drastically increase the size of the cipher text in their scheme. Recently, fully homomorphic encryption (FHE) schemes (e.g., [3–5]) have been proposed, which enable performing any arbitrary computation on encrypted data. However, FHE schemes are currently not suitable for mobile cloud computing applications due to extremely large cipher text size and computational requirements beyond the capabilities of mobile devices today. Our work presents a far better alternative suitable for mobile systems.

Yao's garbled circuits have been mainly used in the context of secure two-party computation [6,7,24]. Lindell *et al.* [25] present an extensive survey of secure multiparty computation, along with numerous potential applications—some examples of which are privacy preserving data mining, private set intersection, electronic voting and electronic auction. Over the years, a number of secure two-party and multiparty computation systems have been built (e.g., [26–28]). In secure multiparty computation systems, multiple parties hold private inputs and receive the result of the computation. While multiple parties participate in the creation of garbled circuits, only the client has the private inputs and obtains the result of the computation, in garbled form however, in our secure cloud computing system. As we have described in Section 4.2, in our work, we adapt secure multiparty computation protocols [8,9,11,12], for building a secure and verifiable cloud computing for mobile systems.

Some existing works [29,30] use arithmetic circuits for performing homomorphic addition and multiplication operations. However, Boolean circuits are sufficient for our purpose of creating the garbled table entries, since it only involves simple binary operations such as *XOR* and *AND* (Section 4.1.1). Additionally, Boolean circuits are more efficient than arithmetic circuits for

homomorphic comparison of numbers, which we have used abundantly in our privacy preserving search application (Section 6), since comparison using an arithmetic circuit involves a large number of multiplications and communication rounds among the participants [29,30].

Branching programs offer an alternate method for representing Boolean functions. Ames *et al.* [31] present a branching program formulation for secure health monitoring using homomorphic encryption. Ames *et al.* [31] present simulation results on real ECG (which stands for electrocardiogram) data, and note that their approach of using branching programs provides significant speedup in comparison to methods using traditional circuits. It could be a very interesting avenue for future research to explore the use of garbled circuits in conjunction with branching programs, especially for secure cloud computing applications and mobile big data systems.

In the Twin clouds [32] secure cloud computing architecture, the client uses a private cloud for creating garbled circuits and a public commodity cloud for evaluating them. Our solution, however, utilizes multiple public cloud servers for creating as well as evaluating the garbled circuits. Stated alternatively, our solution obviates the requirement of private cloud servers. Furthermore, in twin clouds, the privacy of the client data is lost if the evaluator colludes with the sole/only server that constructs the garbled circuit. In our work, on the other hand, the use of multiple servers for construction of garbled circuits offers greater resistance to collusion (Section 2).

While FHE schemes remain impractical currently, they, however, offer interesting constructions, such as reusable garbled circuits [33] and verifiable computing capabilities [34]. In our proposed system, we enable the client to efficiently verify whether a cloud server has actually evaluated the garbled circuit, without depending on any FHE scheme.

Naehrig *et al.* [35] present an implementation of a "somewhat" homomorphic encryption scheme, which is capable of computing simple statistics such as mean, standard deviation and regression, which require only addition operations, and a very few multiplication operations. Another recent work (P4P [36]) provides privacy preservation for computations involving summations only. In comparison, our scheme, which is based on Yao's garbled circuits, is not limited to computation of simple statistics or summations only; *i.e.*, our scheme is more generic, and can perform any arbitrary computation.

Carter *et al.* [37] have proposed a secure two party computation system, which however has three participants: Alice, Bob and a Proxy. In their work, Alice is a mobile device that delegates the task of evaluating the garbled circuits to the Proxy, which is a cloud server, and Bob is a webserver that creates garbled circuits. We note that our computation and adversary models are very different from that of Carter *et al.*'s work [37]. First, in their work, both Alice and Bob provide private inputs for the secure two party computation that they wish to perform jointly; however, in our work, only one party, i.e., the mobile client, provides inputs and obtains the result of the computation in garbled form. Second, Carter *et al.*'s [37] scheme requires that neither Alice nor Bob can collude with the Proxy; in a sharp contrast, even if the evaluating server colludes with all but one of the cloud servers that participated in the creation of the garbled circuit, our method can preserve the privacy of the client's data.

## 8. Concluding Remarks

We proposed a novel secure and verifiable cloud computing for mobile big data systems using multiple servers. Our method is a unique combination of the secure multiparty computation protocol of Goldreich *et al.* [8,9], the garbled circuit design of Beaver *et al.* [11,12], and the cryptographically secure pseudorandom number generation method of Blum *et al.* [13,14]. Even if the evaluator colludes with all but one of the servers that participated in the creation of the garbled circuit, our method can preserve the privacy of the mobile client's inputs and the results of the computation. Furthermore, our system can detect a cheating evaluator that returns arbitrary output without performing any computation at all. We presented the server-side and client-side complexity analysis of our system. Using real-world data, we evaluated our system for a privacy preserving search application that locates the nearest point of interest from the mobile client. We also evaluated the time taken to construct

and evaluate a garbled circuit for varying number of servers to demonstrate the feasibility of our proposed approach.

To summarize, our method shows how publicly available location information from different sources (e.g., from Wells Fargo, Chase) may be used in conjunction with private location information of a mobile device in a privacy preserving computation.

Our software implementation is available through our computing in the cloud website [38].

**Acknowledgments:** Our work is part of the National Science Foundation (NSF) Future Internet Architecture Project, and is supported by NSF under the following grant numbers: CNS-1040689, ECCS-1308208 and CNS-1352880. We thank Srinivasa Vamsi Laxman Perala for implementing various components of the system in Section 5.7.

**Author Contributions:** S.N.P. contributed to the design of the proposed scheme, analysis, formulation of the case-study, literature survey and manuscript preparation. Z.J.H. supervised this research work, contributed to the refinement of all aspects of this work, and also in enhancing the manscript.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I.; Zaharia, M. A view of cloud computing. *Commun. ACM* **2010**, *53*, 50–58.
2. Rivest, R.L.; Adleman, L.; Dertouzos, M.L. On data banks and privacy homomorphisms. *Found. Secur. Comput.* **1978**, *4*, 169–180.
3. Gentry, C. Computing arbitrary functions of encrypted data. *Commun. ACM* **2010**, *53*, 97–105.
4. Brakerski, Z.; Vaikuntanathan, V. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52Nd Annual Symposium on Foundations of Computer Science*; IEEE Computer Society: Washington, DC, USA, 2011; pp. 97–106.
5. Zhang, X.; Xu, C.; Jin, C.; Xie, R.; Zhao, J. Efficient fully homomorphic encryption from {RLWE} with an extension to a threshold encryption scheme. *Future Gener. Comput. Syst.* **2014**, *36*, 180–186.
6. Yao, A.C. Protocols for secure computations. In Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, SFCS '08, Chicago, IL, USA, 3–5 November 1982.
7. Yao, A.C.-C. How to generate and exchange secrets. In Proceedings of the 27th Annual Symposium on Foundations of Computer Science, Toronto, ON, Canada, 27–29 October 1986.
8. Goldreich, O. *Foundations of Cryptography: Volume 2, Basic Applications*; Cambridge University Press: Cambridge, UK, 2004.
9. Goldreich, O.; Micali, S.; Wigderson, A. How to play ANY mental game. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 25–27 May 1987; pp. 218–229.
10. Premnath, S.N.; Haas, Z.J. A Practical, Secure, and Verifiable Cloud Computing for Mobile Systems. In Proceedings of the 9th International Conference on Future Networks and Communications (FNC'14)/The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14)/Affiliated Workshops, Niagara Falls, Canada, 17–20 August 2014; Series Procedia Computer Science, Volume 34, pp. 474–483.
11. Beaver, D.; Micali, S.; Rogaway, P. The round complexity of secure protocols. In Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, 13–17 May 1990; pp. 503–513.
12. Rogaway, P. The Round Complexity of Secure Protocols. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 22 April 1991.
13. Blum, L.; Blum, M.; Shub, M. A simple unpredictable pseudo random number generator. *SIAM J. Comput.* **1986**, 15, 364–383.
14. Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed.; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1995.
15. Naor, M.; Pinkas, B. Efficient oblivious transfer protocols. In Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, USA, 7–9 January 2001; pp. 448–457.
16. Naor, M.; Pinkas, B. Computationally secure oblivious transfer. *J. Cryptol.* **2005**, *18*, 1–35.

17. Barker, E.; Barker, W.; Burr, W.; Polk, W.; Smid, M. Recommendation for key management—Part 1: General (Revision 3). *NIST Spec. Publ.* **2012**, 700–857. Available online: http://www.amazon.com/Recommendation-Key-Management-General-Revision/dp/1499160224 (accessed on 5 May 2016).
18. Tillich, S.; Smart, N. Circuits of Basic Functions Suitable For MPC and FHE. Available online: http://www.cs.bris.ac.uk/Research /CryptographySecurity/MPC/ (accessed on 15 April 2014).
19. Chase. Available online: http://www.chase.com (accessed on 15 April 2014).
20. Wells Fargo. Available online: http://www.wellsfargo.com (accessed on 15 April 2014).
21. Kolesnikov, V.; Sadeghi, A.R.; Schneider, T. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1–20.
22. Huang, Y.; Evans, D.; Katz, J. Private set intersection: Are garbled circuits better than custom protocols. In *NDSS*; The Internet Society: San Diego, CA, USA, 2012.
23. Domingo-Ferrer, J. A provably secure additive and multiplicative privacy homomorphism. In *Information Security*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 471–483.
24. Lindell, Y.; Pinkas, B. A proof of security of yao's protocol for two-party computation. *J. Cryptol.* **2009**, *22*, 168–188.
25. Lindell, Y.; Pinkas, B. Secure multiparty computation for privacy-preserving data mining. *J. Priv. Confid.* **2009**, *1*, 59–98.
26. Henecka, W.; Sadeghi, A.R.; Schneider, T.; Wehrenberg, I. TASTY: Tool for automating secure two-party computations. In Proceedings of the 17th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 4–8 October 2010; pp. 451–462.
27. Malkhi, D.; Nisan, N.; Pinkas, B.; Sella, Y. Fairplay—A secure two-party computation system. In *USENIX Security Symposium*; USENIX Association: San Diego, CA, USA, 2004.
28. Ben-David, A.; Nisan, N.; Pinkas, B. FairplayMP: A system for secure multi-party computation. In Proceedings of the 15th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 27–31 October 2008; pp. 257–266.
29. Burkhart, M.; Strasser, M.; Many, D.; Dimitropoulos, X. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In Proceedings of the 19th USENIX conference on Security, Washington, DC, USA, 11–13 August 2010.
30. Damgård, I.; Pastro, V.; Smart, N.; Zakarias, S. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology–CRYPTO 2012*; Springer: Berlin/Heidelberg, Germany, 2012.
31. Ames, S.; Venkitasubramaniam, M.; Page, A.; Kocabas, O.; Soyata, T. Secure health monitoring in the cloud using homomorphic encryption, a branching-program formulation. In *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*; Soyata, T., Ed.; IGI Global: Hershey, PA, USA, 2015; pp. 116–152.
32. Bugiel, S.; Nürnberger, S.; Sadeghi, A.R.; Schneider, T. Twin clouds: Secure cloud computing with low latency. In *Communications and Multimedia Security*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 32–44.
33. Goldwasser, S.; Kalai, Y.; Popa, R.A.; Vaikuntanathan, V.; Zeldovich, N. Reusable garbled circuits and succinct functional encryption. In Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, Palo Alto, CA, USA, 1–4 June 2013; pp. 555–564.
34. Gennaro, R.; Gentry, C.; Parno, B. Non interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 465–482.
35. Naehrig, M.; Lauter, K.; Vaikuntanathan, V. Can homomorphic encryption be practical? In Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, Chicago, IL, USA, 17–21 October 2011; pp. 113–124.
36. Duan, Y.; Canny, J.; Zhan, J. P4P: Practical large-scale privacy-preserving distributed computation robust against malicious users. In Proceedings of the 19th USENIX conference on Security, Washington, DC, USA, 11–13 August 2010.
37. Carter, H.; Mood, B.; Traynor, P.; Butler, K. Secure outsourced garbled circuit evaluation for mobile devices. In Proceedings of the 22nd USENIX Conference on Security, Washington, DC, USA, 14–16 August 2013.

38. Computing in the Cloud. Available online: https://people.ece.cornell.edu/haas/wnl/CiC (accessed on 15 July 2014).