Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 34 (2014) 474 – 483

Procedia Computer Science

The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC-2014)

A Practical, Secure, and Verifiable Cloud Computing for Mobile Systems

Sriram N. Premnath[a], Zygmunt J. Haas[a,b,*]

[a]*Cornell University, 324 Frank Rhodes Hall, Ithaca, NY 14853, U.S.A.*
[b]*University of Texas at Dallas, 800 W. Campbell Road, Richardson, TX 75080, U.S.A.*

Abstract

Cloud computing systems, in which clients rent and share computing resources of third party platforms, have gained widespread use in recent years. Furthermore, cloud computing for mobile systems (i.e., systems in which the clients are mobile devices) have too been receiving considerable attention in technical literature. We propose a new method of delegating computations of resource-constrained mobile clients, in which multiple servers interact to construct an encrypted program known as garbled circuit. Next, using garbled inputs from a mobile client, another server executes this garbled circuit and returns the resulting garbled outputs. Our system assures privacy of the mobile client's data, even if the executing server chooses to collude with all but one of the other servers. We adapt the garbled circuit design of Beaver et al. and the secure multiparty computation protocol of Goldreich et al. for the purpose of building a secure cloud computing for mobile systems. Our method incorporates the novel use of the cryptographically secure pseudo random number generator of Blum et al. that enables the mobile client to efficiently retrieve the result of the computation, as well as to verify that the evaluator actually performed the computation. We analyze the server-side and client-side complexity of our system. Using real-world data, we evaluate our system for a privacy preserving search application that locates the nearest bank/ATM from the mobile client. We also measure the time taken to construct and evaluate the garbled circuit for varying number of servers, demonstrating the feasibility of our secure and verifiable cloud computing for mobile systems.
© 2014 Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/3.0/).
Selection and peer-review under responsibility of Conference Program Chairs
Keywords: Secure Cloud Computing; Garbled Circuits, Secure Multiparty Computation

1. Introduction

Cloud computing systems, in which the clients rent and share computing resources of third party platforms such as Amazon Elastic Cloud, Microsoft Azure, etc., have gained widespread use in recent years. Provisioned with a large pool of hardware and software resources, these cloud computing systems enable clients to perform computations on a vast amount of data without setting up their own infrastructure[1]. However, providing the cloud service provider with the client data in *plaintext form* to carry out the computations will result in complete loss of data privacy.

Homomorphic encryption[2] is an approach to tackle the problem of preserving data privacy, which can allow the cloud service providers to perform specific computations directly on the encrypted client data, without requiring

* Corresponding author. Tel.: +1-607-255-3454 ; fax: +1-607-255-9072.
E-mail address: haas@ece.cornell.edu

1877-0509 © 2014 Elsevier B.V. This is an open access article under the CC BY-NC-ND license
(http://creativecommons.org/licenses/by-nc-nd/3.0/).
Selection and peer-review under responsibility of Conference Program Chairs
doi:10.1016/j.procs.2014.07.033

private decryption keys. Recently, fully homomorphic encryption (FHE) schemes (e.g., Gentry et al.[3]) have been proposed, which enable performing any arbitrary computation on encrypted data. *However, FHE schemes are currently impractical for mobile cloud computing applications due to extremely large cipher text size.* For instance, to achieve 128-bit security, the client is required to exchange a *few Giga bytes of ciphertext* with the cloud server, for each bit of the plain text message[3]. *Thus, there is a need for a more efficient alternative, which is suitable for mobile systems.*

Yao's garbled circuits approach[4,5], which we consider in our work, is a potential alternative to FHE schemes that can drastically reduce the ciphertext size. Any computation can be represented using a Boolean circuit, for which, there exists a corresponding garbled circuit[4,5,6,7]. Each gate in a garbled circuit can be unlocked using a pair of input *wire keys* that correspond to the underlying plaintext bits; and the association between the wire keys and the plaintext bits is kept secret from the cloud server that performs the computation. Unlocking a gate using a pair of input wire keys reveals 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. Thus, garbled circuits can enable *oblivious evaluation* of any arbitrary function, expressible as a Boolean circuit, on a third-party cloud server.

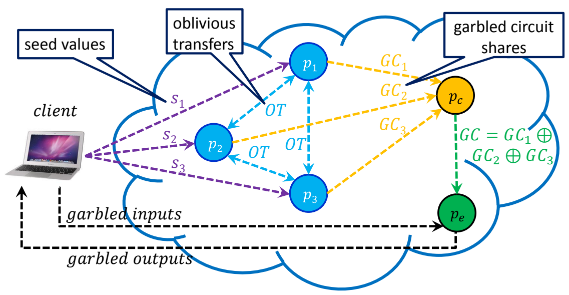
While garbled circuits preserve the privacy of client data, they are, however, one time programs – using the same version of the circuit more than once compromises the garbled circuit and reveals to an adversarial evaluator whether the semantics have changed or remained the same for a set of input and output wires between successive evaluations. Expecting the client to create a new version of the garbled circuit for each evaluation, however, is an unreasonable solution, since creating a garbled circuit is at least as expensive as evaluating the underlying Boolean circuit! *Thus, in contrast to FHE schemes such as that of Gentry[3], that 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.*

We propose a new method, in which whenever the client needs to perform a computation, it employs a number of cloud servers to create a new version of the garbled circuit in a distributed manner. Each server generates a set of private input bits using unique seed value from the client 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 engage in a secure multiparty computation protocol (e.g., Goldreich et al.[6,7]) to construct the desired garbled circuit without revealing their private inputs to one another. Once a new version of the garbled circuit is created using multiple servers, the client delegates the evaluation to an arbitrary server in the cloud. 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, even if it chooses to collude with any strict-subset of servers that participated in the creation of the garbled circuit.

Our proposed system is designed to readily exploit the real-world asymmetry that exists between typical mobile clients and cloud servers – while the mobile clients are *resource-constrained*, the cloud servers, on the other hand, are sufficiently provisioned to perform numerous intensive computation and communication tasks. To achieve secure and verifiable computing capability, our system requires very little computation and communication involvement from the mobile client beyond the generation and exchange of *compact cipher text messages*. However, using significantly larger resources, the cloud servers can efficiently generate and exchange a large volume of random bits necessary for carrying out the delegated computation. *Thus, our proposed scheme is very suitable for mobile environments.*

We adapt the garbled circuit design of Beaver, Micali, Rogaway (BMR[8,9]), and the secure multiparty computation protocol of Goldreich et al.[6,7] to suit them for the purpose of building a secure cloud computing system. To facilitate the construction of the garbled circuit, and also to enable the client to *efficiently retrieve and verify the result of the computation*, our method incorporates the novel use of the cryptographically secure pseudo random number generator of Blum, Blum, Shub[10,11], whose strength relies on the computational difficulty of factorizing large numbers into primes. *Our proposed system enables the client to efficiently verify that the evaluator actually and fully performed the requested computation.*

Our major contributions in this work include the following: (i) we design a secure mobile cloud computing system using multiple servers that enables the client to delegate any arbitrary computation, (ii) our system assures the privacy of the client input and the result of the computation, even if the evaluating server colludes with all but one of the servers that created the garbled circuit, (iii) our system enables the client to efficiently recover the result of the computation and to verify whether the evaluator actually performed the computation, (iv) we present an analysis of the server-side and client-side complexity of our proposed scheme. Our findings show that in comparison to Gentry's FHE scheme, our scheme uses very small cipher text messages suitable for mobile clients, (v) using real-world data, we evaluate our



1. Client sends seed values s_1, s_2, s_3 to p_1, p_2, p_3 respectively.
2. p_1, p_2, p_3 interact with one another to construct their shares of the garbled circuit, GC.
3. p_1, p_2, p_3 send their shares GC_1, GC_2, GC_3, respectively, 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.
6. p_e evaluates GC, and sends the garbled outputs to the client.

Fig. 1: Our secure cloud computing model with a mobile client using $(n + 2) = 5$ servers, $\{p_1, p_2, p_3, p_c, p_e\}$.

system for a privacy preserving search application that locates the nearest bank/ATM from the mobile client, and (vi) we measure the time taken to construct and evaluate the garbled circuit for varying number of servers, demonstrating the feasibility of our system.

2. A High-level Overview of our System

In the proposed system, the client employs a set of $(n + 2)$ servers, $\{p_1, p_2, \ldots, p_n, p_c, p_e\}$. Initially, the client sends a description of the desired computation (e.g., addition of two numbers), and a unique seed value s_i to each server $p_i, (1 \leq i \leq n)$. Each of these n servers first creates (or retrieves from its repository, if available already) a Boolean circuit (B) that corresponds to the requested computation. Using the unique seed value s_i, each server $p_i, (1 \leq i \leq n)$ generates a private pseudorandom bit sequence whose length is proportional to the total number of wires in the Boolean circuit (B). Then, using the private pseudorandom bit sequences and the Boolean circuit (B) as inputs, these n servers interact with one another and 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 garbled circuit.

Once the shares for the garbled circuit are created, the client requests each server, $p_i, (1 \leq i \leq n)$, to send its share, GC_i, to the server p_c. Performing 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.

Now, using the unique seed values $s_i, (1 \leq i \leq n)$, the client generates on its own garbled input values for each input wire in the circuit and sends them to the server p_e for evaluation. Using these garbled inputs, the server p_e unlocks the gates in the first level of the circuit to obtain the corresponding garbled outputs, which, in turn, unlocks the gates in the second level of the circuit, and so on. In this manner, the server p_e unlocks all the gates in the circuit, obtains the garbled outputs of the circuit, and sends them to the client. The client now converts these garbled output values into plaintext bits to recover the result of the desired computation. Fig. 1 depicts an overview of our proposed system.

2.1. *Our Adversary Model*

We assume the existence of a secure communication channel between the client and each of the $(n + 2)$ servers, $\{p_1, p_2, \ldots, p_n, p_c, p_e\}$, for sending unique seed values for pseudorandom bit generation, identity of the other servers, etc. We assume that all pairs of communicating servers authenticate one another. We assume a very capable adversary, where the evaluator p_e may individually collude with any proper subset of the n servers, $\{p_1, p_2, \ldots, p_n\}$, and still remain unable to determine the semantics of any garbled value that the evaluator observes during evaluation. Thus, our adversary model depicts 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 attempts to eavesdrop and analyze the set of all message exchanges between different parties, and also analyze the set of all the messages that it has legitimately received from the other parties, it still cannot determine the shares of the other parties, or the semantics of the garbled value pairs that are assigned to each wire in the circuit. Further, if the evaluator, p_e, returns arbitrary numbers as outputs to the client, the client can detect this efficiently. In our model, a new garbled circuit is created for every evaluation. This prevents an adversarial evaluator from determining the set of inputs and outputs that have changed or remained the same between different evaluations.

3. Secure and Verifiable Cloud Computing for Mobile Systems

In this section, we briefly present how we use, adapt, and combine the various cryptographic constructions, as well as present a summary of our proposed system.

3.1. *Oblivious Transfer Protocol*

There are two parties, a sender and a chooser. The sender holds four private messages, M_{00}, M_{01}, M_{10}, M_{11}, and the chooser holds two private choice bits, σ_1, σ_2. At the end of the 1-out-of-4 oblivious transfer (OT) protocol[12,13], the chooser learns $M_{\sigma_1\sigma_2}$ only, while the sender learns nothing.

Essentially, the sender and chooser use their private messages and choice bits as inputs to perform a small constant number of pseudorandom function computations such as AES, SHA, modular arithmetic operations in a multiplicative sub-group of Z_p^*, where p is a safe-prime, and interact with one another to realize the OT functionality[12,13].

3.2. *Secure Multiparty Computation Protocol*

We briefly describe the secure multipary computation protocol of Goldreich[6,7], which allows n parties to compute any 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 $f(x_1, x_2, \ldots, x_n)$ is expressed as a Boolean circuit (B') using a set of XOR and AND gates.

Splitting and sharing: For each wire in the Boolean circuit, the actual binary value corresponds to the XOR-sum of shares of all the n parties. In the original protocol, each party splits and shares each of its private input bits with all the other parties over pairwise secure communication channels. In our approach, 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 generates $r_{kj}, (\forall k \neq i)$, using the seed value s_{ik}. More 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 circuit. Likewise, party p_k sets its own share as $r_{kj} = R(s_{ik}, j, gate_{id}, entry_{id})$. *Thus, our approach eliminates the exchange of a very large number of bits.*

XOR gates: Evaluation of each XOR gate in the circuit is carried out locally. Specifically, each party merely performs an XOR operation over its shares for the two input wires to obtain its share for the output wire.

AND gates: Evaluation of each AND gate in the circuit, on the other hand, requires interaction between all pairs of parties. For the two inputs wires to 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 XOR-sum of the shares for the output wire 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 locally computes $((a_i.b_i).I)$; and the computation of each *partial-product*, $((a_i \oplus a_j).(b_i \oplus b_j))$, is accomplished using 1-out-of-4 OT[12,13] between p_i and p_j, such that no party reveals its shares to the other party[6,7].

Following the above procedure, the n parties evaluate every gate in the circuit. Thus, in the end, each server $p_i, (1 \leq i \leq n)$, obtains the share, $(f(x_1, x_2, \ldots, x_n))_i$, such that $f(x_1, x_2, \ldots, x_n) = \bigoplus_{i=1}^{n}(f(x_1, x_2, \ldots, x_n))_i$.

3.3. *Garbled Circuits – for Secure Computation*

In the following, we describe the construction and evaluation of the garbled circuit, GC. In our work, we use an enhanced variant of the original Yao's garbled circuits[4,5], namely, the garbled circuit design from Beaver, Micali, Rogaway (BMR[8,9]), and adapt it for the purpose of building a secure and verifiable cloud computing system.

3.3.1. *Construction of GC*

Garbled Value Pairs: Each wire in the circuit is associated with a pair of garbled values representing the underlying plaintext bits 0 and 1. Let A denote a specific gate in the circuit, whose two input wires are x, y; and whose output wire is z. Let (α_0, α_1), (β_0, β_1) and (γ_0, γ_1) denote the pair of garbled values associated with the wires x, y and z, respectively. Note that $LSB(\alpha_0) = LSB(\beta_0) = LSB(\gamma_0) = 0$ and $LSB(\alpha_1) = LSB(\beta_1) = LSB(\gamma_1) = 1$.

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

$\underline{\lambda \text{ Value:}}$ Each wire in the circuit is also associated with a 1-bit λ value that determines the semantics for the pair of garbled values. Specifically, the garbled value whose $LSB = b$ represents the underlying plaintext bit $(b \oplus \lambda)$.

Collusion Resistance: Let $\lambda_x, \lambda_y, \lambda_z$ denote the λ values for the 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 $p_i, (1 \le i \le n)$. Note that the λ value of each wire is unknown to any individual server. Consequently, the evaluator of the garbled circuit must collude with all 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, $[A_{00}, A_{01}, A_{10}, A_{11}]$, which represents the garbled table for gate A. Let $\otimes \in \{XOR, AND\}$ denote the binary operation of gate A. Then, the value of one specific entry, $A_{ab} = \gamma_{[((\lambda_x \oplus a) \otimes (\lambda_y \oplus b)) \oplus \lambda_z]} \oplus [G_b(\alpha_{a1}) \oplus G_b(\alpha_{a2}) \oplus \ldots \oplus G_b(\alpha_{an})] \oplus [G_a(\beta_{b1}) \oplus G_a(\beta_{b2}) \oplus \ldots \oplus G_a(\beta_{bn})]$, where G_a and G_b are pseudorandom functions that expand k bits into $(nk + 1)$ bits. Specifically, let G denote 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. Then, $G_0(s)$ and $G_1(s)$ denote the first and last $(nk + 1)$ bits of $G(s)$ respectively.

To compute a garbled table entry A_{ab}, such as the one shown above, the n servers use the secure multiparty computation protocol of Goldreich[6,7] (Section 3.2), where $f(x_1, x_2, \ldots, x_n) = A_{ab}$, and for each server, $p_i, (1 \le i \le n)$, 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. In this manner, the n servers jointly compute each entry in the garbled table for each gate in the circuit.

3.3.2. Evaluation of GC

Let α, β denote the garbled values for the two input wires of a gate A during evaluation. Let a, b denote the LSB values of α, β respectively. Let the ordered list $[A_{00}, A_{01}, A_{10}, A_{11}]$ denote the garbled table for gate A. Then, the garbled value for the output wire, γ, is recovered using α, β, A_{ab}, as shown in the two-step process below:

1. split the most significant nk bits of α into n parts, $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n$, each with k bits; similarly, split the most significant nk bits of β into n parts, $\beta_1, \beta_2, \beta_3, \ldots, \beta_n$, each with k bits; i.e., $|\alpha_i| = |\beta_i| = k$, where $1 \le i \le n$.
2. compute $\gamma = [G_b(\alpha_1) \oplus G_b(\alpha_2) \oplus \ldots \oplus G_b(\alpha_n)] \oplus [G_a(\beta_1) \oplus G_a(\beta_2) \oplus \ldots \oplus G_a(\beta_n)] \oplus A_{ab}$.

Thus, 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, $p_i, (1 \le i \le n)$, the server p_e can perform the evaluation *independently*.

3.4. Cryptographically Secure Pseudorandom Number Generation – for Recovery and Verification of Outputs

We address the following questions in this subsection. *First, how does the client efficiently recover the result of the computation without itself having to repeat the delegated computations? Second, how does the client verify that the garbled circuit is actually evaluated? In other words, is it possible for the client to determine cheating if the evaluator returns arbitrary numbers as output without carrying out any computation at all?*

We can enable the client to efficiently retrieve and verify the outputs returned by the evaluator, p_e. To achieve 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[10,11], as we have outlined below.

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

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

Each wire ω in the circuit is associated with a pair of garbled values, (ω_0, ω_1), and a 1-bit λ_ω value. Then, $\omega_0 = \omega_{01}||\omega_{02}||\omega_{03}||\ldots||\omega_{0n}||0$, $\omega_1 = \omega_{11}||\omega_{12}||\omega_{13}||\ldots||\omega_{1n}||1$; and $\lambda_\omega = \lambda_{\omega 1} \oplus \lambda_{\omega 2} \oplus \lambda_{\omega 3} \oplus \ldots \oplus \lambda_{\omega n}$. In these three expressions, ω_{0i}, ω_{1i} and $\lambda_{\omega i}$ are shares of the party $p_i, (1 \le i \le n)$. Note that $|\omega_{0i}| = |\omega_{1i}| = k$, and $|\lambda_{\omega i}| = 1$.

For each wire $\omega, (0 \le \omega \le W - 1)$, in the circuit, each party, $p_i, (1 \le i \le n)$, needs to generate $(2k + 1)$ pseudo random bits, where W denotes the total number of wires in the circuit. Thus, each party, p_i, generates a total of $(W(2k + 1))$ pseudorandom bits.

Party p_i generates its shares ω_{0i}, ω_{1i}, and $\lambda_{\omega i}$ for wire ω as a concatenation of the $b_{i,j}$ values, where the indices j belong to the range: $[(\omega(2k+1)+1), (\omega+1)(2k+1)]$. For concise notation, let $\Omega_{\omega k} = \omega(2k+1)$. 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: Notice that each party p_i is required to compute all the previous $(j-1)$ bits before it can compute the j^{th} bit. However, using its 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*, where $C(N)$ denotes the *Carmichael function*, which equals the least common multiple of $(p-1)$ and $(q-1)$.

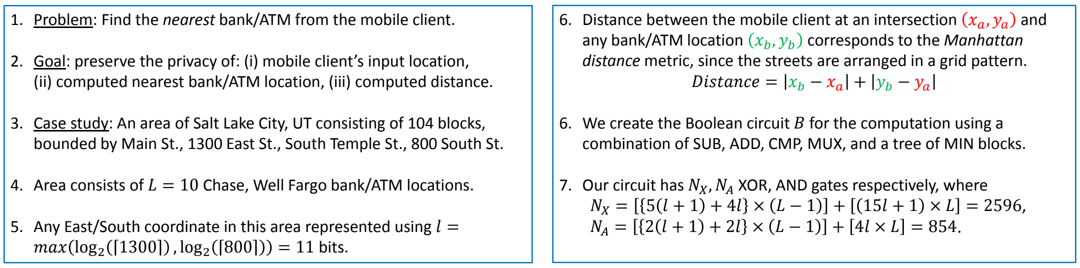
Recovery and Verification: *Therefore, using the secret values p, q, the client can readily compute ω_0, ω_1, and λ_ω for any output wire ω in the circuit; i.e., without having to compute ω_0, ω_1, and λ_ω for any intermediate wire in the circuit. Using the λ_ω values for the output wires, the client can translate each of the garbled values returned by the evaluator p_e into a plaintext bit and recover the result of the requested computation. The client declares successful output verification only if the garbled output returned by the evaluator matches with either ω_0 or ω_1, for each output wire ω of the circuit.*

Collusion Resistance: Note that, without performing any computation, the evaluator can return one of the two actual garbled outputs for each output wire in the circuit, if and only if it colludes with all the n servers, $\{p_1, p_2, \ldots, p_n\}$, that participated in the creation of the garbled circuit, or factorizes N into its prime factors, p and q, which is infeasible.

Unpredictability: Further, the Blum, Blum, Shub pseudorandom number generator guarantees that one cannot predict the next/previous bit output from the generator, even with the knowledge of all the previous/future bits[10,11]. Thus, based on the observations of the garbled values during evaluation, the evaluator cannot predict the preceding or subsequent garbled values, or the λ values for any wire in the circuit.

3.5. Summary of our Proposed System

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



1. Problem: Find the *nearest* bank/ATM from the mobile client.
2. Goal: preserve the privacy of: (i) mobile client's input location, (ii) computed nearest bank/ATM location, (iii) computed distance.
3. Case study: An area of Salt Lake City, UT consisting of 104 blocks, bounded by Main St., 1300 East St., South Temple St., 800 South St.
4. Area consists of $L = 10$ Chase, Well Fargo bank/ATM locations.
5. Any East/South coordinate in this area represented using $l = max(\log_2(\lceil 1300 \rceil), \log_2(\lceil 800 \rceil)) = 11$ bits.

6. Distance between the mobile client at an intersection (x_a, y_a) and any bank/ATM location (x_b, y_b) corresponds to the *Manhattan distance* metric, since the streets are arranged in a grid pattern.
$$Distance = |x_b - x_a| + |y_b - y_a|$$
6. We create the Boolean circuit B for the computation using a combination of SUB, ADD, CMP, MUX, and a tree of MIN blocks.
7. Our circuit has N_X, N_A XOR, AND gates respectively, where
$N_X = [\{5(l+1) + 4l\} \times (L-1)] + [(15l+1) \times L] = 2596$,
$N_A = [\{2(l+1) + 2l\} \times (L-1)] + [4l \times L] = 854$.

Fig. 2: Privacy preserving search application that finds the nearest bank/ATM location from the mobile client.

4. Results

4.1. Circuit size of one garbled table entry

In this section, we analyze the size of the Boolean circuit (B') that computes one specific entry (A_{ab}) in the garbled table. Note that B' is different from B, which corresponds to the requested computation of the client (e.g., addition of two numbers). While creating the garbled circuit GC, the n parties use the 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.

Assume that each gate takes two input bits to produce one output bit. Recall from Section 3.3.1 that $A_{ab} = \gamma_{[((\lambda_x \oplus a) \otimes (\lambda_y \oplus b)) \oplus \lambda_z]} \oplus [G_b(\alpha_{a1}) \oplus G_b(\alpha_{a2}) \oplus \ldots \oplus G_b(\alpha_{an})] \oplus [G_a(\beta_{b1}) \oplus G_a(\beta_{b2}) \oplus \ldots \oplus G_a(\beta_{bn})]$, where $\otimes \in \{XOR, AND\}$ denotes the binary operation of gate A.

Let $s = ((\lambda_x \oplus a) \otimes (\lambda_y \oplus b)) \oplus \lambda_z$. Computing s requires a total of $3 + 3(n-1) = 3n$ XOR gates and 1 $\otimes$ gate, 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, $(1 \le i \le n)$.

Boolean circuit B' includes a *multiplexer* that chooses γ_s using the expression, $\gamma_s = ((\gamma_0 \oplus \gamma_1).s) \oplus \gamma_0$, which is composed of 2 XOR gates and 1 AND gate. Since $|\gamma_0| = |\gamma_1| = nk + 1$, and $LSB(\gamma_s) = s$, multiplexing is performed on the most significant nk bits. We can build this multiplexer using a total of $2nk$ XOR gates and nk AND gates.

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

To summarize, 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. For example, when $n = 6$ and $k = 128$, the circuit that computes A_{ab} has a total of 10782 XOR and 769 AND gates. *While the number of XOR gates increases quadratically with n, the number of AND gates increases only linearly with n.*

4.2. Server-side Cost

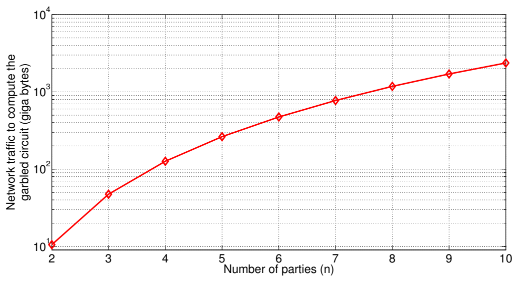
A 1-out-of-4 OT exchange between two parties involves the exchange of a small number of messages. Let $s_{1:4}$ denote the total number of bits that are exchanged during a 1-out-of-4 OT. Then, $s_{1:4} = 8(|p| + k)$[12,13]. Note that $|p|$ and k are public and symmetric key security parameters, respectively. For example, $|p| = 3072$ achieves the equivalent of $k = 128$-bit security[14]; in this case, $s_{1:4} = 3200$ bytes.

For each AND gate in the circuit B', all possible pairs of the n servers, $(p_i, p_j), 1 \le i < j \le n$, engage in a 1-out-of-4 OT. Since the number of AND gates in the circuit B' is at most $(nk+1)$, the total number of 1-out-of-4 OTs is $t_{1:4} = (nk+1) \times n(n-1)/2$.

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

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

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



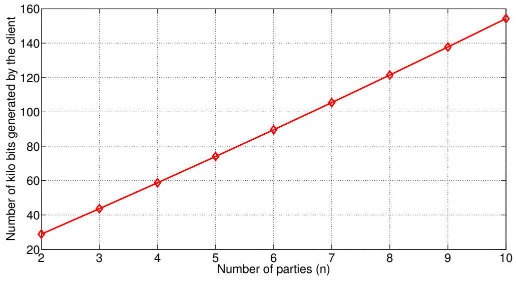


Fig. 3: Server-side and client-cost for the privacy preserving search application that finds the nearest bank/ATM from the mobile client.

Privacy preserving search for nearest bank/ATM: To evaluate our secure cloud computing approach for mobile systems, we consider the privacy preserving search application of finding the nearest bank/ATM in Salt Lake City, UT, which we have shown in Fig. 2. Fig. 3 shows the amount of network traffic to create the garbled circuit for this application. For example, when $n = 4$, the cloud servers exchange about 126 GB of information to construct the garbled circuit, which demonstrates feasibility of our approach for real-world privacy preserving computations.

4.3. Client-side Cost

To enable the creation of 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 $(p_i, p_k), (1 \leq i, k \leq n)$.

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

The client generates garbled input of length $(nk + 1)$ bits for each plaintext input bit to the circuit. Since the λ value of a wire, in turn, depends on the 1-bit shares for the n parties, the number of bits that the client needs to generate for each input wire equals $b_i = (nk + n)$.

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

To summarize, 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, where W_i and W_o denote the number of input and output wires, respectively, in the Boolean circuit B. In our privacy preserving search application, $W_i = 2l = 22$ and $W_o = 2l + l + 1 = 34$. Fig. 3 shows the total number of kilo bits that the client generates for the privacy preserving search for the nearest bank/ATM from the mobile client.

Comparing the client-side and network-side costs from Fig. 3, we note that while the servers generate and exchange Gigabytes of information to create the garbled circuit, the mobile client, on the other hand, generates and exchanges only kilobytes of information with the evaluator and the other servers. *For example, with $n = 4$ servers, the mobile client generates and exchanges less than 60 kilo bits of information to preserve its location privacy. Furthermore, the client-side cost grows much slowly with the number of servers, in comparison to the server-side cost.*

4.4. Comparison of Our Scheme with Gentry's FHE Scheme

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

4.5. Time Taken for Construction and Evaluation of Garbled Circuit

We implemented our secure cloud computing system using BIGNUM routines and crypto functions from the OpenSSL library. We built our system as a collection of modules, and the servers communicate using TCP sockets.

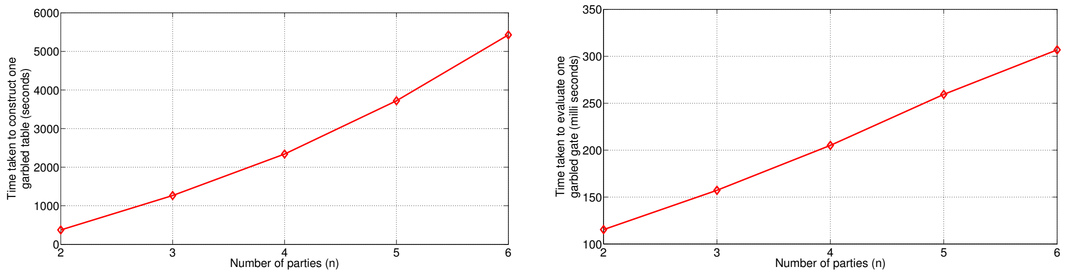


Fig. 4: (a) Time taken for constructing one garbled gate; (b) Time taken for evaluating one garbled gate.

We evaluated our system on a server with Intel Xeon 2.53 GHz processor, with 6 cores and 32 GB RAM. Fig. 4 shows the time taken to construct one garbled table as a function of n. *We note that the garbled tables for any number of gates in the circuit can be constructed in parallel, which will significantly reduce the construction time.*

Fig. 4 also shows the time taken to evaluate one garbled gate as a function of n. We observe that *evaluation is significantly faster than construction, where the latter can be done offline.* If the garbled circuits are *pre-computed*, and made available to the evaluator, in advance, it can readily carry out the requested computation, and therefore, drastically reduce the *response time* for the mobile client.

5. Related Work

Homomorphic encryption is an approach that enables performing computations directly on the encrypted data, without requiring private decryption keys. For example, in the RSA public key system, the product of two ciphertext messages produces a ciphertext corresponding to the product of the underlying plain text messages[2]. Domingo-Ferrer et al.[15] present a homomorphic scheme that represents ciphertext as polynomials, allowing addition and multiplication operations on the underlying plain text. Recently, proposed fully homomorphic encryption (FHE) schemes (e.g., Gentry et al.[3]) can perform any arbitrary computation on encrypted data. However, FHE schemes are currently impractical for cloud computing applications due to extremely large cipher text size (Section 4.3).

Yao's garbled circuits have been used in existing work in conjunction with oblivious transfer protocols for secure two-party computation[4,5,16]. Lindell et al.[17] present an excellent survey of secure multiparty computation, along with numerous potential applications, such as privacy preserving data mining, private set intersection, electronic voting and electronic auction. A number of secure two-party and multiparty computation systems have been built over the years (e.g.,[18,19,20]). Note that in secure multiparty computation systems multiple parties hold private inputs and receive the result of the computation; however, in a secure cloud computing system, such as ours, 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. In our work, we adapt secure multiparty computation protocols[6,7,8,9], for building a secure and verifiable cloud computing for mobile systems.

Twin clouds[21] is a secure cloud computing architecture, in which the client uses a *private* cloud for creating garbled circuits and a *public* commodity cloud for evaluating them. Our solution, on the other hand, employs multiple public cloud servers for creating as well as evaluating the garbled circuits. *In other words, our solution obviates the requirement of private cloud servers.*

While FHE schemes currently remain impractical, they, however, offer interesting constructions, such as reusable garbled circuits[22] and verifiable computing capabilities that permit a client to verify whether an untrusted server has actually performed the requested computation[23]. *In our proposed system, we enable the client to efficiently verify whether an untrusted server has actually evaluated the garbled circuit, without relying on any FHE scheme.*

Carter et al.[24] have proposed an atypical secure two party computation system with three participants: Alice, Bob and a Proxy. In their work, Bob is a webserver that creates garbled circuits, and Alice is a mobile device that delegates the task of evaluating the garbled circuits to the Proxy, which is a cloud server. We note that the computation and adversary models in Carter et al.'s work are very different from that of our work. First, in their work, being a secure two party computation system, *both Alice and Bob* provide private inputs for the computation that they wish to perform jointly; however, in our secure cloud computing model, *only one party, i.e., the mobile client*, provides inputs and obtains result of the computation in garbled form. Second, Cartel et al.'s scheme requires that neither Alice nor Bob can collude with the Proxy; in a sharp contrast, our method preserves the privacy of the client data even if the evaluating server colludes with all but one of the cloud servers that participated in the creation of the garbled circuit.

6. Concluding Remarks

We proposed a novel secure and verifiable cloud computing for mobile system using multiple servers. Our method combines the secure multiparty computation protocol of Goldreich et al. and the garbled circuit design of Beaver et al. with the cryptographically secure pseudorandom number generation method of Blum et al. Our method preserves the privacy of the mobile client's inputs and the results of the computation, even if the evaluator colludes with all but one of the servers that participated in the creation of the garbled circuit. Further, our method can efficiently detect a cheating evaluator that returns arbitrary values as output without performing any computation. We presented an analysis of the server-side and client-side complexity of our system. Using real-world data, we evaluated our system for a privacy preserving search application that locates the nearest bank/ATM from the mobile client. We evaluated the time taken to construct and evaluate a garbled circuit for varying number of servers, and demonstrated the feasibility of our proposed approach.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., et al. A view of cloud computing. *Commun ACM* 2010; **53**(4):50–58.
2. Rivest, R.L., Adleman, L., Dertouzos, M.L.. On data banks and privacy homomorphisms. *Foundations of secure computation* 1978; **32**(4):169–178.
3. Gentry, C.. Computing arbitrary functions of encrypted data. *Commun ACM* 2010;**53**(3):97–105.
4. Yao, A.C.. Protocols for secure computations. In: *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*; SFCS '82. Washington, DC, USA: IEEE Computer Society; 1982, p. 160–164.
5. Yao, A.C.C.. How to generate and exchange secrets. In: *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*; SFCS '86. Washington, DC, USA: IEEE Computer Society; 1986, p. 162–167.
6. Goldreich, O.. *Foundations of Cryptography: Volume 2, Basic Applications*. New York, NY, USA: Cambridge University Press; 2004.
7. Goldreich, O., Micali, S., Wigderson, A.. How to play any mental game. In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*; STOC '87. New York, NY, USA: ACM; 1987, p. 218–229.
8. 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*; STOC '90. New York, NY, USA: ACM. ISBN 0-89791-361-2; 1990, p. 503–513.
9. Rogaway, P.. *The round complexity of secure protocols*. Ph.D. thesis; Massachusetts Institute of Technology; 1991.
10. Blum, L., Blum, M., Shub, M.. A simple unpredictable pseudo random number generator. *SIAM J Comput* 1986;**15**(2):364–383.
11. Schneier, B.. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. New York, NY, USA: John Wiley & Sons, Inc.; 1995. ISBN 0-471-11709-9.
12. Naor, M., Pinkas, B.. Efficient oblivious transfer protocols. In: *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*; SODA '01. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics. ISBN 0-89871-490-7; 2001, p. 448–457.
13. Naor, M., Pinkas, B.. Computationally secure oblivious transfer. *Journal of Cryptology* 2005;**18**(1):1–35.
14. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.. Recommendation for key management - part 1: General (revision 3). *NIST Special Publication 800-57* 2012;URL: `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf`.
15. Domingo-Ferrer, J.. A provably secure additive and multiplicative privacy homomorphism. In: *Proceedings of the 5th International Conference on Information Security*; ISC '02. London, UK, UK: Springer-Verlag; 2002, p. 471–483.
16. Lindell, Y., Pinkas, B.. A proof of security of yao's protocol for two-party computation. *J Cryptol* 2009;**22**(2):161–188.
17. Lindell, Y., Pinkas, B.. Secure multiparty computation for privacy-preserving data mining. *J Privacy and Confidentiality* 2009;**1**(1).
18. Henecka, W., K ögl, S., Sadeghi, A.R., Schneider, T., Wehrenberg, I.. Tasty: Tool for automating secure two-party computations. In: *Proc. of the 17th ACM Conference on Computer and Communications Security*; CCS '10. New York, NY, USA: ACM; 2010, p. 451–462.
19. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.. Fairplay – a secure two-party computation system. In: *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*; SSYM'04. Berkeley, CA, USA: USENIX Association; 2004, p. 20–20.
20. 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*; CCS '08. New York, NY, USA: ACM; 2008, p. 257–266.
21. Bugiel, S., Nürnberger, S., Sadeghi, A.R., Schneider, T.. Twin clouds: Secure cloud computing with low latency. In: *Proc. of the 12th IFIP TC 6/TC 11 Intl. Conference on Communications and Multimedia Security*; CMS'11. Berlin, Heidelberg: Springer-Verlag; 2011, p. 32–44.
22. Goldwasser, S., Kalai, Y., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.. Reusable garbled circuits and succinct functional encryption. In: *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*; STOC '13. New York, NY, USA: ACM; 2013, p. 555–564.
23. Gennaro, R., Gentry, C., Parno, B.. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: *Proceedings of the 30th Annual Conference on Advances in Cryptology*; CRYPTO'10. Berlin, Heidelberg: Springer-Verlag; 2010, p. 465–482.
24. 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*; SEC'13. Berkeley, CA, USA: USENIX Association; 2013, p. 289–304.