Efficient Non-Interactive Verifiable Outsourced Computation for Arbitrary Functions

Chunming Tang [1,2], Yuenai Chen [1]

[1] School of Mathematics and Information Sciences, Guangzhou University,
Guangzhou 510006, China. *E-mail:ctang@gzhu.edu.cn*
[2] Key Laboratory of Mathematics and Interdisciplinary Sciences of Guangdong
Higher Education Institutes, Guangzhou University, 510006, China.
E-mail:chenyuenai@163.com

Abstract. Non-interactive verifiable outsourced computation enables a computationally weak client to outsource the computation of a function f on input x to a more powerful but untrusted server, who will return the result of the function evaluation as well as a proof that the computation is performed correctly. A basic requirement of a verifiable outsourced computation scheme is that the client should invest less time in preparing the inputs and verifying the proof than computing the function by himself.

One of the best solutions of such non-interactive schemes are based on Yao's garble circuit and full homomorphic encryption, which leads to invest $poly(T)$ running time in offline stage and $poly(logT)$ time in online stage of the client, where T is the time complexity to compute f.

In this paper, we'll present a scheme which does not need to use garble circuit, but to use a very simple technique to confuse the function we are going to compute, and only invests $poly(logT)$ running time in the offline stage.

Keywords: cloud computing, non-interactive outsourced computation, verifiable outsourced computation, full homomorphic encryption

1 Introduction

Outsourced computation, in which a weak client, who has very limited computational resources, wishes to "outsource" computing to a powerful server, who has a great quantity of resources and infinite capacities of computation, becomes a very active research area in cryptograph recently. Maybe the following two reasons can explain why it becomes so heat. One is the prevalence of *cloud computing*, a paradigm where businesses buy computing time from a service, rather than purchase and maintain their own computing resources. The other is the proliferation of computationally weak mobile devices, such as cell-phones, sensors, tablets, and security access-cards, which might need to buy computing time of a costly computations from cloud, such as a cryptographic operation or a photo manipulation, which they are not able to perform by themselves.

The central problem of outsourced computation is: *1) if the server is malicious that he may return an incorrect result of the computation due to the financial purpose; Or 2) if the information about the computation is so sensitive that you must guarantee both of the privacy of the inputs and outputs of the computation.* Thus we must concern about the following very important two aspects associated with the security of outsourced computation:

- *The privacies of input and output of the computation.* In other words, the computation is done over encrypted data.
- *The correctness of the computation.* This implies the server has to prove the correctness of the computation, and the client should be able to *efficiently verify* this correctness, where 'efficiently' means the client must invest less time in the verification of the result than computation from scratch by himself.

Therefore, we are concern about a *verifiable outsourced computation*, which enables a computationally weak client to outsource the computation of a function f on input x to a more powerful but untrusted server, who returns the result of the function evaluation as well as a proof that the computation is performed correctly. Such a verifiable outsourced computation requires that 1) the privacy of the client's input and output; 2) the correctness of the computation; 3) the security of the computation (i.e., a malicious server cannot persuade the client to accept an incorrect answer); 4) the efficiency of the computation (i.e., the client should invest less time in preparing its input and verifying the server's proof than computing f on its own). Another reasonable requirement is that the running time of the worker carrying out the proof should also be reasonable. For example, when the computing of function f takes time T and the length of the inputs and outputs is n, then we would like the client invest $poly(n, logT)$ time and the server invest $poly(T)$ time.

1.1 Previous Work

Computing over encrypted data is an attractive research problem and has a long research history. The first breakthrough construction of *full homomorphic encryption* (FHE) proposed by Gentry [Gen09] improved the development of computation over encrypted data. After that, many other more efficient FHE schemes have been constructed based on it [Gen10, GH11, BGV12, Bra12, GHS12].

The problem of verifying the correctness of arbitrarily computation, that is, verifiable computation, also gains much attentions. Actually, it is the goal of *interactive proofs* [Bab85,GMR89], where a powerful prover can convince a weak verifier of the truth of statements that the verifier could not compute on its own. In order to make the verification executed efficiently by the client, *probabilistically checkable proofs* (PCPs) [BFL90,BFLS91,AS92] were proposed, where a prover can prepare a proof that the verifier needs to check in only very few place. Based on PCP proof, several protocols came up: Kilian's efficient arguments [Kil92,Kil95] with interactive verification, Micali's non-interactive CS proofs for any polynomial-time computation in random oracle model [Mic94], Goldwasser et al.'s interactive proof only for small-depth computation, their proof can be converted into non-interactive [GKR08].

In this paper, we only restrict our attention to non-interactive solutions for arbitrary computation. That is, the server/prover should be able to send a proof to the client/verifier in the same message that it sends the result of the computation. Micali's non-interactive CS proofs [Mic94] invests $poly(T, k)$ time by prover and $poly(n, k, logT)$ time by verifier in random oracle model (where k is the security parameter and n is the length of input). However, the random oracle heuristic is known to be unsound in general [CGH04]. Goldwasser et al.'s non-interactive proof [GKR08] applies only to small-depth computation, since the verifier complexity grows linearly with the depth.

Considering the limitations of above two non-interactive solutions, Gennaro, Gentry, and Parno [GGP10] formalized the notion of verifiable computation, and showed how to outsource arbitrary computation by increasing the verifier's offline complexity and involving large public key, based on Yao's garbled circuit construction [Yao82,Yao86] and FHE scheme [Gen09]. In their protocol, in order to construct a garbled circuit of function f (which is computable in time T) in the offline stage, the client invests $poly(T, k)$ running time to generate a public key of size $poly(T, k)$ and a secret key of size $poly(k)$. In the online stage, the client/verifier's complexity is reduced to $poly(n, k, logT)$, and the server/prover's complexity is $poly(T, k)$. The online stage of the protocol can be executed many times to outsource the computation of f on many inputs so as to amortized the client's large investment in the offline stage.

Chung et al.[CKV10] proposed a protocol of non-interactive verifiable outsourced computation based on FHE without using garble circuit, which eliminated the large public key from Gennaro et al.'s scheme. However, it still needed to cost $poly(T, k)$ time for the client to compute the function f on several random inputs in the offline stage. In order to decrease the large investment of client in the offline stage, they presented another construction of the protocol at the price of a 4-message (offline) interaction with a server, which we are not interested in here.

The two mentioned protocols above are in a very weak model in which the adversary is *not* allowed to issue *verification queries* to the client. In other words, the acceptance/rejection bit of the verification performed by the client remains private and not learn by the adversary. Gennaro and Pastro [GP14] present a non-interactive verifiable outsourced computation protocol in the presence of verification queries based on the FHE and the existing verifiable outsourced computation (which is not allowed the verification queries) for arbitrary poly-time computation, which means, if the existing verifiable outsourced computation needs $poly(T, k)$ offline complexity, then their protocol needs $poly(T, k)$ offline complexity, too. As we have mentioned, there is no such protocols that invest $poly(logT, k)$ time in the offline stage even in the weak model.

1.2 Our contribution

In this work, we are focused on non-interactive verifiable outsourced computation for arbitrary polynomial-time computation. As the non-interactive protocols given by Gennaro et al.[GGP10] and Chung et al.[CKV10] have large investment in the offline stage, both of which are $poly(T,k)$. Considering this, we will provide a protocol which has more efficient non-interactive offline complexity than both of the above.

- Our protocol is based on full homomorphic encryption scheme and involves no garble circuit while using a technique to confuse the function we are going to compute. In the offline stage of our protocol, it needs neither to generate a large public key as Gennaro et al.'s scheme, nor to compute the function on a random input by the client itself as Chung et al.'s scheme. More importantly, the client's offline complexity can be reduced from $poly(T,k)$ to $poly(k,logT)$ (see table 1), and the online stage also can be executed efficiently by the client.

Since our protocol is based on a FHE scheme, any improvements in future FHE scheme will be directly gained by our schemes.

2 Preliminaries on FHE

A full homomorphic encryption scheme FHE=(KeyGen, Enc, Dec, Eval) consists of four PPT algorithms defined as follows.

- $\texttt{KeyGen}(k) \to (pk, sk)$. The key generation algorithm KeyGen takes as input a secure parameter k, and outputs a public encryption key pk and a secret decryption key sk.
- $\texttt{Enc}(pk, x) \to c_x$. The algorithm Enc encrypts an input x under public key pk, outputs the ciphertext c_x.
- $\texttt{Eval}(pk, C, c_x) \to c_y$. the evaluation algorithm Eval takes in a public key pk, a circuit C and a ciphertext c_x, and outputs a new ciphertext c_y that decrypts to the results $y = C(x)$.
- $\texttt{Dec}(sk, c_y) \to y$. The algorithm Dec decrypts the ciphertext c_y to a plaintext y under the secret decryption key sk.

We require that an FHE scheme satisfies four properties, which is encryption correctness (i.e., it always holds that Dec(sk,Enc(pk,x))=x), evaluation correctness (means that Dec(sk, Eval(pk, c_x, C))=$C(x)$), succinctness, which requires that the size of the ciphertext always depends polynomially on the secure parameter, but is independent of the size of the evaluated circuit C, and semantic security.

Gentry's construction [Gen09] satisfies the encryption correctness and evaluation correctness, furthermore, the Eval algorithm of his scheme can be made deterministic.

3 The Model

Informally, a verifiable outsourced computation scheme is a two-party protocol where the client runs a preprocessing phase (offline stage) to outsource the function f to the server. Later, in the online stage, the client encrypts the input x and asks the server to evaluate the function on it. Then the server responds an encrypted value of $f(x)$ as well as a proof of its correctness which will be verified by the client. We use the definition and secure model of [GGP10].

A verifiable outsourced computation scheme $\mathcal{VOC}$=$(\texttt{KeyGen},\texttt{ProbGen},\texttt{Compute},\texttt{Verify})$ consists of the four algorithms defined as follow.

(1) $\texttt{KeyGen}(f,k) \rightarrow (PK, SK)$. The randomized key generation algorithm takes the security parameter k as inputs and generates a public key that encode the function f, which is used by the server to compute f, and a secret key which is kept private by the client.

(2) $\texttt{ProbGen}_{SK}(x) \rightarrow (\sigma_x,\tau_x)$. The problem generation algorithm uses the secret key SK to encode the function input x as a public value σ_x which is given to the server to compute with, and a secret value τ_x which is kept private by the client.

(3) $\texttt{Compute}_{PK}(\sigma_x) \rightarrow \sigma_y$. Using the client's public key and the encoded input, the server computes an encoded version of the function's output $y = f(x)$.

(4) $\texttt{Verify}_{SK}(\tau_x, \sigma_y) \rightarrow y \cup \perp$. Using the secret key SK and the secret decoding τ_x, the verification algorithm converts the server's encoded output into the output of the function $y = f(x)$ or outputs $\perp$ indicating that σ_y does not represent the valid output of f on x. If $acc = 1$, then we say the client accepts $y = f(x)$; otherwise, we say the client rejects.

Now, we'll recall the four properties for a verifiable outsourced computation defined in [GGP10], which are correctness, security, full privacy (both input and output privacy) and efficiency.

(1) *Correctness.*

A verifiable outsourced computation scheme is correct if the problem generation algorithm produces values that allow an honest server to evaluate the values that will verify successfully and correspond to the evaluation of f on those inputs. More formally:

Definition 1 (Correctness). A verifiable outsourced computation scheme $\mathcal{VOC}$ is correct if for any function f, the key generation algorithm produces keys $(PK, SK) \leftarrow \texttt{KeyGen}(f, k)$ such that, for all x in the domain of f, if $(\sigma_x, \tau_x) \leftarrow \texttt{ProbGen}_{SK}(x)$ and $\sigma_y \leftarrow \texttt{Compute}_{PK}(\sigma_x)$, then $(1, y = f(x)) \leftarrow \texttt{Verify}_{SK}(\tau_x, \sigma_y)$.

(2)*Security*

Intuitively, a verifiable outsourced computation scheme is secure if a malicious server cannot persuade the verification algorithm to accept an incorrect output. In other words, for a given function f and input x, a malicious server should not be able to convince the verification algorithm to output $\hat{y}$ such that

$\hat{y} \neq f(x)$. The formal security is defined in experiment-based model.

```
Experiment 1: $\mathbf{Exp}_A^{Verif}[\mathcal{VOC}, f, k]$
    $(PK, SK) \leftarrow \mathtt{KeyGen}(f,k)$;
    For $i = 1, ..., l = poly(k)$;
        $x_i \leftarrow A(PK, x_1, \sigma_1, ..., x_{i-1}, \sigma_{i-1})$;
        $(\sigma_i, \tau_i) \leftarrow \mathtt{ProbGen}_{SK}(x_i)$
    $(i, \hat{\sigma}_y) \leftarrow A(PK, x_1, \sigma_1, ..., x_l, \sigma_l)$;
    $\hat{y} \leftarrow \mathtt{Verify}_{SK}(\tau_i, \hat{\sigma}_y)$
    If $\hat{y} \neq \perp$ and $\hat{y} \neq f(x_i)$, output 1, else 0.
```

Essentially, the adversary is given oracle access to generate the encoding of multiple problem instances. The adversary succeeds if it produces an output that convinces the verification algorithm to accept on the wrong output value for a given input value.

Definition 2 (Security). For a verifiable outsourced computation scheme $\mathcal{VOC}$, we define the advantage of an adversary A in the experiment above as:

$$\mathbf{Adv}_A^{Verif}(\mathcal{VOC}, f, k) = Pr[\mathbf{Exp}_A^{Verif}[\mathcal{VOC}, f, k] = 1] \tag{1}$$

A verifiable outsourced computation scheme $\mathcal{VOC}$ is secure for a function f, if for any adversary A running in probabilistic polynomial time,

$$\mathbf{Adv}_A^{Verif}(\mathcal{VOC}, f, k) \leq negl(k) \tag{2}$$

where $negl(\cdot)$ is a negligible function of its input.

(3)*Full Privacy*

Full privacy means both the input and output privacy. Input privacy is defined based on a typical indistinguishability argument guaranteeing that no leakage of the information about the inputs. The definition of output privacy can be made similarly.

Intuitively, a verifiable outsourced computation scheme is private when the public outputs of the problem generation algorithm `ProbGen` over two different inputs are indistinguishable; that is, an adversary cannot decide which encoding is the correct one for a given input. More formally, consider the following experiment 2: the adversary is given the public key for the scheme and selects two inputs x_0,x_1. He is then given the encoding of a randomly selected one of the two inputs and must guess which one was encoded. During this process the adversary is allowed to request the encoding of any input he desires. The experiment is described below. The oracle $\mathtt{PubProbGen}_{SK}(x)$ calls $\mathtt{ProbGen}_{SK}(x)$ to obtain (σ_x, τ_x) and returns only the public part σ_x.

```
Experiment 2: $\mathbf{Exp}_A^{Priv}[\mathcal{VOC}, f, k]$
    $(PK, SK) \leftarrow \mathtt{KeyGen}(f, k)$;
    $(x_0, x_1) \leftarrow A^{\mathtt{PubProbGen}_{SK}(\cdot)}(PK)$;
```

```
$(\sigma_0, \tau_0) \leftarrow \texttt{ProbGen}_{SK}(x_0)$;
$(\sigma_1, \tau_1) \leftarrow \texttt{ProbGen}_{SK}(x_1)$;
$b \leftarrow \{0, 1\}$ ;
$\hat{b} \leftarrow A^{\texttt{PubProbGen}_{SK}(\cdot)}(PK, x_0, x_1, \sigma_b)$;
If $\hat{b} = b$, output 1, else 0;
```

Definition 3 (Privacy). For a verifiable outsourced computation scheme $\mathcal{VOC}$, we define the advantage of an adversary A in the experiment above as:

$$\mathbf{Adv}_A^{Priv}(\mathcal{VOC}, f, k) = |Pr[\mathbf{Exp}_A^{Priv}[\mathcal{VOC}, f, k] = 1] - \frac{1}{2}| \tag{3}$$

A verifiable outsourced computation scheme $\mathcal{VOC}$ is private for a function f, if for any adversary A running in probabilistic polynomial time,

$$\mathbf{Adv}_A^{Priv}(\mathcal{VOC}, f, k) \leq negl(k) \tag{4}$$

where $negl(\cdot)$ is a negligible function of its input.

(4)*Efficiency*

The final condition we require from a verifiable outsourced computation scheme is that the time to encode the input and verify the output must be less than the time to compute the function from scratch.

Definition 4 (Efficiency). A $\mathcal{VOC}$ is efficient if for any x and any σ_y, the time required for $\texttt{ProbGen}_{SK}(x)$ plus the time required for $\texttt{Verify}(\sigma_y)$ is $o(T)$, where T is the fastest known time required to compute $f(x)$.

4 Efficient Non-interactive Verifiable Outsourced Computation Scheme

As we have mentioned, the well-known two protocols given by [GGP10] and [CKV10] have very expensive offline complexity. In [GGP10], client needs to create a garbled circuit for the evaluation function f in the offline stage, which costs the client $poly(T)$ time to finish it and generates a large public key of size $poly(T)$. This large public key is eliminated by Chung et al.[CKV10] in their protocols, but the client needs to evaluate the function f on randomly selected input r by herself, which still invests $poly(T)$ time to compute $f(r)$ in the offline stage. In this section, we propose two protocols in which we both eliminate the large public key and reduce the offline complexity to $poly(logT)$.

4.1 Protocol $\mathcal{VOC}$: Reusable Verifiable Outsourced Computation Scheme

Suppose we compute the function f in domain $\{0, 1\}^n$. Informally, our first protocol works as follow: in the preprocessing phase, the client will confuse the function f with two variables, that is, sets $F(z) \triangleq F(x, a, b) = af(x) + b$. Then represent the function F as a circuit C such that $C(z) = F(z)$, and this circuit C will be evaluated by the server instead of the function f. In the online stage, given the input x of function f, the client will first run a key generation algorithm `Fkeygen` of a full homomorphic encryption and obtain public/secret key pair. Then the client randomly chooses two values a_1, a_2 and b_1, b_2 for the variables a and b respectively and sends the ciphertexts of $z_1 = (x, a_1, b_1)$ and $z_2 = (x, a_2, b_2)$ under the public key to the server, who will return two encrypted values of $F(z_1)$ and $F(z_2)$. Finally, the client decrypts them using the sectet key of `Fkeygen` and verifies the correctness of the computation, i.e., if $(F(z_1) - b_1)a_1^{-1} = (F(z_2) - b_2)a_2^{-1}$, then the client accept and the computation result of function f on input x is $(F(z_1) - b_1)a_1^{-1}$ or $(F(z_2) - b_2)a_2^{-1}$. We give a detailed description below.

Protocol $\mathcal{VOC}$.
Let k be security parameter, `FHE=(Fkeygen,Fenc,Fdec,Feval)` be a semantically secure full homomorphic encryption scheme. Suppose the computation function f is in domain $\{0,1\}^n$.

1. `KeyGen`$(f, k) \to (PK, SK)$. The algorithm `KeyGen` first encodes the function f with n-bit input x into a function F with $3n$-bit input $a\|b\|x$, such that $F(a\|b\|x) = af(x) + b$, where a, b are the variables in domain $\{0,1\}^n$. Then it represents F as a circuit C such that $C(z) = F(z)$ for any $z \in \{0,1\}^{3n}$. The circuit C will be evaluated by the server instead of f. The algorithm then sets the public key $PK = C$ and the secret key SK is the position of the bites of strings a, b, x. This preprocessing phase will be executed in the offline stage by the client.
2. `ProbGen`$_{SK}(x) \to (\sigma_x, \tau_x)$. The problem generation algorithm takes as the input x and output a public value σ_x, which will be used to evaluate the value of function F by the server in the next phase, as well as a private value τ_x, which will be used by the client to verify the correctness of the computation. This algorithm will be executed in the online stage by the client as below.
   – It first calls the key generation algorithm `Fkeygen` of FHE to create a key pair: $(pk, sk) \leftarrow \texttt{Fkeygen}(k)$.
   – Chooses randomly two pairs (a_1, b_1) and (a_2, b_2) for the variables (a, b), where $a_i, b_i \in \{0,1\}^n, i \in \{1,2\}$.
   – Runs the encryption algorithm `Fenc` of FHE to encrypt the tuples (a_1, b_1, x) and (a_2, b_2, x) in a random order:
   $$(\hat{a}_1, \hat{b}_1, \hat{x}_1) \leftarrow \texttt{Fenc}_{pk}(a_1, b_1, x)$$
   $$(\hat{a}_2, \hat{b}_2, \hat{x}_2) \leftarrow \texttt{Fenc}_{pk}(a_2, b_2, x)$$
   – Sets $\sigma_x^1 = (\hat{a}_1, \hat{b}_1, \hat{x}_1)$, $\sigma_x^2 = (\hat{a}_2, \hat{b}_2, \hat{x}_2)$, $\tau_x^1 = (a_1, b_1)$, $\tau_x^2 = (a_2, b_2)$, then the public value $\sigma_x = (pk, \sigma_x^1, \sigma_x^2)$, and the private value $\tau_x = (sk, \tau_x^1, \tau_x^2)$
3. `Compute`$_{PK}(\sigma_x) \to \sigma_y$. The computation algorithm `Compute` takes as the input σ_x and outputs the evaluation of the circuit C on this input. This will be done by the server in the online stage as below. It firstly calls the

evaluation algorithm `Feval` of the FHE and computes $\texttt{Feval}_{pk}(\sigma_x^1, C) \rightarrow \sigma_y^1$, $\texttt{Feval}_{pk}(\sigma_x^2, C) \rightarrow \sigma_y^2$. Then the output of algorithm `Compute` is set as $\sigma_y = (\sigma_y^1, \sigma_y^2)$, where σ_y^1, σ_y^2 are the ciphertexts of $F(x, a_1, b_1), F(x, a_2, b_2)$ respectively according to the properties of the FHE.

4. $\texttt{Verify}_{SK}(\tau_x, \sigma_y) \rightarrow y \cup \perp$. After receiving the result of the computation, the client firstly decrypt the two ciphertexts σ_y^1 and σ_y^2 of σ_y into y_1 and y_2 using the decryption algorithm `Fdec` of the FHE uder the secret key sk; Then uses (a_1, b_1) and (a_2, b_2) to verify if $(y_1 - b_1)a_1^{-1} = (y_2 - b_2)a_2^{-1}$. If the equation establishes, set the consistent value as the output of $y = f(x)$, otherwise, output $\perp$.

Theorem 1. Let `FHE` be a semantically secure full homomorphic encryption scheme. Then protocol $\mathcal{VOC}$ is a correct, secure, private and efficient non-interactive verifiable outsourced computation scheme.

Proof. In order to prove Theorem 1, it is easy to see that the protocol $\mathcal{VOC}$ is correct and efficient, because the time required for $\texttt{ProbGen}_{SK}(x)$ is $poly(logT)$, and the verification algorithm `Verify` invests also $poly(logT)$ time, thus, the addition of them is less than T, the time to compute f from scratch. It is also not difficult to show that the protocol $\mathcal{VOC}$ is private, because the encryption of the input is probabilistic and the output of the computation is an encrypted version.

Now, we are focused on the proof of security of the protocol. The proof needs two high-level steps. We'll first show that the protocol $\mathcal{VOC}$ has one-time security (Lemma 1), i.e., the protocol can be used to compute f securely on one input. Then we reduce the security of protocol $\mathcal{VOC}$ (with multiple executions) to the security of one time execution in which we expect the adversary to cheat.

Definition 5 (One-time Security Error). Let $\mathcal{VOC}$ be a verifiable outsourced computation scheme and k be a security parameter. The one-time security experiment $\mathbf{Exp}_{A^*}^{Verif}[\mathcal{VOC}, f, k]$ for $\mathcal{VOC}$ is the same as experiment 1 excepts that it only runs one round in the online stage. We say that $\mathcal{VOC}$ has **one-time security error** ε if for every PPT adversary A^*, $Adv_{A^*}^{Verif}(\mathcal{VOC}, f, k) \leq \varepsilon(k)$.

Lemma 1. Assume that the full homomorphic encryption scheme is semantically secure, then the verifiable outsourced computation scheme $\mathcal{VOC}$ has one-time security error $\frac{1}{2^{3n}} + negl(k)$.

Proof of lemma 1. Fix any security parameter k, and any efficient (cheating) adversary A^*. According to experiment $\mathbf{Exp}_{A^*}^{Verif}[\mathcal{VOC}, f, k]$, the adversary A^* is allowed to query the oracle $\texttt{ProbGen}_{SK}(\cdot)$ only once (i.e., $l = 1$) and must cheat on that input. From the experiment $\mathbf{Exp}_{A^*}^{Verif}[\mathcal{VOC}, f, k]$, A^* firstly get the circuit C evaluating the function $F(z) = af(x) + b$ which is the encoding of the function f.

Then A^* queries the oracle `ProbGen` only once on input x and obtains a public key $\sigma_x = (pk, \sigma_x^1, \sigma_x^2)$, where σ_x^1 and σ_x^2 are the two ciphertexts of x assiciated with two randomly chosen pairs (a_1, b_1) and (a_2, b_2) in $\{0, 1\}^n$, which are kept private by the client, respectively. Then, A^* replies with answer $\sigma_{y'} = (\sigma^1_{y'}, \sigma^2_{y'})$ to the client which will be decrypted to y'_1 and y'_2 using the decryption algorithm `Fdec` of the FHE scheme. At the end of the experiment, if A^* succeeded in cheating, the decryption y'_1 and y'_2 should be decoded into the same value, i.e., $(y'_1 - b_1)a_1^{-1} = (y'_2 - b_2)a_2^{-1} = y'$, by which we mean a value other than $f(x)$.

Instead of computing the function F honestly, the adversary A^* firstly picks y'_1 randomly in domain $\{0,1\}^n$. For any choice y'_1, there is only one corresponding $f(x')$ such that $y'_1 = a_1 f(x') + b_1$. Thus A^* can successfully guess $f(x')$ with probability $\frac{1}{2^n}$. Then A^* constructs y'_2 such that $y'_2 = a_2 f(x') + b_2$ with probability $\frac{1}{2^{3n}}$ since the probabilities of A^* successfully guess a_2, b_2 are $\frac{1}{2^n}$, $\frac{1}{2^n}$ respectively. Therefore,

$$Pr[\tfrac{y'_1 - b_1}{a_1} = \tfrac{y'_2 - b_2}{a_2}] = \tfrac{1}{2^{3n}}$$

Let $Adv^{Verif}_{A^*}(\mathcal{VOC}, f, k)$ denotes the advantage of adversary A^* in the experiment $\mathbf{Exp}^{Verif}_{A^*}[\mathcal{VOC}, f, k]$, then we have

$$Adv^{Verif}_{A^*}(\mathcal{VOC}, f, k) \leq \tfrac{1}{2^{3n}} + negl(k).$$

□

Now, we are ready to show that if the homomorphic encryption scheme is semantically, then we are able to reduce the security of protocol $\mathcal{VOC}$ (with multiple executions) to the security of one time execution in which we expect the adversary to cheat. Let $\mathbf{Exp}^{Verif}_{A}$ be the security experiment corresponding to $\mathcal{VOC}$, and $\mathbf{Exp}^{Verif}_{A^*}$ be the one-time security experiment (that is, the adversary is allowed to query the oracle `ProbGen` only once, i.e., $l = 1$) corresponding to one-time execution of $\mathcal{VOC}$.

Assume for the contradiction that there exists a PPT adversary A for experiment $\mathbf{Exp}^{Verif}_{A}$ and a non-negligible function ϵ such that

$$Adv^{Verif}_{A}(\mathcal{VOC}, f, k) \geq \epsilon(k).$$

We use A to construct a PPT adversary A^* for the one-time experiment $\mathbf{Exp}^{Verif}_{A^*}$, and prove that there is a non-negligible function ϵ^* such that

$$Adv^{Verif}_{A^*}(\mathcal{VOC}, f, k) \geq \epsilon^*(k),$$

which is contradict with lemma 1, then the proof of theorem 1 is complete.

Now we build the PPT adversary A^* below.

Let L be an upper bound on the number of queries that A makes to its `ProbGen` oracle. A^* chooses randomly an index $i \in [L]$. Loosely speaking, the adversary A^* executes the one-time experiment $\mathbf{Exp}^{Verif}_{A^*}$ by simulating the adversary A executes the repeated experiment $\mathbf{Exp}^{Verif}_{A}$, while embedding his single query in the i-th query of $\mathbf{Exp}^{Verif}_{A}$, and simulating the messages in the other queries by encrypting random n-bit strings under a randomly choosing public key of FHE scheme. Formally, the PPT adversary A^* is defined as follow.

– For the j-th query of A, where $j \neq i$, A^* will simulate the j-th query of A by 1) choosing a fresh, random public/private key pair (pk_j, sk_j) for the FHE scheme; 2) encrypting random n-bit strings (a_1^j, b_1^j, r_1^j) and (a_2^j, b_2^j, r_2^j) under pk_j in a random order.
– For the i-th query, x, the adversary A^* gives x to its oracle `ProbGen` and returned back σ_x by 1) choosing a random public/private key pair (pk_i, sk_i) for the FHE scheme; 2) encrypting the input x with two randomly chosen n-bit pairs, i.e., encrypting (a_1^i, b_1^i, x) and (a_2^i, b_2^i, x) under pk_j in a random order.

Now we claim that $Adv_{A^*}^{Verif}(\mathcal{VOC}, f, k) \geq \epsilon^*(k)$ and prove it in Lemma 2 and complete the proof of Theorem 1.

Lemma 2. $Adv_{A^*}^{Verif}(\mathcal{VOC}, f, k) \geq \epsilon^*(k)$, where ϵ^* is a non-negligible function.

Proof of Lemma 2: For every $t = 0, ..., L$, we define the hybrid experiment $\mathcal{H}_A^t(\mathcal{VOC}, f, k)$ for PPT adversary A below.

Experiment $\mathcal{H}_A^t(\mathcal{VOC}, f, k)$:

- For the j-th query with $j \leq t$ and $j \neq i$: the oracle will respond by 1) choosing a random public/private key pair (pk_j, sk_j) for the FHE scheme; 2) encrypting random n-bit strings (a_1^j, b_1^j, r_1^j) and (a_2^j, b_2^j, r_2^j) under pk_j in a random order.
- For the j-th query with $j > t$ or $j = i$: the oracle will respond exactly as in $\mathcal{VOC}$ by 1) choosing a random public/private key pair (pk_j, sk_j) for the FHE scheme; 2) encrypting the correct input x_j (when $j = i$, then $x_i = x$) with 4 randomly chosen n-bit strings $a_1^j, b_1^j, a_2^j, b_2^j$, i.e., encrypting (a_1^j, b_1^j, x_j) and (a_2^j, b_2^j, x_j) under pk_j in a random order.

If A successfully cheats on the i-th input, then the experiment $\mathcal{H}_A^t$ outputs 1 and otherwise 0. Denotes $Adv_A^t(\mathcal{VOC}, f, k) = Pr[\mathcal{H}_A^t(\mathcal{VOC}, f, k) = 1]$.

According to the construction of the hybrid experiment $\mathcal{H}_A^t$, we stress the three properties:

– When $t = 0$, $\mathcal{H}_A^0(\mathcal{VOC}, f, k)$ is the same as experiment $Exp_A^{Verif}[\mathcal{VOC}, f, k]$ except for the way the output bit is computed at the end. Therefore, we have

$$Adv_A^0(\mathcal{VOC}, f, k) = \frac{Adv_A^{Verif}(\mathcal{VOC}, f, k)}{L} \geq \frac{\epsilon}{L},$$

because the index i is randomly chosen between 1 and L.
– When $t = L$, $\mathcal{H}_A^L(\mathcal{VOC}, f, k)$ is the same as the construction of the adversary A^* above, thus, we have

$$Adv_A^L(\mathcal{VOC}, f, k) = Adv_{A^*}^{Verif}(\mathcal{VOC}, f, k)$$

– For $t = 0, ..., L$, experiments $\mathcal{H}_A^t$ and $\mathcal{H}_A^{t-1}$ are computationally indistinguishable, that is, for every A,

$$|Adv_A^{t-1}(\mathcal{VOC}, f, k) - Adv_A^t(\mathcal{VOC}, f, k)| \leq negl(k)$$

The first two properties are obviously, and if we can prove the third property, then we can finish the proof of Lemma 2, because we have

$$\begin{aligned} Adv_{A^*}^{Verif} &= Adv_A^L = Adv_A^0 - \textstyle\sum_{t=1}^{L}(Adv_A^{t-1} - Adv_A^t) \\ &\geq Adv_A^0 - \textstyle\sum_{t=1}^{L}|Adv_A^{t-1} - Adv_A^t| \\ &\geq \tfrac{\epsilon}{L} - L \cdot negl(k) \triangleq \epsilon^*(k) \end{aligned}$$

Actually, the third property easily follows the semantic security of the FHE scheme. Assume that experiments $\mathcal{H}_A^t$ and $\mathcal{H}_A^{t-1}$ are computationally distinguishable. Now let's see what is the difference between $\mathcal{H}_A^t$ and $\mathcal{H}_A^{t-1}$. The only deference is the t-th query in experiments $\mathcal{H}_A^t$ and $\mathcal{H}_A^{t-1}$. Let $view_j^t$ denotes the views of the j-th query of $\mathcal{H}_A^t$, then

$$\begin{aligned} view_t^t &= (pk_t, \texttt{Fenc}_{pk_t}(a_1^t, b_1^t, r_1^t, a_2^t, b_2^t, r_2^t)) \\ view_t^{t-1} &= (pk_t, \texttt{Fenc}_{pk_t}(a_1^t, b_1^t, x_t, a_2^t, b_2^t, x_t)) \end{aligned}$$

If we could distinguish between $\mathcal{H}_A^t$ and $\mathcal{H}_A^{t-1}$, then we could distinguish between $(pk_t, \texttt{Fenc}_{pk_t}(r_1^t), \texttt{Fenc}_{pk_t}(r_2^t))$ and $(pk_t, \texttt{Fenc}_{pk_t}(x_t), \texttt{Fenc}_{pk_t}(x_t))$ which is contradict to the semantic security of the FHE scheme. □

4.2 Protocol $\mathcal{VOC}'$: Reusable Verifiable Outsourced Computation Scheme with Negligible Soundness

Our first protocol has security error $1/2^{3n} + negl(k)$, if the length of the input is mall (i.e., n is not large enough), then the security error is non-negligible on k. In order to improve this security error, the protocol could ask the server to compute F on multiple independent randomized inputs $\sigma_x^1, ..., \sigma_x^t$, where t is an additional security parameter associated with k. Upon receiving the t encrypted inputs, the server evaluates and replies the answers $F(\sigma_x^1), ..., F(\sigma_x^t)$ to the client, who checks whether these t answers are decrypted to the same value and set this consistent value as $f(x)$. The details are described as below.

Protocol $\mathcal{VOC}'$.
Let k be security parameter, `FHE=(Fkeygen,Fenc,Fdec,Feval)` be a full homomorphic encryption scheme. Suppose the computation function f is in domain $\{0,1\}^n$.

1. `KeyGen`$(f, k) \rightarrow (PK, SK)$. The algorithm `KeyGen` first encodes the function f with n-bit input x into a function F with $3n$-bit input $a\|b\|x$, such that $F(a\|b\|x) = af(x) + b$, where a, b are the variables in domain $\{0,1\}^n$. Then it represents F as a circuit C such that $C(z) = F(z)$ for any $z \in \{0,1\}^{3n}$. The circuit C will be evaluated by the server instead of f. The algorithm then sets the public key $PK = C$ and the secret key SK is the positions of bites of strings a, b, x.
2. `ProbGen`$_{SK}(x) \rightarrow (\sigma_x, \tau_x)$. The algorithm `ProbGen` is executed by the client as below.
   - Calls the key generation algorithm `Fkeygen` of FHE to create a key pair: $(pk, sk) \leftarrow$ `Fkeygen`(k).

- Chooses randomly t pairs $\tau_x^1 \triangleq (a_1, b_1), ..., \tau_x^t \triangleq (a_t, b_t)$ for the variables (a, b), where $a_i, b_i \in \{0,1\}^n, i \in [t]$.
- Runs the encryption algorithm `Fenc` of FHE to encrypt the tuples (a_1, b_1, x), ..., (a_t, b_t, x):

$$\sigma_x^1 \triangleq (\hat{a}_1, \hat{b}_1, \hat{x}_1) \leftarrow \texttt{Fenc}_{pk}(a_1, b_1, x),$$

$$\cdots$$

$$\sigma_x^t \triangleq (\hat{a}_t, \hat{b}_t, \hat{x}_t) \leftarrow \texttt{Fenc}_{pk}(a_t, b_t, x)$$

- Sets the public value $\sigma_x = (pk, \sigma_x^1, ..., \sigma_x^t)$, and the private value $\tau_x = (sk, \tau_x^1, ..., \tau_x^t)$

3. $\texttt{Compute}_{PK}(\sigma_x) \rightarrow \sigma_y$. The `Compute` algorithm is executed by the server to compute the value of circuit C as below. Calls the evaluation algorithm `Feval` of the FHE and calculates

$$\texttt{Feval}_{pk}(\sigma_x^1, C) \rightarrow \sigma_y^1,$$

$$\cdots$$

$$\texttt{Feval}_{pk}(\sigma_x^t, C) \rightarrow \sigma_y^t.$$

Then the output is $\sigma_y = (\sigma_y^1, ..., \sigma_y^t)$, where $\sigma_y^1, ..., \sigma_y^t$ are the ciphertexts of $F(x, a_1, b_1), ..., F(x, a_t, b_t)$ respectively according to the properties of the FHE.

4. $\texttt{Verify}_{SK}(\tau_x, \sigma_y) \rightarrow y \cup \perp$. The client uses the decryption algorithm `Fdec` of FHE to decrypt the t ciphertexts $\sigma_y^1, ..., \sigma_y^t$ under secret key sk, obtaining $F(x, a_1, b_1)$, ..., $F(x, a_t, b_t)$; Then he uses $\tau_x^1 = (a_1, b_1), ..., \tau_x^t = (a_t, b_t)$ to decode $F(x, a_1, b_1)$, ..., $F(x, a_t, b_t)$, if $(F(x, a_1, b_1) - b_1)a_1^{-1} = \cdots = (F(x, a_t, b_t) - b_t)a_t^{-1} \triangleq y$, sets the consistent value y as the output of $f(x)$, otherwise, output $\perp$.

Lemma 3. Assume that the full homomorphic encryption scheme is semantically secure, then the verifiable outsourced computation scheme $\mathcal{VOC}'$ has one-time soundness error $\frac{1}{2^{2(t-1)n+n}} + negl(k)$, which is negligible in k

Theorem 2. Let `FHE` be a semantically secure full homomorphic encryption scheme. Then protocol $\mathcal{VOC}'$ is a correct, secure, private and efficient non-interactive verifiable outsourced computation scheme.

The proofs of Lemma 3 and Theorem 2 are similar to the proofs of Lemma 1 and Theorem 1.

5 Conclusion

To our best of knowledge, the existing non-interactive verifiable outsourced computation schemes for arbitrary functions have offline complexity of $poly(T)$, where T is the time complexity of computing the functions. In this work, we propose a non-interactive verifiable outsourced computation scheme for arbitrary functions satisfying the four properties defined in [GGP10] (i.e., correctness, privacy, security, efficiency), which invests $poly(logT)$ time in offline stage.

However, we are not sure if our scheme is secure in the presence of verification queries defined in [GP14], so in the next step, we'll carefully concern about the security model in which the adversary is allowed to issue verification queries to the client.

Reference	offline complexity(C)	$\|PK\|$	$\|SK\|$	online complexity(C)	online complexity(S)
GGP10	$poly(k,T)$	$poly(k,T)$	$poly(k,n)$	$poly(k,n,logT)$	$poly(k,T)$
CKV10	$poly(k,T)$	0	$poly(k,n)$	$poly(k,n,logT)$	$poly(k,T)$
Ours	$poly(k,logT)$	$\|C\|$	$poly(k,n)$	$poly(k,n,logT)$	$poly(k,T)$

Table 1. The comparison of the complexity.

References

1. A. Yao. Protocols for secure computations. In Proceedings of the IEEE Symposium on Foundations of Computer Science, 1982.
2. L*á*szl*ó* Babai. Trading group theory for randomness. In Robert Sedgewick, editor, STOC, pages 421-429. ACM, 1985.
3. A. Yao. How to generate and exchange secrets. In Proceedings of the IEEE Symposium on Foundations of Computer Science, 1986.
4. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. SIAM J. Comput., 18(1):186-208, 1989.
5. L*á*szl*ó* Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. In FOCS, pages 16-25. IEEE Computer Society, 1990.
6. L*á*szl*ó* Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in poly-logarithmic time. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, STOC, pages 21-31. ACM, 1991.
7. Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; a new characterization of np. In FOCS, pages 2-13. IEEE Computer Society, 1992.
8. Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, STOC, pages 723-732. ACM, 1992.
9. Silvio Micali. Cs proofs (extended abstracts). In FOCS, pages 436-453. IEEE Computer Society, 1994.
10. Joe Kilian. Improved efficient arguments (preliminary version). In Don Coppersmith, editor, CRYPTO, volume 963 of Lecture Notes in Computer Science, pages 311-324. Springer, 1995.
11. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. Journal of the ACM, 51(4):557-594, 2004.
12. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, STOC, pages 113-122. ACM, 2008.
13. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor,STOC, pages 169-178. ACM, 2009.

14. Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In Rabin [Rab10],pages 116-137.
15. Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In CRYPTO, volume 6223 of Lecture Notes in Computer Science, pages 483-501. Springer, 2010.
16. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Rabin [Rab10], pages 465-482.
17. Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In Kenneth G. Paterson, editor, EUROCRYPT, volume 6632 of Lecture Notes in Computer Science, pages 129-148. Springer, 2011.
18. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Goldwasser [Gol12], pages 309-325.
19. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, CRYPTO, volume 7417 of Lecture Notes in Computer Science, pages 868-886. Springer, 2012.
20. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, EUROCRYPT, volume 7237 of Lecture Notes in Computer Science, pages 465-482. Springer, 2012.
21. Rosario Gennaro, Valerio Pastroeprint. Verifiable Computation over Encrypted Data in the Presence of Verification Queries. IACR Cryptology ePrint Archive, 2014. http://iacr.org/2014/202.pdf