Faster Non-interactive Verifiable Computing

Pascal Lafourcade[1], Gael Marcadet[1], and Léo Robert[1]

Université Clermont-Auvergne, CNRS,
Mines de Saint-Étienne, LIMOS,
63000 Clermont-Ferrand, France.
firstname.lastname@uca.fr

Abstract. In 1986, A.Yao introduced the notion of *garbled circuits*, designed to verify the correctness of computations performed on an untrusted server. However, correctness is guaranteed for only one input, meaning that a new garbled circuit must be created for each new input. To address this drawback, in 2010 Gennaro *et al.* performed the evaluation of the garbled circuit homomorphically using Fully Homomorphic Encryption scheme, allowing to reuse the same garbled circuit for new inputs. Their solution requires to encrypt the garbled circuit at every new input. In this paper, we propose a verifiable-computation scheme allowing to verify the correctness of computations performed by an untrusted server for multiple inputs, where the garbled circuit is homomorphically encrypted only once. Hence, we have a faster scheme comparing to Gennaro's solution, since for each new input, we reduce the computations by the size of the circuit representing the function to be computed, for the same security level. The key point to obtain this speed-up is to rely on Multi-Key Homomorphic Encryption (MKHE) and then to encrypt only once the garbled circuit.

1 Introduction

With the Cloud-As-A-Service model, clients are interested to outsource their computations on an external cloud managed by a third-party. One goal is to limit the cost due to the maintenance and ensure security of the architecture. Two problems are faced by the client when computations are outsourced: *correctness* and *privacy*. The client wants a result computed as expected, even if computations are performed on a cloud. In addition, the client wants to keep the ownership of the data, meaning that the cloud should not learn any information from the input, the output but also all intermediate computations. Many applications require to perform verifiable computation, such that billing systems [PHCP13] or e-voting [Cha04, CRS05].

The Fully Homomorphic Encryption (FHE), introduced for the first time by Gentry [Gen09], allows a server to perform any computations over encrypted data without relying on a decryption key. A typical use-case of FHE is the delegation of a computation from a client to a *semi-trusted* server which follows a protocol as expected but could try to learn as much information as possible. FHE schemes can be improved in two ways, either increasing the performance

of homomorphic operations, or introducing new features such as Multi-Key Homomorphic Encryption (MKHE). Intuitively, MKHE allows a server to perform a computation over two (or more) encrypted inputs, encrypted under *different* keys.

A *verifiable-computation scheme* allows a client to verify the computation $f(x) = y$ of a function f over an input x, performed by a malicious server. In a verifiable-computation scheme, the server performs the function f over σ_x instead over x, preventing the server to learn some information about the input x. The server computes σ_y the encoding of $y = f(x)$ by using the verifiable-computation scheme. At the end, if the computations done by the server are valid, then the client can to retrieve $f(x) = y$ from σ_y, else the client can detect that the computed result is malformed.

In 1986, Andrew Yao introduced the notion of *garbled circuit* [Yao86], a verifiable-computation scheme designed to verify the computation of $f(x)$ performed by a malicious server, for any function f represented as a boolean circuit and any input x. A boolean circuit is a set of *gates* (*e.g.,* AND, XOR) connected together with *wires*, allowing to transmit the value of a bit from a gate to another. Each gate takes two *input wires* and outputs a single *output wire*. The value provided to the input wires of the first gates of the circuit is the binary representation of the input x. The output wires of the last gates of the circuit output the binary representation of the output $f(x)$. In a garbled circuit, the encoding of the input x, denoted σ_x, is constructed via Oblivious Transfer and symmetric encryption scheme which is required to be *Yao-Secure*. We depict in Fig. 1 the general workflow of a protocol using garbled circuits.

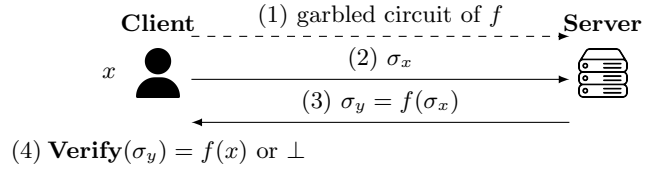

Fig. 1: Overview of garbled circuits for delegation of $y = f(x)$. By σ_x, σ_y we denote respectively to the encoding of x, y. The **Verify** function returns $f(x)$ if given encoding is correct, $\perp$ if encoding σ_y is incorrect, meaning that the server has returned an invalid result.

The computation of $f(x)$ are delegated using a garbled circuit; the client starts by creating the boolean circuit simulating f. For the sake of clarity, let $f(a, b) = a \wedge b$ the boolean circuit representing the AND function, taking two input bits a and b, producing 1 if $a = b = 1$ else 0.

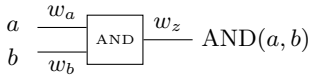

Fig. 2: Boolean circuit of the AND gate.

The AND gate, represented in Fig. 2, is composed of two input wires denoted w_a and w_b used to transmit the bit value of a and b respectively, to the gate. The output wire denoted w_y, which handles the value of the bit computed by the AND gate corresponds to AND(a, b). Yao's idea to prevent the server to learn information about bit that the server handles, is that for each wire w composing the boolean circuit of f, we pick two uniformly chosen random *labels* denoted k_w^0 and k_w^1, corresponding to 0 and 1 respectively. Note that labels are defined in higher space than bit. As an example, for the wire w_a, we have two labels $k_{w_a}^0$ and $k_{w_a}^1$. We replace each bit by its corresponding label. Hence, instead of sending bits a and b to the server, the client sends $k_{w_a}^a$ and $k_{w_b}^b$ the *encoding* respectively of a and b. Since the server sees only the encoding of the inputs and not the inputs itself, and since each label is randomly chosen, the serve cannot infer the real values of the inputs a and b.

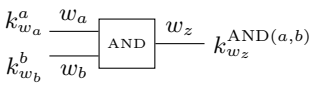

Fig. 3: Boolean circuit of the AND gate where each bit has been replaced by a random label.

Instead of computing $f(a, b)$, the server computes $f(k_{w_a}^a, k_{w_b}^b)$ which is intuitively difficult since the boolean circuit f works only over bits and not over labels.

w_a	w_b	w_z
0	0	0
0	1	0
1	0	0
1	1	1

w_a	w_b	w_z
$k^0_{w_a}$	$k^0_{w_b}$	$k^0_{w_z}$
$k^0_{w_a}$	$k^1_{w_b}$	$k^0_{w_z}$
$k^1_{w_a}$	$k^0_{w_b}$	$k^0_{w_z}$
$k^1_{w_a}$	$k^1_{w_b}$	$k^1_{w_z}$

Garbled Table (known by the server)
$\gamma^{00}_{\mathrm{AND}} = \mathrm{Enc}_{k^0_{w_a}}(\mathrm{Enc}_{k^0_{w_b}}(k^0_{w_z}))$
$\gamma^{01}_{\mathrm{AND}} = \mathrm{Enc}_{k^0_{w_a}}(\mathrm{Enc}_{k^1_{w_b}}(k^0_{w_z}))$
$\gamma^{10}_{\mathrm{AND}} = \mathrm{Enc}_{k^1_{w_a}}(\mathrm{Enc}_{k^0_{w_b}}(k^0_{w_z}))$
$\gamma^{11}_{\mathrm{AND}} = \mathrm{Enc}_{k^1_{w_a}}(\mathrm{Enc}_{k^1_{w_b}}(k^1_{w_z}))$

Fig. 4: The left table corresponds to the truth table for AND gate. The middle table corresponds to the truth table where for each wire $w \in \{w_a, w_b, w_z\}$, we have replaced 0 (resp. 1) to uniformly chosen random label k_w^0 (resp. k_w^1). Note that the two leftmost tables are not known by the server. The right table corresponds to the garbled table of the gate AND which is sent to the server.

The execution of the AND gate over labels is possible by considering the usage of a symmetric encryption scheme. For each gate, we construct four ciphertexts, one ciphertext for each possible AND result. By *garbled table*, we denote the set of four ciphertexts. In Fig. 4, we present the garbled table of our AND gate. To construct the garbled table of the AND gate, we compute the set $\{\gamma^{00}_{\text{AND}}, \gamma^{01}_{\text{AND}}, \gamma^{10}_{\text{AND}}, \gamma^{11}_{\text{AND}}\}$ as detailed in Fig. 4 using the symmetric encryption scheme. To simulate the function f over the labels $k^a_{w_a}$ and $k^b_{w_b}$, the server decrypts γ^{00}_{AND}. Supposing that the symmetric encryption scheme allows to know if the decryption of γ^{00}_{AND} fails, the server decrypts γ^{01}_{AND} then γ^{10}_{AND} and finally γ^{11}_{AND}, until a successfully decryption. The symmetric encryption scheme used in the garbled circuit is *not* commutative. Hence, each ciphertext can be decrypted using the correct combination of two labels used as keys of the ciphertext. After a valid decryption, the server obtains a new label, which is our example $k^{\text{AND}(a,b)}_{w_z}$ since the boolean circuit of the AND gate is composed by a single gate. Then, the server sends back $k^{\text{AND}(a,b)}_{w_z}$ to the client, which knows the mapping of the bit value associated to $k^{\text{AND}(a,b)}_{w_z}$. The so-called *garbled circuit* corresponds to the set of all garbled tables. The server knows only the input labels $k^a_{w_a}$ and $k^b_{w_b}$, from which the server simulates f using the garbled circuits. At the end of the simulation, the server has access to the output label $k^{\text{AND}(a,b)}_{w_z}$, which is returned to the client able to retrieve $\text{AND}(a, b)$.

Suppose that the client wants to reuse the garbled circuits simulating the AND function over a' and b'. We assume that a' is different from a; or b' is different from b. As explained above, the client sends to the server the labels $k^{a'}_{w_a}$ and $k^{b'}_{w_b}$ to the server. The server could simulate f using the garbled circuit to get $k^{\text{AND}(a',b')}_{w_z}$ as expected. But the server is also able to simulate f to get either $k^{\text{AND}(a',b)}_{w_z}$ or $k^{\text{AND}(a,b')}_{w_z}$ by replacing one input label with an input label of the previous execution (*e.g.,* replace $k^{a'}_{w_a}$ with $^{a}_{w_a}$), or to reuse the output label $k^{\text{AND}(a,b)}_{w_z}$ computed at the previous execution. Therefore, a garbled circuit is a *one-time* verifiable-computation scheme. To compute the same function over two different inputs, the client must generate two distinct garbled circuits which is not efficient. An ideal verifiable-computation scheme should be *reusable* for any number of inputs, while still ensuring the same verifiability property.

Contribution. We address the one-time limitation of Yao's garbled circuit by introducing a verifiable-computation scheme denoted $\mathcal{VC}_{mk}$ based on Multi-Key Homomorphic Encryption [CCS19, AJJM20, AJJ20, CDKS19]. Our verifiable-computation scheme uses several MKHE keys: one key dedicated to encrypt the input and one key used to encrypt the garbled circuit once. This approach allows our scheme to be faster than state-of-the-art of garbled-circuit homomorphic encryption based verifiable-computation scheme. As an overview, our verifiable-computation scheme works in three steps, depicted in Fig. 5.

– At step (1), executed once, the client generates two distinct key pairs: a *function-keypair* (pk_f, sk_f) and an *input-keypair* (pk_x, sk_x). The client sends to the server the garble tables encrypted under pk_f.
– At step (2), the client creates σ_x the encoding of the input x, encrypted under pk_x.
– At step (3), the server evaluates the boolean circuit using the encrypted garbled tables and the encrypted encoding inputs to obtain σ_y the encoding of $y = f(x)$, encrypted under both pk_f and pk_x. The client retrieves the decrypted result using the secret keys sk_f and sk_x.

With this settings, if the client wants to compute f over a new input x', the client generates a new key pair $(pk_{x'}, sk_{x'})$ used to encrypt the encoding of x'. The garbled circuit is then encrypted only once, which makes our scheme more efficient, compared to the state-of-the-art [GGP10]. In details, for each new input, we reduce the required computations by the size of the circuit representing a function f to be evaluated. Our scheme is *correct*, meaning that an honest server can to provide a valid computation, accepted by the client. Also, our scheme is *secure* (with respect to the verifiable-computation scheme secure property) meaning that a malicious server can provide an invalid computation without being noticed by the client only with a negligible probability. In addition, our scheme is *private* meaning that the server cannot learn information about the input, the output but also any intermediate computations. Finally, our scheme is *outsourceable*, meaning that the client performs less computations by outsourcing the computations of a function f compared to the case where the client computes f locally. At the exception of the initialization phase, the functions executed by the client depends only on the size of the inputs and the outputs, regardless about the size of f.

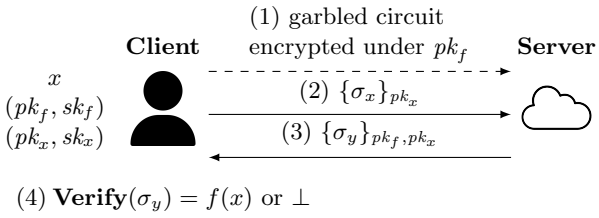

Fig. 5: Overview of our scheme for delegation of two inputs $y = f(x)$. By σ_x (resp. σ_y) we denote the encoding of x (resp. y).

In this paper, we propose a verifiable-computation scheme based on garbled circuit improved using a MKHE scheme.

Related Work. Verifiability of computation done by an untrusted server is an active research topic. *Interactive Proofs* [Kil92] [Mic00] allow a server to prove the validity of a statement (or computations in our case) to a client. The verification process is performed during an interaction between the server and the client.

Probability Checkable Proofs (PCP) allows a client to detect malicious server with a probability which increases after many executions. We focus on *Non-Interactive Proofs* where the client and the server interact only to transmit during an initialization and a result obtaining steps. Arguments from [GKR15] or Micali's CS proofs [Mic00] are the state-of-the-art proofs in non-interactive PCPs. In this work, we move away from notions of PCP to focus on *garbled circuits*. The garbled circuits introduced by Yao [Yao86] is a *one-time* non-interactive verifiable-scheme able to verify the correctness of computations performed by an untrusted server. Garbled circuits are extensively studied by the research community to increase the reusability [HKK+14, Lin16, HKE13]. Indeed, the original version garbled circuit suffers of a one-time limitation problem, restricting the verification to only one input. This one-time limitation has been addressed by using *Cut-And-Choose* [LP15, HKK+14, GGP10] technique or by using Homomorphic Encryption [GGP10]. In this paper, we focus on Gennaro *et al.* [GGP10] work which addresses the one-time verifiability limitation of the garbled circuit using Fully Homomorphic Encryption, introduced for the first time by Gentry in 2009 [Gen09]. They proposed a verifiable-computation scheme $\mathcal{VC}_{gen}$ where the evaluation of the garbled circuit is done homomorphically by a server. Although their approach allows to reuse the same garbled circuits without limitation in the number of inputs, the garbled circuit should be entirely encrypted with a different key, each time that the client wants to compute f over a new input.

We go further in this last direction and improve significatively the Gennaro *et al.* result by using MKHE scheme.

Outline. In Section 2, we introduce preliminaries including a definition of verifiable-computation scheme, garbled circuit and a MKHE scheme. In Section 3, we formally introduce the security definitions of verifiable-computation scheme. Then, in Section 4, we present the verifiable-computation scheme $\mathcal{VC}_{gen}$ introduced in [GGP10], followed by the description of our scheme $\mathcal{VC}_{mk}$ based on MKHE. In Section 5, we prove that $\mathcal{VC}_{mk}$ is a correct, secure, private and outsourceable verifiable-computation scheme. Finally, in Section 6, we detail a performance comparison between $\mathcal{VC}_{mk}$ and $\mathcal{VC}_{gen}$ schemes.

2 Preliminaries

First, we present the definition of a verifiable-computation scheme. Then, we present the Yao's garbled circuits. Finally, we present the definition of a MKHE scheme.

2.1 Definition of Verifiable-Computation Scheme

A verifiable-computation scheme is a method allowing the verification of computations outsourced on an untrusted server. A verifiable-computation scheme works in four steps: suppose that a client wants to delegate $y = f(x)$ the computation of the function f over an input x to a server:

1. The client generates a key pair denoted $(\mathrm{PK}, \mathrm{SK})$ where the public part PK is the encoding of f and SK is the matching secret key.
2. Then the client generates σ_x the public encoding of x, given as an input to the server, and the secret verification string denoted τ_x, kept private by the client and used to verify the result returned by the server.
3. Given the public encoding of f and σ_x, the server sends back to the client the encoding of y denoted σ_y.
4. Finally, the client retrieves y from σ_y using the secret verification τ_x and SK.

Definition 1 (Verifiable-computation scheme). *A* verifiable-computation scheme $\mathcal{VC}$ *is a tuple composed of four algorithms* $\mathcal{VC} = (\boldsymbol{KeyGen}, \boldsymbol{ProbGen}, \boldsymbol{Compute}, \boldsymbol{Verify})$ *where:*

- $\boldsymbol{KeyGen}(f, \lambda) \rightarrow (PK, SK)$*: Given a function* f *and a security parameter* λ*, outputs a public key PK that encodes the function* f*, and a secret key SK.*
- $\boldsymbol{ProbGen}_{SK}(x) \rightarrow (\sigma_x, \tau_x)$*: Given the input* x *and the private key SK, returns* σ_x *the public encoding of input* x *used by the server to compute* $y = f(x)$*, and* τ_x *a secret verification string, which is different for every input, used for to verify the correctness of computations.*
- $\boldsymbol{Compute}_{PK}(\sigma_x) \rightarrow \sigma_y$*: Given the public key PK and the encoded input* σ_x*, returns* σ_y *the encoded output.*
- $\boldsymbol{Verify}_{SK}(\tau_x, \sigma_y) \rightarrow y \cup \perp$*: Given the private key SK and* τ_x *the secret verification string and* σ_y *the encoded output, outputs* y *if the correctness of computations is verified,* $\perp$ *otherwise.*

A verifiable-computation scheme $\mathcal{VC}$ has several properties. $\mathcal{VC}$ must be *correct* meaning that a verifiable-computation scheme for any input x, an honest server can compute $f(x)$. $\mathcal{VC}$ must be *secure* meaning that the client is able to detect if a malicious server returns $\hat{y}$, an invalid result instead of the valid result $f(x)$, with a high probability. $\mathcal{VC}$ must be *private* meaning that the server cannot learn some information about the inputs x and the output $f(x)$. Finally, $\mathcal{VC}$ must be *outsourceable* meaning that the client must perform strictly less operations than computing the function by himself.

2.2 Yao's Garbled Circuit

Suppose that a client wants to delegate $f(x)$, the computation of a function f over an input x to an untrusted server. The first step performed by the client is to compute $\Delta_f = (\mathcal{G}, \mathcal{W}, \mathcal{W}_{\text{in}}, \mathcal{W}_{\text{out}})$ the boolean circuit corresponding to f, where $\mathcal{G}$ is the set of gates in Δ_f, $\mathcal{W}$ is the set of wires in Δ_f, $\mathcal{W}_{\text{in}}$ (resp. $\mathcal{W}_{\text{out}}$) are the wires at the input (resp. output) of Δ_f, with $\mathcal{W}_{\text{in}} \subset \mathcal{W}$ (resp. $\mathcal{W}_{\text{out}} \subset \mathcal{W}$). Each gate $g \in \mathcal{G}$ takes as input two wires $w_a, w_b \in \mathcal{W}$ and as an output a wire $w_z \in \mathcal{W}$, denoted by $(g : w_a, w_b \rightarrow w_z)$. For each wire $w \in \mathcal{W}$, the client chooses uniformly at random two random *labels* k_w^0, k_w^1 corresponding respectively to 0 (wire off) and 1 (wire on). Once each wire has been affected with two random

labels, then for each gate $(g : w_a, w_b \to w_z)$ we compute γ_g as follows:

$$\begin{aligned}
\gamma_g &= (\gamma_g^{00}, \gamma_g^{01}, \gamma_g^{10}, \gamma_g^{11}) \\
\gamma_g^{00} &= \mathrm{Enc}_{k_{w_a}^0}(\mathrm{Enc}_{k_{w_b}^0}(k_{w_z}^{g(0,0)})) \\
\gamma_g^{01} &= \mathrm{Enc}_{k_{w_a}^0}(\mathrm{Enc}_{k_{w_b}^1}(k_{w_z}^{g(0,1)})) \\
\gamma_g^{10} &= \mathrm{Enc}_{k_{w_a}^1}(\mathrm{Enc}_{k_{w_b}^0}(k_{w_z}^{g(1,0)})) \\
\gamma_g^{11} &= \mathrm{Enc}_{k_{w_a}^1}(\mathrm{Enc}_{k_{w_b}^1}(k_{w_z}^{g(1,1)}))
\end{aligned} \tag{1}$$

where $\mathrm{Enc}_k(\cdot)$ is a symmetric encryption function. In Fig. 6, we depicted the setup of a garbled table for a gate $g \in \mathcal{G}$. From the above equation (1), we can construct the *garbled circuit* $\gamma = \{\gamma_g | g \in \mathcal{G}\}$.

w_a	w_b	w_z
0	0	$g(0,0)$
0	1	$g(0,1)$
1	0	$g(1,0)$
1	1	$g(1,1)$

w_a	w_b	w_z
$k_{w_a}^0$	$k_{w_b}^0$	$k_{w_z}^{g(0,0)}$
$k_{w_a}^0$	$k_{w_b}^1$	$k_{w_z}^{g(0,1)}$
$k_{w_a}^1$	$k_{w_b}^0$	$k_{w_z}^{g(1,0)}$
$k_{w_a}^1$	$k_{w_b}^1$	$k_{w_z}^{g(1,1)}$

Garbled Table (known by the server)
$\gamma_g^{00} = \mathrm{Enc}_{k_{w_a}^0}(\mathrm{Enc}_{k_{w_b}^0}(k_{w_z}^{g(0,0)}))$
$\gamma_g^{01} = \mathrm{Enc}_{k_{w_a}^0}(\mathrm{Enc}_{k_{w_b}^1}(k_{w_z}^{g(0,1)}))$
$\gamma_g^{10} = \mathrm{Enc}_{k_{w_a}^1}(\mathrm{Enc}_{k_{w_b}^0}(k_{w_z}^{g(1,0)}))$
$\gamma_g^{11} = \mathrm{Enc}_{k_{w_a}^1}(\mathrm{Enc}_{k_{w_b}^1}(k_{w_z}^{g(1,1)}))$

Fig. 6: The table at left corresponds to the truth table for the gate $(g : w_a, w_b \to w_z)$. The table at center corresponds to the truth table where for each wire $w \in \{w_a, w_b, w_z\}$, we have replaced 0 (resp. 1) to uniformly chosen random label k_w^0 (resp. k_w^1). The table at right corresponds to the garbled table of the gate g which is sent to the server.

Definition 2 (Yao's garbled circuit). *A Yao's garbled circuit is a verifiable-computation scheme denoted* $\mathcal{VC}_{yao} = (\boldsymbol{KeyGen}^{yao}, \boldsymbol{ProbGen}^{yao}, \boldsymbol{Compute}^{yao}, \boldsymbol{Verify}^{yao})$ *where:*

- $\boldsymbol{KeyGen}^{yao}(f, \lambda) \to (PK, SK)$: *Computes* $\Delta_f = (\mathcal{G}, \mathcal{W}, \mathcal{W}_{in}, \mathcal{W}_{out})$ *the boolean circuit corresponding to* f*. For each wire* $w \in \mathcal{W}$*, chooses two random labels* $k_w^0, k_w^1 \in_R \{0,1\}^\lambda$*. Then, computes the garbled circuit* $\gamma = \{\gamma_g | g \in \mathcal{G}\}$ *where* γ_g *is defined as in equation (1). Outputs* $PK = \gamma$ *and* $SK = \cup_{w \in \mathcal{W}} \{k_w^0, k_w^1\}$*.*
- $\boldsymbol{ProbGen}_{SK}^{yao}(x) \to (\sigma_x, \tau_x)$: *Given binary encoding* $x_1, \ldots, x_n$ *of input* x*, outputs* $\sigma_x = \{k_{w_1}^{x_1}, \ldots, k_{w_n}^{x_n}\}$ *the set of labels associated to the wire regarding on the input bits, and* $\tau_x = SK$*.*
- $\boldsymbol{Compute}_{PK}^{yao}(\sigma_x) \to \sigma_y$: *Given* σ_x *the set of labels representing the binary encoding of input* x*, outputs* $\sigma_y = \{k_{w_1}^{y_1}, \ldots, k_{w_m}^{y_m}\}$ *the set of labels representing the binary encoding* $y_1, \ldots, y_m$ *of the output* $y = f(x)$*, by computing gate-by-gate using the garble circuit* γ*.*

– $\textbf{\textit{Verify}}^{yao}_{SK}(\sigma_y, \tau_x) \rightarrow y \cup \perp$: *Given σ_y the set of labels representing the binary encoding of $y = f(x)$ and τ_x, outputs $y = y_1, \ldots, y_m$ only if for all $i \in [1, m]$ we have $k^{y_i}_{w_i} \in \{k^0_{w_i}, k^1_{w_i}\}$, otherwise the server is cheating, thus we refuse the result with $\perp$.*

We stress that the $\mathcal{VC}_{yao}$ scheme is one-time verifiable-computation scheme at the condition that the symmetric encryption scheme E, used for the double encryption of labels, must be *Yao-Secure*.

Definition 3 (Yao-Secure Symmetric Encryption Scheme). *A symmetric encryption scheme E is Yao-Secure if:*

– E *is* indistinguishable under multiple messages*, meaning that for every two vectors of polynomial $\bar{x}$ and $\bar{y}$, no polynomial time adversary can distinguish between an encryption of $\bar{x}$ or $\bar{y}$.*
– E *has an* elusive range*, meaning that the encryption of a message falls under different keyspaces depending on the encryption key used.*
– E *has an* efficient checkable range *property, where given a key k and a ciphertext c, there is a polynomial time algorithm able to check if c has been encrypted under k.*

2.3 Multi-Key Homomorphic Encryption Scheme

Introduced in 2009 by Gentry [Gen09], the Fully Homomorphic Encryption (FHE) allows to perform any boolean circuit over encrypted data. A traditional use-case of FHE starts by a client who sends to an untrusted server $\{x\}_{pk}$ an input x encrypted under a public key pk. Then, the server sends back $\{f(x)\}_{pk}$ the result of the function f over x, without relying on the decryption key sk. The MKHE is a natural extension of FHE where the evaluation of a circuit is performed over inputs encrypted under different keys. Many MKHE schemes have been designed these last years [CCS19, BP16, CZW17], showing the interest of this approach for Multi-Party Computations. Note that the number of possible keys is unbounded for MKHE, yet we give the definition for only 2 keys as needed in our setup and for clarity.

Definition 4 (Multi-Key Homomorphic Encryption Scheme). *A* MKHE *scheme is a tuple* $\mathsf{M} = (\textbf{\textit{MKKeyGen}}, \textbf{\textit{MKEnc}}, \textbf{\textit{MKEval}}, \textbf{\textit{MKDec}})$ *where:*

– $\textbf{\textit{MKKeyGen}}(\lambda) \rightarrow (pk, sk)$*: Given a security parameter λ, outputs a new key pair (pk, sk) with pk the public key and sk the secret key.*
– $\textbf{\textit{MKEnc}}(m, pk) \rightarrow \{m\}_{pk}$*: Given a message m and a public key pk, outputs $\{m\}_{pk}$ the message m encrypted under pk.*
– $\textbf{\textit{MKEval}}(f, \{m_0\}_{pk_0}, \{m_1\}_{pk_1}) \rightarrow \{f(m_0, m_1)\}_{pk_0, pk_1}$*: Given $f \in \mathcal{F}$ a function and $\{m_0\}_{pk_0}, \{m_1\}_{pk_1}$ two messages encrypted respectively under public key pk_0 and pk_1, outputs $\{f(m_0, m_1)\}_{pk_0, pk_1}$ the image of the function f over m_0, and m_1 encrypted under both pk_0 and pk_1.*
– $\textbf{\textit{MKDec}}(\{m\}_{pk_0, pk_1}, \{sk_0, sk_1\}) \rightarrow m$*: Given $\{m\}_{pk_0, pk_1}$ the message m encrypted under pk_0 and pk_1, the related secret keys sk_1 and sk_1, outputs m.*

$\mathbf{Exp}^{\mathbf{Sec}}_{\mathcal{A}}[\mathcal{VC}, f, \lambda]$
$(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathbf{KeyGen}(f, \lambda)$
for $i \in \{1, \ldots, l = \mathrm{poly}(\lambda)\}$
$x_i \leftarrow \mathcal{A}(\sigma_{x_1}, \ldots, \sigma_{x_{i-1}})$
$(\sigma_{x_i}, \tau_{x_i}) \leftarrow \mathbf{ProbGen}(x_i)$
$(i, \sigma_{\hat{y}}) \leftarrow \mathcal{A}(\mathrm{PK}, \sigma_{x_1}, \ldots, \sigma_{x_l})$
$\hat{y} \leftarrow \mathbf{Verify}_{\mathrm{SK}}(\tau_{x_i}, \sigma_{\hat{y}})$
if $\hat{y}_l \neq \perp$ and $\hat{y}_l \neq f(x_l)$ then returns 1 else 0

Fig. 7: Experiment for the security of a verifiable-computation scheme. By $\mathrm{poly}(\lambda)$, we denote a polynomial function depending on the security parameter λ.

3 Security Properties of Verifiable-Computation Scheme

As described informally in previous sections, we give the formal definitions of usual security properties of a verifiable-computation scheme.

A verifiable-computation scheme is *correct* if a server following the scheme as expected is able to return to the client a valid result.

Definition 5 (Correctness). *Let $\mathcal{VC}$ a verifiable-computation scheme, f a function, and $(PK, SK) \leftarrow \boldsymbol{KeyGen}(f, \lambda)$. A verifiable computation scheme $\mathcal{VC}$ is* correct *if for any valid input x, we have:*

$$f(x) = \boldsymbol{Verify}_{SK}(\boldsymbol{Compute}_{PK}(\boldsymbol{ProbGen}_{SK}(x)))$$

A verifiable-computation scheme is *outsourceable* if the client performs strictly less operations by using the scheme, compared to the execution of the function by itself.

Definition 6 (Outsourceability). *Let f a function, a verifiable-computation scheme $\mathcal{VC}$ is* outsourceable *if the asymptotic complexities of **ProbGen** and **Verify** are strictly lower than the asymptotic complexity of the fastest algorithm to compute f.*

A verifiable-computation scheme is *secure* if the client is able to detect an invalid result produced by the server with a high probability.

Definition 7 (Security). *Let $\mathcal{VC}$ a verifiable-computation scheme, f a function, λ a security parameter and $(PK, SK) \leftarrow \boldsymbol{KeyGen}(f, \lambda)$. We state that $\mathcal{VC}$ a verifiable computation scheme is* secure *if for every PPT adversary $\mathcal{A}$, we have:*

$$\boldsymbol{Adv}^{\boldsymbol{Sec}}_{\mathcal{A}}[\mathcal{VC}, f, \lambda] \leq negl(\lambda)$$

where $\boldsymbol{Adv}^{\boldsymbol{Sec}}_{\mathcal{A}}[\mathcal{VC}, f, \lambda] = \Pr[\boldsymbol{Exp}^{\boldsymbol{Sec}}_{\mathcal{A}}[\mathcal{VC}, f, \lambda] = 1]$, and the experiment $\boldsymbol{Exp}^{\boldsymbol{Sec}}_{\mathcal{A}}$ is given in Figure 7.

A verifiable-computation scheme is *private* if the server does not learn information about the input, or with a negligible probability.

$\mathbf{Exp}^{\mathbf{Priv}}_{\mathcal{A}}[\mathcal{VC}, f, \lambda]$
$(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathbf{KeyGen}(f, \lambda)$
$(x_0, x_1) \leftarrow \mathcal{A}^{\mathbf{PubProbGen}_{\mathrm{SK}}}(\mathrm{PK})$
$b \xleftarrow{\$} \{0, 1\}$
$(\sigma_{x_b}, \tau_{x_b}) \leftarrow \mathbf{ProbGen}_{\mathrm{SK}}(x_b)$
$\hat{b} \leftarrow \mathcal{A}(\sigma_{x_b})$
If $b = \hat{b}$ then returns 1 else 0

Fig. 8: Experiment for the Privacy property of a verifiable computation scheme.

Definition 8 (Privacy). *Let $\mathcal{VC}$ a verifiable-computation scheme, f a function, λ a security parameter and $(PK, SK) \leftarrow \boldsymbol{KeyGen}(f, \lambda)$. We state that a verifiable computation scheme $\mathcal{VC}$ is* private *if for every PPT adversary $\mathcal{A}$, we have:*

$$\boldsymbol{Adv}^{\boldsymbol{Priv}}_{\mathcal{A}}[\mathcal{VC}, f, \lambda] \leq negl(\lambda)$$

where $\boldsymbol{Adv}^{\boldsymbol{Priv}}_{\mathcal{A}}[\mathcal{VC}, f, \lambda] = |\Pr[\boldsymbol{Exp}^{\boldsymbol{Priv}}_{\mathcal{A}}[\mathcal{VC}, f, \lambda] = 1] - \frac{1}{2}|$ where $\boldsymbol{Exp}^{\boldsymbol{Priv}}_{\mathcal{A}}$ is described in Figure 8.

4 MKHE Verifiable-Computation Scheme

Our verifiable-computation scheme $\mathcal{VC}_{mk}$ relies on a garbled circuit, hence it requires a *Yao-Secure* symmetric encryption scheme as introduced in Section 2. In addition, our scheme relies on a semantically secure MKHE scheme. Our scheme $\mathcal{VC}_{mk}$ is composed of four algorithms $\mathbf{KeyGen}^{mk}$, $\mathbf{ProbGen}^{mk}$, $\mathbf{Compute}^{mk}$ and $\mathbf{Verify}^{mk}$.

$\mathbf{KeyGen}^{mk}$ algorithm, executed once by the client, takes as input the function f to be simulated and a security parameter λ. The $\mathbf{KeyGen}^{mk}$ algorithm is in charge of the construction of the garbled circuit simulating f, but also to generate the *function key pair* $(pk_f, sk_f) \leftarrow \mathbf{MKKeyGen}(\lambda)$. Finally, it outputs the public key PK containing the garbled circuit simulating f encrypted under pk_f, and the secret key SK containing both the mapping of the labels associated to the bit value for each wire and the secret key sk_f.

$\mathbf{ProbGen}^{mk}_{\mathrm{SK}}$ algorithm, executed by the client at every input, encodes the input x given by the client. The algorithm generates the *input key pair* $(pk_x, sk_x) \leftarrow \mathbf{MKKeyGen}(\lambda)$ and outputs $\{\sigma_x\}_{pk_x}$ the encoding of x, encrypted under pk_x.

$\mathbf{Compute}^{mk}_{\mathrm{PK}}$ algorithm, executed by the server, simulates f with the garbled circuit, over $\{\sigma_x\}_{pk_x}$ the encoding of x encrypted under pk_x. The $\mathbf{Compute}^{mk}_{\mathrm{PK}}$ algorithm assumes a circuit Γ able to perform the decryption of a garbled table. A gate $(g : w_a, w_b \rightarrow w_z)$ takes as an input two labels $k^x_{w_a}, k^y_{w_b}$ and γ_g the garbled table associated with g as defined in equation (1), and outputs the resulting label $k^{g(x,y)}_{w_z}$ corresponding to the bit $g(x, y)$ on the wire w_z. Using Γ, the server computes $\{\sigma_y\}_{pk_f, pk_x}$ based on the garbled circuit encrypted under pk_f and $\{\sigma_x\}_{pk_x}$ the encoding of the input x encrypted under pk_x.

Finally, $\mathbf{Verify}^{mk}_{\mathrm{SK}}$ algorithm, executed by the client, verifies the result computed by the server. Given $\{\sigma_y\}_{pk_f,pk_x}$ and the two secret keys sk_f and sk_x, the client computes σ_y and verifies that σ_y is a valid encoding using the secret key SK.

Our verifiable-computation scheme relies on MKHE scheme instead of a homomorphic encryption scheme as it is done in $\mathcal{VC}_{gen}$ (see Appendix A). Moreover, the distinction between $\mathcal{VC}_{gen}$ and $\mathcal{VC}_{mk}$ is that in $\mathcal{VC}_{gen}$, the circuit Γ takes an input the garbled circuit encrypted under pk a public key different at every new input. Hence, the drawback we want to address is that for each input, γ_g is encrypted using the public key pk, which changes at every new input. This implies that γ_g must be encrypted every time a new input is provided by the client. In our scheme $\mathcal{VC}_{mk}$, thanks to the *function key pair* (pk_f, sk_f), we encrypt the garbled tables only once. Therefore, while the function f does not change, we can reuse the encrypted garbled tables over multiple inputs.

Definition 9 (Definition of $\mathcal{VC}_{mk}$). *The verifiable-computation scheme $\mathcal{VC}_{mk}$ is defined by a tuple of four functions*
$(\boldsymbol{KeyGen}^{mk}, \boldsymbol{ProbGen}^{mk}_{SK}, \boldsymbol{Compute}^{mk}_{PK}, \boldsymbol{Verify}^{mk}_{SK})$ *where:*

- $\boldsymbol{KeyGen}^{mk}(f,\lambda) \rightarrow (PK, SK)$*:*
  1. *Compute $\Delta_f = (\mathcal{G}, \mathcal{W}, \mathcal{W}_{in}, \mathcal{W}_{out})$ the boolean circuit representing f.*
  2. *For each wire $w \in \mathcal{W}$, choose randomly two values $k^0_w, k^1_w \in_R \{0,1\}^\lambda$ representing respectively 0 and 1 on the wire w.*
  3. *Compute the garbled circuit $\gamma = \{\gamma_g | g \in \mathcal{G}\}$ where γ_g is computed as in equation (1).*
  4. *Generate $(pk_f, sk_f) \leftarrow \boldsymbol{MKKeyGen}(\lambda)$ the* function-keypair*.*
  5. *Compute $\gamma = \{\boldsymbol{MKEnc}_{pk_f}(\gamma_g) | g \in \mathcal{G}\}$ where γ_g is defined as in the Equation (1).*
  6. *Output $PK = \gamma$ and $SK = \cup_{w\in\mathcal{W}}\{k^0_w, k^1_w\} \cup \{sk_f\}$ the set of randomly chosen labels and the secret key related with the input.*
- $\boldsymbol{ProbGen}^{mk}_{SK,\lambda}(x) \rightarrow (\{\sigma_x\}_{pk_x}, \tau_x)$*:*
  Let the binary encoding $x_1, \ldots, x_n$ of input x.
  1. *Generate the* input-keypair *(pk_x, sk_x) using $\boldsymbol{MKKeyGen}(\lambda)$.*
  2. *Output $\{\sigma_x\}_{pk_x} = \{\boldsymbol{MKEnc}_{pk_x}(k^{x_1}_{w_1}), \ldots, \boldsymbol{MKEnc}_{pk_x}(k^{x_n}_{w_n})\}$ the set of labels associated to the wire regarding on the input bits encrypted under pk_x, and $\tau_x = SK \cup \{sk_x\}$.*
- $\boldsymbol{Compute}^{mk}_{PK}(\{\sigma_x\}_{pk_x}) \rightarrow \{\sigma_y\}_{pk_f,pk_x}$*:*
  Let Γ be the circuit such that given $k^x_{w_a}, k^y_{w_b}$ and γ_g outputs $k^{g(x,y)}_{w_z}$.
  1. *Compute $\boldsymbol{MKEval}(\Gamma, \{k^x_{w_a}\}_{pk_x}), \{k^y_{w_b}\}_{pk_x}, \{\gamma_g\}_{pk_f})$ successively for each gate $(g : w_a, w_b \rightarrow w_z)$, until to get $\{k^{y_i}_{w_i}\}_{pk_x,pk_f}$ the value of output wires $w_i \in \mathcal{W}_{out}$, encrypted under pk_x and pk_f.*
  2. *Output $\{\sigma_y\}_{pk_f,pk_x} \leftarrow \{\{k^{y_i}_{w_i}\}_{pk_f,pk_x} | w_i \in \mathcal{W}_{out}\}$.*
- $\boldsymbol{Verify}^{mk}_{SK}(\tau_x, \{\sigma_y\}_{pk_f,pk_x}) \rightarrow y \cup \perp$*:*
  1. *Given sk_f from SK and sk_x from τ_x, compute $\boldsymbol{MKDec}(\{k^{y_i}_{w_i}\}_{pk_f,pk_x}, \{sk_f, sk_x\})$ for each $w_i \in \mathcal{W}_{out}$.*
  2. *Output $y = y_1, \ldots, y_m$ if for all $i \in [1,m]$, $k^{y_i}_{w_i} \in \{k^0_{w_i}, k^1_{w_i}\}$, otherwise reject the result with $\perp$.*

5 Security Analysis

We prove that our scheme $\mathcal{VC}_{mk}$ is a *correct*, *secure*, *private* and *outsourceable* verifiable-computation scheme. For the proofs of these results, we use the following result proven in [LP08].

Theorem 1 ($\mathcal{VC}_{yao}$ is a verifiable-computation scheme). *[LP08] Let* E *a Yao-Secure symmetric encryption scheme. Then* $\mathcal{VC}_{yao}$ *is a correct, one-time secure, private and outsourceable computation scheme.*

We want to prove that our verifiable-computation scheme $\mathcal{VC}_{mk}$ is *correct*, meaning that an honest server can produce a valid result. We state the correctness theorem as follows:

Theorem 2 (Correctness of $\mathcal{VC}_{mk}$). *Let* E *a Yao-Secure symmetric encryption scheme and* M *a semantically secure* MKHE *scheme.* $\mathcal{VC}_{mk}$ *is a correct verifiable-computation scheme, hence we have:*

$$f(x) = \boldsymbol{Verify}_{SK}^{mk}(\boldsymbol{Compute}_{PK}^{mk}(\boldsymbol{ProbGen}_{SK}^{mk}(x)))$$

Proof. The proof relies on the correctness of the $\mathcal{VC}_{yao}$ stated in Theorem 1 and on the correctness of the MKHE scheme M used in $\mathcal{VC}_{mk}$. Since we assume a Yao-Secure symmetric encryption scheme, then by Theorem 1, $\mathcal{VC}_{yao}$ is a correct verifiable-computation scheme.

To prove that $\mathcal{VC}_{mk}$ is correct, we create a new scheme $\mathcal{VC}'_{mk}$ which works as the same as $\mathcal{VC}_{mk}$, except that we replace every homomorphic operation (*i.e.,* encryption, evaluation and decryption) by his equivalent in clear. This manipulation is possible since M is assumed to be correct. Hence, instead of performing computation in the ciphertext space, computations are realized in the plaintext space. In the protocol, the garbled circuit is no more encrypted by the client before to be sent to the server, as well as the labels σ_x of the input x. The server performs the execution of the garbled circuit over the plaintext labels to produce the output labels $\sigma_{f(x)}$ for a given function f, then sent back to the client able to verify the verification of the produced result. Clearly, we have $\mathcal{VC}'_{mk} = \mathcal{VC}_{yao}$. By assumption from Theorem 1, $\mathcal{VC}_{yao}$ is correct, therefore $\mathcal{VC}_{mk}$ is correct by reduction. □

To formalize the privacy of our verifiable-computation scheme $\mathcal{VC}_{mk}$, we consider the experiment in Fig. 8 presented in [GGP10] where a PPT adversary $\mathcal{A}$ sends two inputs x_0 and x_1 to the challenger. The challenger encodes the input x_b where b is a bit randomly chosen by the challenger. The challenger sends to adversary $\mathcal{A}$ the public encoding of σ_{x_b}. The adversary $\mathcal{A}$ predicts the selected input by outputting a bit $\hat{b}$. The output of the experiment is 1 if the adversary $\mathcal{A}$ successfully predicts the input chosen by the challenger *i.e.,* $\hat{b} = b$, 0 otherwise. Before to propose the two inputs x_0 and x_1 to the challenger, the adversary $\mathcal{A}$ is allowed to call a polynomial number of times (depending on the security parameter λ) the oracle $\mathbf{PubProbGen}_{\mathrm{SK}}(x)$ which for a given input x different from x_0 and x_1, computes $(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{\mathrm{SK}}^{yao}(x)$ and returns σ_x the public encoding of the input x.

Theorem 3. *Privacy of $\mathcal{VC}_{mk}$ Let* E *be a Yao-Secure symmetric encryption scheme and* M *be a semantically secure* MKHE *scheme. Then, $\mathcal{VC}_{mk}$ is a private verifiable-computation scheme, with respect to the definition 8.*

Proof. Theorem 3 says that given a function f and security parameter λ, there is no adversary $\mathcal{A}$ able to break the privacy of our scheme $\mathcal{VC}_{mk}$ with a non-negligible advantage, leading to have:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathbf{Priv}}[\mathcal{VC}_{mk}, f, \lambda] \leq \mathit{negl}(\lambda)$$

By contradiction, we assume that there is an adversary $\mathcal{A}$ able to break the privacy of $\mathcal{VC}_{mk}$. Hence, we have:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathbf{Priv}}[\mathcal{VC}_{mk}, f, \lambda] \geq \varepsilon$$

where ϵ is a non-negligible probability. Our goal is to construct an adversary $\mathcal{B}$ able to win against the privacy of the one-time verifiable-computation scheme $\mathcal{VC}_{yao}$ (on which our scheme is based) with a non-negligible advantage ε', leading to following relation:

$$\mathbf{Adv}_{\mathcal{B}}^{\mathbf{Priv}}[\mathcal{VC}_{yao}, f, \lambda] = \varepsilon' + \mathit{negl}(\lambda) \leq \mathbf{Adv}_{\mathcal{A}}^{\mathbf{Priv}}[\mathcal{VC}_{mk}, f, \lambda] = \varepsilon + \mathit{negl}(\lambda)$$

In the reduction we want to construct, the challenger $\mathcal{C}_{yao}$ is playing against a PPT adversary $\mathcal{B}$ which wants to break the privacy of $\mathcal{VC}_{yao}$ the one-time verifiable-computation scheme. The adversary $\mathcal{B}$ simulates the challenger $\mathcal{C}_{mk}$ challenging the adversary $\mathcal{A}$ against the privacy of our scheme $\mathcal{VC}_{mk}$. The adversary $\mathcal{A}$ is allowed to call a polynomial number of times the oracle $\mathbf{PubProbGen}_{\mathrm{SK}}$ simulated by $\mathcal{B}$, which given an input x, computes $(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{\mathrm{SK}}^{yao}(x)$, generates a new MKHE key pair (pk_x, sk_x) and outputs $\{\sigma_x\}_{pk_x}$ the public encoding of x encrypted under pk_x. The reduction works as follows:

1. The challenger $\mathcal{C}_{yao}$ computes $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathbf{KeyGen}^{yao}(\lambda)$ and actives the adversary $\mathcal{B}$ with PK. The adversary $\mathcal{B}$ generates a MKHE key pair $(pk_f, sk_f) \leftarrow \mathbf{MKKeyGen}(\lambda)$ and activates the adversary $\mathcal{A}$ with $\{\mathrm{PK}\}_{pk_f}$ the garbled circuit produced by $\mathcal{C}_{yao}$ encrypted under pk_f.
2. The adversary $\mathcal{A}$ calls the $\mathbf{PubProbGen}_{\mathrm{SK}}$ oracle simulated by $\mathcal{B}$, a number of times bounded by a polynomial depending on the security parameter λ. This oracle captures the fact that $\mathcal{A}$ has access to a polynomial number of encrypted encoding of inputs. This property is necessary since our protocol allows the server to reuse the same garbled circuits over different inputs.
3. Finally, the adversary $\mathcal{A}$ sends to the adversary $\mathcal{B}$ two inputs x_0 and x_1, forwarded to the challenger $\mathcal{C}_{yao}$. The challenger $\mathcal{C}_{yao}$ responds to $\mathcal{B}$ with σ_{x_b} the encoding of the input x_b where b is a randomly chosen bit. Then, $\mathcal{B}$ computes $(pk_{x_b}, sk_{x_b}) \leftarrow \mathbf{MKKeyGen}(\lambda)$ and sends $\{\sigma_{x_b}\}_{pk_{x_b}}$ to $\mathcal{A}$. The adversary $\mathcal{A}$ sends $\hat{b}$ a prediction over the bit b, to the adversary $\mathcal{B}$, directly forwarded to the challenger $\mathcal{C}_{yao}$.

By assumption, the adversary $\mathcal{A}$ has a non-negligible advantage ε to break the privacy of our scheme. As the adversary $\mathcal{B}$ sends to the challenger $\mathcal{C}_{yao}$ the prediction $\hat{b}$ over the bit b from $\mathcal{A}$, then the non-negligible advantage of $\mathcal{B}$ to win against the challenger $\mathcal{C}_{yao}$ is only based on the advantage of $\mathcal{A}$ to win against $\mathcal{C}_{mk}$, simulated by $\mathcal{B}$. Therefore, we have $\varepsilon = \varepsilon'$, leading to the relation:

$$\mathbf{Adv}^{\mathbf{Priv}}_{\mathcal{B}}[\mathcal{VC}_{yao}, f, \lambda] = \varepsilon' + \mathit{negl}(\lambda) = \mathbf{Adv}^{\mathbf{Priv}}_{\mathcal{A}}[\mathcal{VC}_{mk}, f, \lambda] = \varepsilon + \mathit{negl}(\lambda)$$
$$\Leftrightarrow \mathbf{Adv}^{\mathbf{Priv}}_{\mathcal{B}}[\mathcal{VC}_{yao}, f, \lambda] = \mathbf{Adv}^{\mathbf{Priv}}_{\mathcal{A}}[\mathcal{VC}_{mk}, f, \lambda] = \varepsilon + \mathit{negl}(\lambda)$$

By reduction, we have shown that assuming an adversary $\mathcal{A}$ able to break the privacy of our scheme $\mathcal{VC}_{mk}$ with a non-negligible advantage ε, it is possible to construct an adversary $\mathcal{B}$ able to break the privacy of $\mathcal{VC}_{yao}$ scheme with the same non-negligible advantage. This contradicts Theorem 1 which states that there is no PPT adversary able to break the privacy of $\mathcal{VC}_{yao}$. This concludes the proof. □

To formalize the security of our verifiable-computation scheme $\mathcal{VC}_{mk}$, we consider the experiment depicted in Fig. 7 presented in [GGP10], where a challenger $\mathcal{C}$ provides a $l = \text{poly}(\lambda)$ number of public encoding σ_{x_i} for an input x_i chosen by a PPT attacker $\mathcal{A}$. At the end, for an index $1 \leq i \leq l$, the attacker $\mathcal{A}$ tries to produce $\sigma_{\hat{y}}$ the encoding of an invalid result $\hat{y} \neq f(x)$ approved by the client through the **Verify** function (defined in $\mathcal{VC}$). The output of this experiment is 1 if the invalid encoding $\sigma_{\hat{y}}$ is accepted by the client, 0 otherwise.

We state the following theorem:

Theorem 4 (Security of $\mathcal{VC}_{mk}$). *Let* E *a Yao-Secure symmetric encryption scheme and* M *a semantically secure* MKHE *scheme. Then* $\mathcal{VC}_{mk}$ *verifiable-computation scheme is secure, with respect to Definition 7.*

Proof. Theorem 4 says that there is no PPT attacker $\mathcal{A}$ able to break the security of our scheme $\mathcal{VC}_{mk}$ with a non-negligible advantage leading to have:

$$\mathbf{Adv}^{\mathbf{Sec}}_{\mathcal{A}}[\mathcal{VC}_{mk}, f, \lambda] \leq \mathit{negl}(\lambda)$$

For the sake of contradiction, consider that there is a PPT attacker $\mathcal{A}$ able to break security of our scheme $\mathcal{VC}_{mk}$ with a non-negligible advantage ε. Hence, we have:

$$\mathbf{Adv}^{\mathbf{Sec}}_{\mathcal{A}}[\mathcal{VC}_{mk}, f, \lambda] \geq \varepsilon$$

The proof is divided into two parts:

1. We construct a reduction between $G_{\text{yao}} = \mathbf{Exp}^{\mathbf{Sec}}_{\mathcal{B}}[\mathcal{VC}_{yao}, f, \lambda]$ the game where the challenger $\mathcal{C}_{\text{yao}}$ challenges the adversary $\mathcal{B}$ in the experiment **Sec** for the $\mathcal{VC}_{yao}$ verifiable-computation scheme, and $G_{\text{mk}} = \mathbf{Exp}^{\mathbf{Sec}}_{\mathcal{A}}[\mathcal{VC}_{mk}, f, \lambda]$ the game where the challenger $\mathcal{C}_{\text{mk}}$ challenges the adversary $\mathcal{A}$ in the experiment **Sec** for the $\mathcal{VC}_{mk}$ verifiable-computation scheme. In G_{mk}, the adversary is allowed to call an oracle $\mathbf{PubProbGen}^{G_{\text{mk}}}_{\text{mk}}$ a polynomial number of times denoted $l = \text{poly}(\lambda)$ depending to security parameter λ.

2. Let G_{rand} the game working exactly as G_{mk}, except that the adversary has only access to random inputs. We perform an hybrid experimentation to show that the game G_{mk} is indistinguishable from the game G_{rand}. By H^k for some $k \in \{1, \ldots, l\}$, we denote the game G_{mk} where we modify the behavior of $\mathbf{PubProbGen}_{\text{mk}}^{G_{\text{mk}}}$. Our hybrid experiment is structured as follows:
   - We set $H^0 = G_{\text{mk}}$ and $H^l = G_{\text{rand}}$.
   - We prove that for $k \in \{0, ..., t-1\}$, H^{k-1} is indistinguishable from H^k by considering that the adversary does not obtain or lose information.
   - We prove that H^{t-1} is indistinguishable from H^t by considering the semantic security of the MKHE scheme. In particular, if a distinguisher is able to distinguish between H^{t-1} and H^t, then the distinguisher can also win against the indishtinguishability of MKHE scheme.
   - We prove that for $k \in \{t+1, ..., l\}$, we have $H^{k-1} = H^k$ by considering that the adversary does not obtain or lose information.

Reduction from G_{mk} to G_{yao}. The challenger $\mathcal{C}_{yao}$ is playing against a PPT adversary $\mathcal{B}$ which wants to break the security of the one-time verifiable-computation scheme $\mathcal{VC}_{yao}$. The adversary $\mathcal{B}$ has access to an oracle $\mathbf{PubProbGen}_{\text{yao}}$ a single time, which given an input x outputs σ_x the encoding of x, where σ_x is generated using the $\mathbf{ProbGen}^{yao}$ simulated by the challenger $\mathcal{C}_{yao}$. The adversary $\mathcal{B}$ simulates the challenger $\mathcal{C}_{mk}$ challenging the adversary $\mathcal{A}$ against the security of our scheme $\mathcal{VC}_{mk}$. The adversary $\mathcal{A}$ is allowed to call an oracle $\mathbf{PubProbGen}_{\text{mk}}$ a poly(λ) number of times where λ is the security parameter. The reduction works as follows:

1. The challenger $\mathcal{C}_{yao}$ computes $(\text{PK}, \text{SK}) \leftarrow \mathbf{KeyGen}^{yao}(\lambda)$ and activates the adversary $\mathcal{B}$ with PK. Then, $\mathcal{B}$ generates a MKHE key pair $(pk_f, sk_f) \leftarrow \mathbf{MKKeyGen}(\lambda)$ and activates the adversary $\mathcal{A}$ with $\{\text{PK}\}_{pk_f}$ the garbled circuit produced by $\mathcal{C}_{yao}$ encrypted under pk_f.
2. $\mathcal{B}$ randomly picks an index $t \xleftarrow{\$} [1, l]$.
3. For $j \in [1, l]$:
   (a) $\mathcal{B}$ computes $(pk_j, sk_j) \leftarrow \mathbf{MKKeyGen}(\lambda)$.
   (b) $\mathcal{A}$ sends an input x_j of his choice to $\mathcal{B}$.
   (c) $\mathcal{B}$ computes $\sigma_{x_j} \leftarrow \mathbf{PubProbGen}_{\text{mk}}(j, t, x_j)$.
   (d) $\mathcal{B}$ sends $\{x_j\}_{pk_{x_j}}$ to $\mathcal{A}$.
4. $\mathcal{A}$ sends the tuple $(i, \{\sigma_{\hat{y}}\}_{pk_f, pk_i})$ to $\mathcal{B}$, where $i \in [1, l]$ is chosen by $\mathcal{A}$.
5. $\mathcal{B}$ computes $\sigma_{\hat{y}} \leftarrow \mathbf{MKDec}(\{\sigma_{\hat{y}}\}_{pk_f, pk_i}, \{sk_f, sk_i\})$.
6. $\mathcal{B}$ sends $\sigma_{\hat{y}}$ to $\mathcal{C}_{yao}$.

The oracle $\mathbf{PubProbGen}_{\text{mk}}^{G_{\text{mk}}}$ works as follows:

Oracle $\mathbf{PubProbGen}_{\mathrm{mk}}^{G_{\mathrm{mk}}}$

Inputs from $\mathcal{A}$: An index $j \in \{1, \ldots, l\}$ and the input x_j
Inputs from $\mathcal{B}$: An index $t \in \{1, \ldots, l\}$
Output: The encoding σ_{x_j} of x_j
if $j \neq t$ **then**
  $r \xleftarrow{\$} \mathbb{F}_2^{\lambda}$ // a random input label
  return r
else
  $\sigma_{x_j} \leftarrow \mathbf{PubProbGen}_{\mathrm{yao}}(x_j)$ // the real input label
  return σ_{x_j}
end if

Hybrid experiments. We present the hybrid experiments where we want to show that the game G_{mk} is indistinguishable from the game G_{rand}. By H^k with $k \in \{1, \ldots, l\}$, we denote the game G_{mk} where we modify the behavior of the $\mathbf{PubProbGen}_{\mathrm{mk}}^{H^k}$ oracle as follows:

Oracle $\mathbf{PubProbGen}_{\mathrm{mk}}^{H^k}$

Inputs from $\mathcal{A}$: An index $j \in \{1, \ldots, l\}$ and the input x_j
Inputs from $\mathcal{B}$: An index $t \in \{1, \ldots, l\}$
Output: The encoding σ_{x_j} of x_j
if $j \neq t$ or $j < k$ **then**
  $r \xleftarrow{\$} \mathbb{F}_2^{\lambda}$ // a random input label
  return r
else
  $\sigma_{x_j} \leftarrow \mathbf{PubProbGen}_{\mathrm{yao}}(x_j)$
  return σ_{x_j}
end if

When $k = 0$, we have $H^0 = G_{\mathrm{mk}}$.

When $k \in \{1, \ldots, t-1\}$, the adversary $\mathcal{A}$ does not obtain more information, since the oracle $\mathbf{PubProbGen}_{\mathrm{mk}}^{H^k}$ in H^k remains unchanged. Hence, the adversary $\mathcal{A}$ has the same advantage to win against the challenger in G_{mk} and in the game H^k. Hence, we have $H^{k-1} = H^k$, leading to the relation:

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{H^0}[\mathcal{VC}_{mk}, f, \lambda] = 1] = \Pr[\mathbf{Exp}_{\mathcal{A}}^{H^{t-1}}[\mathcal{VC}_{mk}, f, \lambda] = 1]$$

When $k = t$, the oracle $\mathbf{PubProbGen}_{\mathrm{mk}}^{H^t}$ is modified, since the oracle no more produces valid encoding but only random labels. Hence, the adversary $\mathcal{A}$ loses some information. To prove that H^{t-1} (where $\mathbf{PubProbGen}_{\mathrm{mk}}^{H^{t-1}}$ remains unchanged) and H^t are distinguishable only with negligible probability, we perform a proof by contradiction where we assume that there is a distinguisher $\mathcal{D}$ able to distinguish between H^{t-1} and H^t with a non-negligible advantage p.

Consider a challenger $\mathcal{C}_{Ind}$ which given two messages m_0 and m_1 by the adversary, picks a random bit b and returns to the adversary $\{x_b\}_{pk}$ the message x_b encrypted under the MKHE public key pk.

From $\mathcal{D}$, we construct a PPT adversary $\mathcal{B}_{Ind}$ able to break the indishtinguishability of the semantically secure MKHE with a non-negligible advantage, against the challenger $\mathcal{C}_{Ind}$. The adversary $\mathcal{B}_{Ind}$ picks a $t \in \{1, \ldots, l\}$. For a query $j \in \{1, \ldots, l\}$, $\mathcal{D}$ sends an input x_j to the adversary $\mathcal{B}_{Ind}$. If $j \neq t$, then $\mathcal{B}_{Ind}$ computes a new MKHE key pair and encrypts a random λ-string which is sent back to the challenger. If $j = t$, then $\mathcal{B}_{Ind}$ sends to the challenger $\mathcal{C}_{Ind}$ the input x_j as message m_0 and a random λ-string label as m_1. The challenger $\mathcal{C}_{Ind}$ responds to $\mathcal{B}_{Ind}$ with $\{m_b\}_{pk}$ where b is a random bit and pk is a MKHE encryption key generated by $\mathcal{C}_{Ind}$. $\mathcal{B}_{Ind}$ transmits $\{m_b\}_{pk}$ to $\mathcal{D}$. Once all queries executed, $\mathcal{D}$ responds to $\mathcal{B}_{Ind}$ with a bit 0 if $\mathcal{D}$ predicts that the input sent by $\mathcal{B}_{Ind}$ corresponds to the game H^{t-1}, 1 for H^t. The bit sent $\mathcal{D}$ to $\mathcal{B}_{Ind}$ is forwarded to $\mathcal{C}_{Ind}$. We see that if $\mathcal{D}$ has a non-negligible advantage p to distinguish between H^{t-1} and H^t, then $\mathcal{B}_{Ind}$ has a non-negligible advantage p to break the indishtinguishability of the MKHE scheme, which contradicts our hypothesis. Therefore, H^{t-1} and H^t are indistinguishable, or at least with a negligible probability ε'. Hence, we have the relation:

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{H^{k-1}}[\mathcal{VC}_{mk}, f, \lambda] = 1] = \Pr[\mathbf{Exp}_{\mathcal{A}}^{H^{k}}[\mathcal{VC}_{mk}, f, \lambda] = 1] + \varepsilon'$$

When $k \in \{t+1, \ldots, l\}$, the game H^{k-1} is the same than H^t, since the adversary $\mathcal{A}$ does not obtain more information. Hence, we have $H^{k-1} = H^k$, leading to the relation:

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{H^{t+1}}[\mathcal{VC}_{mk}, f, \lambda] = 1] = \Pr[\mathbf{Exp}_{\mathcal{A}}^{H^{l}}[\mathcal{VC}_{mk}, f, \lambda] = 1]$$

Finally, when $k = l$, we have the game $H^l = G_{\text{rand}}$ where the adversary has only a negligible advantage to win against the challenger $\mathcal{C}_{mk}$ in game $H^l = G_{\text{rand}}$. To conclude our hybrid experiment, we show that we

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{H^{0}}[\mathcal{VC}_{mk}, f, \lambda] = 1] = \Pr[\mathbf{Exp}_{\mathcal{A}}^{H^{l}}[\mathcal{VC}_{mk}, f, \lambda] = 1] + \varepsilon'$$

Since $H^0 = G_{\text{mk}}$ and $H^l = G_{\text{rand}}$ we have

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{G_{\text{mk}}}[\mathcal{VC}_{mk}, f, \lambda] = 1] = \Pr[\mathbf{Exp}_{\mathcal{A}}^{G_{\text{rand}}}[\mathcal{VC}_{mk}, f, \lambda] = 1] + \varepsilon'. \quad (2)$$

And since the adversary $\mathcal{A}$ in G_{rand} can only cheat with a negligible probability, then $\Pr[\mathbf{Exp}_{\mathcal{A}}^{G_{\text{rand}}}[\mathcal{VC}_{mk}, f, \lambda] = 1]$ is negligible in the security parameter λ. Therefore, $\Pr[\mathbf{Exp}_{\mathcal{A}}^{G_{\text{mk}}}[\mathcal{VC}_{mk}, f, \lambda] = 1]$ is negligible by Equation (2). □

Theorem 5 (Outsourceability of $\mathcal{VC}_{mk}$). *Let* E *a Yao-Secure symmetric encryption scheme and* M *a semantically secure* MKHE *scheme. Then, $\mathcal{VC}_{mk}$ is an outsourceable verifiable-computation scheme.*

Proof. To prove the outsourceability of $\mathcal{VC}_{mk}$, we focus on an asymptotic complexity. The client starts the scheme with $\mathbf{KeyGen}^{mk}$ to generate the garbled

circuit of $\Delta_f = (\mathcal{G}, \mathcal{W}, \mathcal{W}_{\text{in}}, \mathcal{W}_{\text{out}})$ corresponding to a function f. The garbled circuit creation required to (1) generate two random labels for each wire and (2) encrypt each gate four times using the Yao-Secure symmetric encryption scheme then encrypt the four resulting ciphertexts, using pk_f, over the input x previously encoded and encrypted by the client under pk_x. The server computes the circuit Δ_f gate-by-gate until to obtain the encoding of $f(x)$ encrypted under both pk_x and pk_f, sent back to the client. Therefore, the complexity of $\mathbf{Compute}^{mk}$ depends on the number of gates $|\mathcal{G}|$ in Δ_f, leading to a complexity $O(|\mathcal{G}|)$. Finally, the client calls $\mathbf{Verify}^{mk}$ function to verify the computation performed by the server. In details, $\mathbf{Verify}^{mk}$ decrypts each encrypted label of $f(x)$ and verifies that the resulting labels are valid, using the secret key SK, leading us to a complexity of $O(|\mathcal{W}_{\text{out}}|)$. Note that $\mathbf{KeyGen}^{mk}$ is computed only once over multiple inputs. Therefore, we obtain an amortized complexity for the client of $O(|\mathcal{W}_{\text{in}}| + |\mathcal{W}_{\text{out}}|)$, which is strictly lower than $O(|\mathcal{G}|)$ corresponding to the complexity needed to compute the function f. Hence, $\mathcal{VC}_{mk}$ is outsourceable. □

Theorem 6. *Let* E *a yao-secure symmetric encryption and* M *a semantically secure* MKHE *scheme. Then,* $\mathcal{VC}_{mk}$ *is a correct, secure, private and outsourceable verifiable-computation scheme.*

Proof. The proof is a simple consequence of Theorem 2, Theorem 4, Theorem 3, and Theorem 5. □

6 Performance Comparison

As presented in Fig. 10, our scheme $\mathcal{VC}_{mk}$ requires stricly less operations than $\mathcal{VC}_{gen}$, thanks to the MKHE scheme. Let a function f represented a boolean circuit $\Delta_f = (\mathcal{G}, \mathcal{W}, \mathcal{W}_{\text{in}}, \mathcal{W}_{\text{out}})$. In details, $\mathcal{VC}_{mk}$ relies on two distinct key pairs, called *function-key pair* denoted (pk_f, sk_f) and *input-key pair* denoted (pk_x, sk_x). The public key pk_f is used during $\mathbf{KeyGen}^{mk}$, to compute $\gamma = \{\mathbf{MKEnc}_{pk_f}(\gamma_g)\}$ the encrypted garbled circuit where each garbled table γ_g for some gate $g \in \mathcal{G}$ has been encrypted under pk_f. As Δ_f does not depend on some input, we can compute γ only once and reuse it for many inputs, as depicted in Fig. 9. Compared to $\mathcal{VC}_{gen}$, we remove $|\mathcal{G}|$ homomorphic encryption, which is significant, especially for large function.

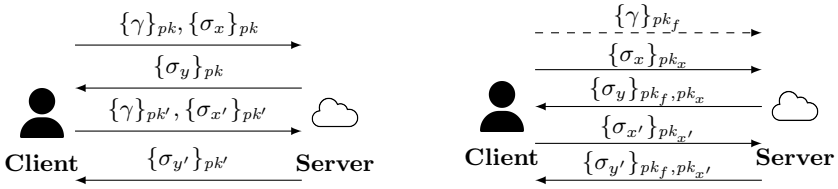

Fig. 9: At the left, the original verifiable-computation scheme $\mathcal{VC}_{gen}$. At the right, our verifiable-computation scheme $\mathcal{VC}_{mk}$.

Functions	$\mathcal{VC}_{gen}$	$\mathcal{VC}_{mk}$
KeyGen Executed *once* by the client.	• $2\|\mathcal{W}\|$ labels generation • $4\|\mathcal{G}\|$ encryptions with E	• $2\|\mathcal{W}\|$ labels generation • $4\|\mathcal{G}\|$ encryptions with E • $\|\mathcal{G}\|$ encryptions with M (under pk_f)
ProbGen Executed by the client.	• $\|\mathcal{W}_{\text{in}}\|$ encryption with S (under pk)	• $\|\mathcal{W}_{\text{in}}\|$ encryptions with M (under pk_x)
Compute Executed by the server.	• $\|\mathcal{G}\|$ encryptions with S (under pk) • $\|\mathcal{G}\|$ homomorphic evaluations with S	• $\|\mathcal{G}\|$ homomorphic evaluations with M
Verify Executed by the client.	• $\|\mathcal{W}_{\text{out}}\|$ decryptions with S (under sk)	• $\|\mathcal{W}_{\text{out}}\|$ decryptions with M (under sk_x and sk_f)

Fig. 10: Number of sub-routines calls performed in $\mathcal{VC}_{gen}$ compared to our scheme $\mathcal{VC}_{mk}$, given a circuit $\Delta_f = (\mathcal{G}, \mathcal{W}, \mathcal{W}_{\text{in}}, \mathcal{W}_{\text{out}})$. We denote by E a Yao-Secure symmetric encryption scheme, S is a semantically secure homomorphic encryption scheme and M a semantically secure MKHE scheme. We assume in $\mathcal{VC}_{gen}$ a key pair (pk, sk). As well, we assume in $\mathcal{VC}_{mk}$ the *function-keypair* (pk_f, sk_f) and the *input-keypair* (pk_x, sk_x).

A direct consequence is that in the function $\mathbf{Compute}^{mk}$, the server is already ready to evaluate $\Delta_f = (\mathcal{G}, \mathcal{W}, \mathcal{W}_{\text{in}}, \mathcal{W}_{\text{out}})$, since every input *i.e.,* the garbled circuit γ and σ_x the encoding of input x, has been already encrypted using the MKHE. This significantly diverges from $\mathcal{VC}_{gen}$ where the server still needs to encrypt the garbled circuit γ at every new input, under the public key pk. Hence, in our scheme $\mathcal{VC}_{mk}$, we remove the necessity to encrypt each gate $g \in \mathcal{G}$. Hence, we reduce the number of encryption by $|\mathcal{G}|$, leading to a complexity of $|\mathcal{G}|$ in our scheme $\mathcal{VC}_{mk}$ instead of $2|\mathcal{G}|$ in $\mathcal{VC}_{gen}$.

We stress that once $\mathbf{KeyGen}^{mk}$ has been computed, we do not require more computation for the client in both $\mathbf{ProbGen}^{mk}$ and $\mathbf{Verify}^{mk}$, compared to the scheme $\mathcal{VC}_{gen}$. Indeed, the client deals respectively with the input pre-processing and output post-processing. In details, the $\mathbf{ProbGen}^{mk}$ computes the encryption of the encoding of the input, consisting of the calculation of a $|\mathcal{W}_{\text{in}}|$ sized set of encrypted labels, which is also done in $\mathcal{VC}_{gen}$. The $\mathbf{Verify}^{mk}$, however, is slightly different since we require two secret keys instead of one secret key, explained by the fact that we rely on a different homomorphic scheme, but the idea does not change. First, each encrypted label is decrypted using the secret keys sk_x and sk_f, which differ from $\mathcal{VC}_{gen}$ where the decryption is performed using secret key sk. Once the decryption is performed, each output label $k_{w_i}^{y_i}$ is compared with the valid labels $k_{w_i}^0, k_{w_i}^1$. This verification does not differ from $\mathcal{VC}_{gen}$ except for the used homomorphic encryption scheme.

To replace the homomorphic encryption scheme defined in $\mathcal{VC}_{gen}$ with a MKHE scheme does not impact significantly the performance of our scheme

in practice. Indeed, MKHE schemes [CCS19, CDKS19] proposed a reasonable execution time overhead.

7 Conclusion

We presented $\mathcal{VC}_{mk}$ a verifiable-computation scheme proven to be *correct*, *secure*, *private* and *outsourceable*. Our scheme $\mathcal{VC}_{mk}$ is based on MKHE scheme, allowing to prepare some ciphertexts during an initialization, reused for every new inputs. Compared to the state-of-the-art of garbled circuit with homomorphic encryption, we have significantly reduced the computations required to outsource a computation on an untrusted server, with the same security guarantees.

References

AJJ20. Prabhanjan Vijendra Ananth, Abhishek Jain, and Zhengzhong Jin. Multiparty homomorphic encryption. 2020.

AJJM20. Prabhanjan Vijendra Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In *TCC*, 2020.

BP16. Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key fhe with short ciphertexts. In *Proceedings, Part I, of the 36th Annual International Cryptology Conference on Advances in Cryptology — CRYPTO 2016 - Volume 9814*, page 190–213, Berlin, Heidelberg, 2016. Springer-Verlag.

CCS19. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from tfhe. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 446–472, Cham, 2019. Springer International Publishing.

CDKS19. Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 395–412, New York, NY, USA, 2019. Association for Computing Machinery.

Cha04. D. Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security Privacy*, 2(1):38–47, 2004.

CRS05. David Chaum, Peter Y. A. Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In *Proceedings of the 10th European Conference on Research in Computer Security*, ESORICS'05, page 118–139, Berlin, Heidelberg, 2005. Springer-Verlag.

CZW17. Long Chen, Zhenfeng Zhang, and Xueqing Wang. Batched multi-hop multi-key fhe from ring-lwe with compact ciphertext extension. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 597–627, Cham, 2017. Springer International Publishing.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

GGP10. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 465–482, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

GKR15. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), sep 2015.

HKE13. Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. volume 8043 LNCS, 2013.

HKK+14. Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. Amortizing garbled circuits. volume 8617 LNCS, 2014.

Kil92. Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 723–732, New York, NY, USA, 1992. Association for Computing Machinery.

Lin16. Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29, 2016.

LP08. Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22:161–188, 2008.

LP15. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. *Journal of Cryptology*, 28, 2015.

Mic00. Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30:1253–1298, 2000.

PHCP13. Ki-Woong Park, Jaesun Han, JaeWoong Chung, and Kyu Ho Park. Themis: A mutually verifiable billing system for the cloud computing environment. *IEEE Transactions on Services Computing*, 6(3):300–313, 2013.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

A Original Verifiable-Computation scheme

We present the verifiable-computation scheme $\mathcal{VC}_{gen}$ presented in [GGP10], a garbled-circuit based verifiable-computation scheme improved with single-key homomorphic encryption scheme. Intuitively, this scheme follows the Yao verifiable-computation scheme $\mathcal{VC}_{yao}$, at the difference that the double decryptions to obtain a label for a gate's output is performed homomorphically. Let E a Yao-Secure symmetric encryption scheme and
$\mathsf{S} = (\mathbf{HEKeyGen}, \mathbf{HEEnc}, \mathbf{HEEval}, \mathbf{HEDec})$ a semantically secure homomorphic encryption scheme.

Definition 10. *Let* $\mathcal{VC}_{gen} = (\boldsymbol{KeyGen}^{gen}, \boldsymbol{ProbGen}^{gen}, \boldsymbol{Compute}^{gen}, \boldsymbol{Verify}^{gen})$ *a verifiable-computation scheme where:*

- $\boldsymbol{KeyGen}^{gen}(f, \lambda) \rightarrow (PK, SK)$*:*
  1. *Compute* $\Delta_f = (\mathcal{G}, \mathcal{W}, \mathcal{W}_{in}, \mathcal{W}_{out})$ *the boolean circuit representing* f*.*
  2. *For each wire* $w \in \mathcal{W}$*, choose randomly two labels* $k_w^0, k_w^1 \in_R \{0,1\}^\lambda$ *representing respectively 0 and 1 on the wire* w*.*
  3. *Compute the garbled circuit* $\gamma = \{\gamma_g | g \in \mathcal{G}\}$ *where* γ_g *is computed as in equation* (1)*.*
  4. *Output* $PK = \gamma$ *and* $SK = \cup_{w \in \mathcal{W}} \{k_w^0, k_w^1\}$ *the set of randomly chosen labels.*
- $\boldsymbol{ProbGen}_{SK}^{gen}(x) \rightarrow (\sigma_x, \tau_x)$*:*
  Let the binary encoding $x_1, \ldots, x_n$ *of input* x*.*
  1. *Generate the public key pair* $(pk, sk) \leftarrow \boldsymbol{HEKeyGen}(\lambda)$*.*
  2. *Outputs* $\sigma_x = \{\boldsymbol{HEEnc}_{pk}(k_{w_1}^{x_1}), \ldots, \boldsymbol{HEEnc}_{pk}(k_{w_n}^{x_n})\}$ *the set of labels associated to the wire regarding on the input bits encrypted under* pk*, and* $\tau_x = SK \cup \{sk\}$*.*
- $\boldsymbol{Compute}_{PK}^{gen}(\sigma_x) \rightarrow \sigma_y$*:*
  Let Γ *the circuit such that given* $k_{w_a}^x, k_{w_b}^y$ *and* γ_g *outputs* $k_{w_z}^{g(x,y)}$*.*
  1. *Compute* $\boldsymbol{HEEval}(\Gamma, \{k_{w_a}^x\}_{pk}), \{k_{w_b}^y\}_{pk}, \boldsymbol{HEEnc}_{pk}(\gamma_g))$ *successively for each gate* $(g : w_a, w_b \rightarrow w_z)$*, until to get* $\{k_{w_i}^{y_i}\}_{pk}$ *the value of output wires* $w_i \in \mathcal{W}_{out}$*, encrypted under* pk*.*
  2. *Outputs* $\sigma_y \leftarrow \{\{k_{w_i}^{y_i}\}_{pk} | w_i \in \mathcal{W}_{out}\}$*.*
- $\boldsymbol{Verify}_{SK}^{gen}(\tau_x, \sigma_y) \rightarrow y \cup \perp$*:*
  Let sk *from* τ_x*.*
  1. *Decrypt* $\{\{k_{w_i}^{y_i}\}_{pk} | w_i \in \mathcal{W}_{out}\}$*.*
  2. *Output* $y = y_1, \ldots, y_m$ *if for all* $i \in [1, m]$*,* $k_{w_i}^{y_i} \in \{k_{w_i}^0, k_{w_i}^1\}$*, otherwise the server is cheating, therefore we refuse the result with* $\perp$*.*