ABSTRACT

A homomorphic encryption (HE) scheme is an advanced encryption technology which allows any user receiving ciphertexts to perform computations over them in a public manner. An important application of an HE scheme is a private delegating computation where clients encrypt their secret data, send the ciphertexts to a (computationally powerful) server who perform computations over encrypted data. In this application, one of the crucial problems is that the delegated server might be not trusted one and in this case, we cannot believe that a server always returns correct computation results. To solve this problem, Lai et al. (ESORICS 2014) proposed a verifiable homomorphic encryption (VHE) as a core primitive realizing private and verifiable secure delegating computation. However, their VHE scheme only supports homomorphic evaluation over ciphertexts generated by a single user. In this paper, we propose a formalization and its construction of multi-key verifiable homomorphic encryption (MVHE), which is a new cryptographic primitive for realizing private and verifiable delegated computation in the multi-client setting. Our construction can be obtained by combining a multi-key homomorphic encryption scheme and a multi-key homomorphic encrypted authentication scheme, which is also a new primitive provided in this work.

INDEX TERMS

Homomorphic encryption, secure delegating computation, verifiability.

I. INTRODUCTION

A homomorphic encryption (HE) scheme is a novel and powerful cryptographic primitive having various applications. This primitive allows anyone to perform computations over encrypted data with the data kept secret from it. One of the most practically important applications of an HE scheme is private delegating computations (over clouds). Consider a situation where there are a client who has only a relatively weak computational device and a server who has more powerful computational resources. In this situation, we assume that a client wants to securely outsource his data to the (possibly malicious) remote server (for getting some computation result over his data), while allowing it to reliably perform computations over the data. Utilizing an HE scheme, we can easily realize this demand (as we call private delegating computations). Concretely, a client generates a ciphertext of his secret data and sends it to a server. Then, the server get a ciphertext of the computation result by performing (possibly heavy) homomorphic evaluations on the client’s ciphertext and return it to the client. Finally, the client decrypts the evaluated ciphertext and get the computation result.\(^\text{1}\) Obviously, as an important requirement on efficiency, the computation cost for a client should be independent from the size of a delegated function.

While this cryptographic protocol is already attractive, it has a significant concern regarding the integrity of computation results by a server. More precisely, there may be an incentive for a server trying to cheat and return an incorrect computation result to the client. This incentive might be

\(^{1}\text{Note that a more advanced situation where a client wants computations requiring multiple inputs (by this single user) also can be solved by using an HE scheme.}\)
occurred due to the nature of the performed computation (such as, the case that the server wants to convince the client of a particular result when it will give the server some benefits) or just the server's laziness for heavy computations. This problem motivates us to consider a more desirable protocol called private and verifiable delegating computation enabling a client to verify the given computation result is correct as well as get the result.\(^2\) As a natural core cryptographic primitive for obtaining private and verifiable delegating computation, Lai et al.\(^{[20]}\) proposed a verifiable homomorphic encryption (VHE) scheme. As the name suggests, a VHE scheme allows a user to not only decrypt the computation result but also check it is the correct evaluated value. Obviously, if given an VHE scheme, we can obtain private and verifiable delegating computation immediately.

At first glance, one might think that the above private and verifiable delegating computation is sufficiently enough for practical applications. However, when taking into account of the real-world setting, we can see that it is not true. Specifically, it is important for us to consider private and verifiable delegating computations in the multi-client setting, which enables secure delegating computations on data by multiple users who have their own respective secret keys.\(^3\) Actually, when considering some practically important applications (such as, big data analysis for medical cares), it is inherently required to support computations on data given multiple users and it is not reasonable to assume that all of the users share only one key (used both for encryption and decryption procedures). Unfortunately, almost of the previous private and verifiable delegating computations support only computations on data given by a single user. Especially, the existing Lai et al.'s VHE scheme\(^{[20]}\) only supports homomorphic evaluation on the ciphertexts generated using the same secret key.

### A. OUR CONTRIBUTION

In this paper, as a core cryptographic primitive for private and verifiable delegating computations in the multi-user setting, we propose multi-key verifiable homomorphic encryption (MVHE). Roughly, an MVHE scheme is a primitive has following features:\(^4\)

- Each user \(U_i\) (having its secret key \(sk_i\)) to generate a ciphertext \(c_i\) of its message \(m_i\).
- Receiving a function \(f\) (for a homomorphic evaluation) and users' ciphertexts \((c_i)_{i \in [n]}\), anyone can perform homomorphic evaluations and get an evaluated ciphertext \(c_f\) for the value \(y = f(m_1, \ldots, m_n)\).
- By using all of the secret keys \((sk_i)_{i \in [n]}\), we can decrypt the evaluated ciphertext \(c_f\) and verify that the evaluated value \(y\) is correct as well as get the result.

In order to obtain an MVHE scheme, we also propose a new notion of multi-key homomorphic encrypted authentication (MHEA), which is a multi-key variant of homomorphic encrypted authentication (HEA) introduced by Lai et al.\(^{[20]}\) as a building block.\(^5\) \(^6\) We show that an MVHE scheme can be constructed by combining a multi-key homomorphic encryption (MHE) scheme\(^{[21]}\) and an MHEA scheme. As we will see below, MVHE plays an important role to realize private and verifiable delegating computations in the multi-client setting.

We explain how to obtain a private and verifiable delegating computation in the multi-user setting by utilizing an MVHE scheme MVHE (with a multi-party computation (MPC) protocol with semi-honest security\(^7\)). For clarity and concreteness, we assume the situation where there are many clients (with only weak computational devices) who want to get a (shared) prediction model, while keeping all their training data secret, and there is one (malicious) server with an enough computational power for generating the prediction model. More precisely, we consider a situation where there are multiple clients \((C_i)_{i \in [n]}\) and a (computationally powerful) server \(S\) and each client \(C_i\) has its (secret) training data \(\text{data}_i\), and they hope to get an efficient prediction model \(f(\text{data}_1, \ldots, \text{data}_n)\) computed from a (public) learning program \(f\) and their training data \((\text{data}_i)_{i \in [n]}\). Here, we assume that all of the clients \((C_i)_{i \in [n]}\) are semi-honest and the server \(S\) is malicious, where "semi-honest" means that the (possibly corrupted) clients follow the specified protocol and nothing is leaked in the transcript. As a preparation phase, each client generates their own secret key \(sk_i\) of MVHE, computes a ciphertext \(c_i\) of \(\text{data}_i\) using \(sk_i\) by itself, and sends the ciphertext \(c_i\) to \(S\).\(^8\) Then, given all users' ciphertexts \((c_i)_{i \in [n]}\), the server \(S\) performs the evaluation algorithm of MVHE on a (public) function \(f\) and ciphertexts \((c_i)_{i \in [n]}\) for obtaining a ciphertext \(c_f\) encrypting \(f(\text{data}_1, \ldots, \text{data}_n)\) and returns to the users. Upon receiving an evaluated ciphertext \(c_f\) from \(S\), the users run the decryption algorithm of MVHE (hardwired the evaluated ciphertext \(c_f\)) on a semi-honest MPC protocol given the secret keys \((sk_i)_{i \in [n]}\) as secret inputs. We note that\(^9\)

\(^2\)This protocol is firstly treated formally by Gennaro, Gentry, and Parno\(^{[15]}\) and it is sometimes called just verifiable computation.

\(^3\)As a related work, Choi, Katz, Kumaresen, and Cid\(^{[7]}\) proposed a cryptographic primitive called a multi-client VC scheme which allows multiple clients to outsource the (heavy) computation depending on their inputs to a server. In Section 1-B, we will compare this primitive and our private and verifiable delegating computations in the multi-client setting.

\(^4\)Due to its nature, in a (M)MVHE scheme, a user uses a secret key both for encryption and decryption procedures.

\(^5\)Note that, when referring (multi-key) homomorphic (encrypted) authentication, we only consider one requiring a secret key in the verification process, that is, the message authentication code setting.

\(^6\)We can see that MHEA is also an extension of multi-key homomorphic authentication (MHA), which was introduced in\(^{[11]}\), with an additional property. More precisely, MHEA is MHA having a privacy requirement that ensuring that the information of an authenticated message is not revealed from the corresponding authentication. In this sense, MHEA is a stronger primitive than MHA and our construction of MVHE requires MHEA essentially for ensuring its privacy.

\(^7\)We note that various MPC protocols with semi-honest security can be constructed based solely on secret sharing schemes\(^{[4]}\),\(^{[17]}\).

\(^8\)While in an actual MVHE scheme, in addition to secret keys, each user generates an (public) evaluation key \(ek\) used by a server \(S\) for executing homomorphic evaluations including a ciphertext generated on \(sk_i\), we omit these evaluation keys for simplifying explanations.
We briefly recall some previous related works on (fully) MHE, (fully) MHA, and private and verifiable delegating computation.

1) MULTI-KEY HOMOMORPHIC ENCRYPTION
López-Alt et al. [21] proposed the notion of MHE and its concrete construction based on the NTRU lattice. Clear and McGoldrick [8] proposed a general transformation from (fully) HE into (fully) MHE based on the learning with errors (LWE) assumption over lattices. As a result, they obtain the first MHE scheme based on the LWE assumption. Mukherjee and Wichs [23] presented a new construction of MHE having a single round threshold decryption process based on the LWE assumption by simplifying the Clear et al.’s MHE scheme. Peikert and Shiehian [25] proposed a notion of multi-hop MHE, which enables that homomorphic evaluated ciphertexts can be reused in the following homomorphic evaluations involving additional users, and its construction based on the LWE assumption (with a restriction that the number of users are limited). Brakerski and Perlman [5] proposed a similar notion called fully dynamic MHE, which is different from multi-hop MHE in that the number of users are not a-priori bounded at the setup phase, and its construction based on the LWE assumption. Chen et al. [9] proposed the first multi-hop MHE scheme based on the ring LWE assumption. Recently, Ananth et al. [1] proposed the first MHE scheme in the plain model (which does not require any trusted setup phase) based on the LWE, ring LWE, and decisional small polynomial ratio assumption.

2) MULTI-KEY HOMOMORPHIC AUTHENTICATION
(M)HA is divided into two types depending on whether a verifier’s is secret or not. In the former case, an MHA is also referred as multi-key homomorphic message authentication code (MHMAC), while in the latter case referred as multi-key homomorphic signature (MHS). Gennaro and Wichs [19] proposed the first fully HMAC scheme based on a (fully) HE scheme (and pseudorandom functions). Then, Gorbunov et al. [18] proposed the first fully HS scheme based on the short integer solution (SIS) assumption. By extending the Gorbunov et al.’s HS scheme, Fiore et al. [11] proposed the first fully MHS scheme based on the SIS assumption. Lai et al. [22] introduced a new security notion called unforgeability under insider corruptions for MHS and showed that an MHS scheme satisfying such a strong security notion can be constructed from zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARK) [2], [12]. Fiore and Pagin [14] proposed a compiler called Matrioska which transforms any (single-key) HA scheme for polynomial-sized circuits into a fully MHA scheme.

3) PRIVATE AND VERIFIABLE DELEGATING COMPUTATION
Gennaro et al. [15] firstly considered the formal notion of private and verifiable delegating computation (called verifiable computation (VC) scheme for short) and proposed its construction based on a fully HE scheme and garbled circuits. Then, Parno et al. [24] (resp., Goldwasser et al. [16]) proposed the first VC scheme based on an attribute-based encryption scheme (resp., a succinct single-key functional encryption scheme). Lai et al. [20] proposed a VHE scheme as a building block for VC. Fiore et al. [10] proposed a generic construction of VC based on a fully HE scheme and a (non-private) VC scheme and its practically efficient instantiation. Recently, Fiore et al. [13] extended the approach of [10] to support public verifiability and the evaluation of more than quadratic functions. Moreover, Bois et al. [3] proposed an improved protocol which solved the restriction on the modulus of the underlying HE scheme used in [13].

Finally, as one of the most important related works, Choi et al. [7] proposed a cryptographic protocol called a multi-client VC scheme which allows multiple clients to outsource the computation of a function f of their inputs to a server. They construct a multi-client VC scheme by combining a Gennaro et al.’s (single-client) VC scheme [15] and a proxy oblivious transfer protocol. Although their multi-client VC scheme is similar to our private and verifiable delegating computation in the multi-user setting, we have important differences between these. Firstly, before running the delegated computation, their multi-client VC scheme requires one client to execute a heavy preprocessing computation depending on the size of a function f as a setup phase. Compared to theirs, our protocol does not require any client to execute such a heavy setup phase. Secondly, while their multi-client VC scheme does not need any interaction among clients, our protocol needs to interact each other when running the decryption procedure since we execute a semi-honest MPC protocol. Compared to each other, both of them have (dis)advantages, we can say that these two notions are incomparable. 10

A. BASIC NOTATIONS
In this paper, \( x \leftarrow X \) denotes sampling an element \( x \) from a finite set \( X \) uniformly at random. \( y \leftarrow \mathcal{A}(x; r) \) denotes that

---

9We note that since a (standard) MPC protocol does not have verifiability, we cannot realize private and verifiable delegating computations solely based on a MPC protocol.

10One might see that these two primitives are same one except that they have different advantages.
a probabilistic algorithm \( A \) outputs \( y \) for an input \( x \) using a randomness \( r \), and we simply denote \( y \leftarrow A(x) \) when we need not write an internal randomness explicitly. Also, \( x := y \) denotes that \( x \) is defined by \( y \). \( \lambda \) denotes a security parameter. A function \( f(\lambda) \) is a negligible function in \( \lambda \), if \( f(\lambda) \) tends to 0 faster than \( \frac{1}{\lambda} \) for every constant \( c > 0 \). \( \negl(\lambda) \) denotes an unspecified negligible function. PPT stands for probabilistic polynomial time. \( \emptyset \) denotes an empty set. 

### Table 1. Notations.

| Notation  | Meaning |
|-----------|---------|
| \( x \leftarrow X \) | An element \( x \) is sampled from a finite set \( X \) uniformly at random |
| \( y \leftarrow A(x; r) \) | A (PPT) algorithm \( A \) outputs \( y \) for an input \( x \) using a randomness \( r \) |
| \( y \leftarrow \mathcal{A}(x) \) | The notation \( y \leftarrow \mathcal{A}(x; r) \) when \( r \) is not needed to written explicitly |
| \( x := y \) | A string \( x \) is defined by a string \( y \) |
| \( \lambda \) | A security parameter |
| \( \negl(\lambda) \) | A negligible function in \( \lambda \) |
| \( PPT \) | Probabilistic Polynomial Time |
| \( \emptyset \) | An empty set |
| \( [n] \) | For \( n \in \mathbb{N} \) a set of integers \( \{1, \ldots, n\} \) |
| \( [a, b] \) | For \( a, b \in \mathbb{Z} \), \( a \leq b \), a set of integers \( \{a, \ldots, b\} \) |
| \( f \) | The number of inputs for a labeled program |
| \( M \) | A message space |
| \( \mathcal{I} \) | An identity space |
| \( \mathcal{F} \) | A function family \( \mathcal{F} := \{ f : M^t \rightarrow M \} \) |
| \( f_{id} \) | An identity function \( f_{id} : M \rightarrow M \) |
| \( l \) | A label \( l = (id, r) \) in \( \mathcal{I} \times \mathcal{F} \) (for an input of a function \( f \)) |
| \( \Delta \) | A dataset identifier |
| \( P = (f, l_1, \ldots, l_t) \) | A labeled program \( P \) consists of a function \( f \) on \( t \) variables and labels \( l_1, \ldots, l_t \) for the inputs |
| \( l_i = (f_{id}, l) \) | An identity program with a label \( l \) |
| \( P^* = (P_1, \ldots, P_t) \) | A composed program \( P^* \) (which is derived by evaluating a function \( g : M^t \rightarrow M \) on the outputs of \( P_1, \ldots, P_t \)) |
| \( \text{EKS} \) | A set of evaluation keys |

### B. MULTI-KEY HOMOMORPHIC ENCRYPTION

In this section, we review the definition of multi-key homomorphic encryption (MHE) and its IND-CPA security. In the following, for a polynomial \( t = t(\lambda) \), sets \( M \) and \( \mathcal{I} \), and a function family \( \mathcal{F} := \{ f : M^t \rightarrow M \} \), we define a program \( P = (f, id_1, \ldots, id_t) \) as a tuple of a function \( f \) and \( t \) identities \( id_1, \ldots, id_t \in \mathcal{I} \).

**Definition 1 (Multi-Key Homomorphic Encryption):** A MHE scheme \( \text{MHE} \) with a plaintext space \( M \) and an identity space \( \mathcal{I} \) consists of the following PPT algorithms.

**MHE.Setup:** The setup algorithm, given a security parameter \( t^1 \), outputs a public parameter \( pp \). This parameter includes the description of an identity label space \( \mathcal{I} \), a message space \( M \), and a set of admissible functions \( \mathcal{F} \). (The public parameter \( pp \) is an input to all of the following algorithms, even when not specified.)

**MHE.KG:** The key generation algorithm, given a public parameter \( pp \) and an identity \( id \), outputs an evaluation key \( ek_{id} \) and a secret key \( sk_{id} \).\(^{11}\)

**MHE.Enc:** The encryption algorithm, given a secret key \( sk \) and a message \( m \in M \), outputs a ciphertext \( c \).

**MHE.Dec:** The decryption algorithm, given a program \( P \), a set of secret keys \( \{sk_{id}\}_{id \in \mathcal{P}} \), and a ciphertext \( c \), outputs a message \( m \in M \) when \( id \in \mathcal{I} \).

**MHE.Eval:** The homomorphic evaluation algorithm, given a function \( f : M^t \rightarrow M \) and a set \( \{ci, \text{EKS}\}_{i \in [t]} \), outputs an evaluated ciphertext \( cf \), where each \( \text{EKS} \) is a set of evaluation keys.

As the basic properties for MHE, we require that MHE satisfies ordinary correctness, evaluation correctness, and succinctness.

#### 1) ORDINARY CORRECTNESS

We say that an MHE scheme \( \text{MHE} \) satisfies ordinary correctness if

\[
\text{Pr}[\text{MHE.Enc}(\mathcal{I}_i, sk_{id}, \text{MHE.Enc}(sk_{id}, m)) \neq m] = \negl(\lambda)
\]

holds, where \( \lambda \in \mathbb{N} \), \( pp \leftarrow \text{MHE.Setup}(t^1) \), \( m \in M \), \( id \in \mathcal{I} \), \( (ek_{id}, sk_{id}) \leftarrow \text{MHE.KG}(pp, id) \), and \( \mathcal{I}_i = (f_{id}, id) \) is an identity program.

#### 2) EVALUATION CORRECTNESS

Let \( \lambda \in \mathbb{N} \), \( t = t(\lambda) \) be some polynomial, \( pp \leftarrow \text{MHE.Setup}(t^1) \), a set of tuples of (honest) evaluation keys and secret keys \( \{ek_{id}, sk_{id}\}_{id \in \mathcal{I}} \) for some \( \mathcal{I} \subseteq \mathcal{I} \), and any set of triples of programs, plaintexts, and ciphertexts \( \{(P_{i}, m_{i}, c_{i})\}_{i \in [t]} \) such that \( m_{i} = \text{MHE.Dec}(P_{i}, (sk_{id})_{id \in \mathcal{I}}, c_{i}) \) holds for all \( i \in [t] \). Let \( f : M^t \rightarrow M \) be a function, \( m^* = (m_1, \ldots, m_t) \), \( P^* = (P_1, \ldots, P_t) \), and \( c^* \leftarrow \text{MHE.Eval}(f, \{ci, \text{EKS}\}_{i \in [t]} \) where \( \text{EKS}_{i} = (ek_{id})_{id \in P_{i}} \). We say that an MHE scheme \( \text{MHE} \) satisfies evaluation correctness if we have

\[
\text{Pr}[\text{MHE.Dec}(P^*, (sk_{id})_{id \in P^*}, c^*) \neq m^*] = \negl(\lambda).
\]

\(^{11}\)Without loss of generality, when we do not need to write an identity \( id \) explicitly, we simply denote \((pk, sk) \leftarrow \text{MHE.KG}(pp)\).
Next, we will recall IND-CPA security for MHE.

Definition 2 (IND-CPA Security): Let MHE be an MHE scheme. Let \( n := n(\lambda) \) be a polynomial in \( \lambda \). Consider the following game between a challenger \( C \) and an adversary \( A \).

1) \( C \) chooses a challenge bit \( b \leftarrow \{0,1\} \) and generates a public parameter \( pp \leftarrow \text{MHE.Setup}(^1\lambda) \) and keys \( (ek_i, sk_i) \leftarrow \text{MHE.KG}(pp) \) for all \( i \in [n] \). Then, \( C \) sends \( pp \) and \( (ek_1, \ldots, ek_n) \) to \( A \) and prepares lists \( L_{ch} := \emptyset \) and \( L_{corr} := \emptyset \).

2) \( A \) can make polynomially many challenge queries and corruption queries as follows:
   - **Challenge Queries.** When \( C \) receives a challenge query \((i,m_0,m_1)\) \( (|m_0| = |m_1|) \), it checks whether \( i \in L_{corr} \) holds. If this is the case, then \( C \) returns \( \bot \) to \( A \). Otherwise, \( C \) computes \( c \leftarrow \text{MHE.Enc}(sk_i,m_0) \), sends \( c \) to \( A \), and appends \( i \) to \( L_{ch} \).
   - **Corruption Queries.** When \( C \) receives a corruption query \( j \), it checks whether \( j \in L_{ch} \) holds. If this is the case, then \( C \) returns \( \bot \) to \( A \). Otherwise, \( C \) sends \( sk_j \) to \( A \) and appends \( j \) to \( L_{corr} \).

3) \( A \) outputs a bit \( b' \in \{0,1\} \).
   
   In this game, we define the advantage of the adversary \( A \) as
   \[
   \text{Adv}_{\text{MHE},A}(\lambda) := 2 \cdot \left| \Pr[b = b'] - \frac{1}{2} \right|.
   \]
   We say that MHE satisfies IND-CPA security if for any PPT adversary \( A \), \( \text{Adv}_{\text{MHE},A}(\lambda) = \negl(\lambda) \) holds.

III. MULTI-KEY HOMOMORPHIC ENCRYPTED AUTHENTICATION

In this section, as a building block for our MVHE scheme, we provide a new cryptographic primitive called multi-key homomorphic encrypted authentication (MHEA). One can see that an MHEA scheme is an extension of a multi-key homomorphic authentication (MHA) scheme given by Fiore et al. \[11\]. Roughly, a MHA scheme enables users \( (U_i)_{i \in [n]} \) (having their own (distinct) secret keys \( (sk_i)_{i \in [n]} \)) to generate authentications \( \sigma_m \) for (possibly different) messages \( m_i \). Then, based on the original authentications \( (\sigma_{m_i})_{i \in [n]} \), anybody (who does not know secret keys) can homomorphically perform an arbitrary program \( \mathcal{P} \) over the authenticated messages \( (m_i)_{i \in [n]} \) to generate a new short authentication \( \sigma_{\mathcal{P}(m_1, \ldots, m_n)} \) for the message \( \mathcal{P}(m_1, \ldots, m_n) \), which ensures the validity of \( y = \mathcal{P}(m_1, \ldots, m_n) \) as the output of \( \mathcal{P} \). In addition to this property of an MHE scheme, an MHEA scheme has a privacy requirement ensuring that the information of an authenticated message is not revealed from the corresponding authentication.

In the following, in Section III-A, we firstly provide the syntax of MHEA. Then, in Section III-B, we give the security notions for MHEA, unforgeability and privacy. Finally, in Section III-C, we show a candidate of instantiation of MHEA. Although the syntax and unforgeability of MHEA is the same as one of MHA given by Fiore et al. \[11\], we provide their formal descriptions for completeness.

A. SYNTAX

In this section, we provide the syntax of MHEA. (For convenience, in Table 1, we give a table of our notations used in Section III.) Firstly, we recall the definition of labeled programs given in \[19\].

Definition 3 (Labeled Programs): Let \( f : \mathcal{M}^t \rightarrow \mathcal{M} \) be a function on \( t \) variables and \( l_i \in \{0,1\}^* \) for \( i \in [t] \) the label of the \( i \)-th input of \( f \). A labeled program \( \mathcal{P} \) is defined as a tuple \( (f, l_1, \ldots, l_t) \). A labeled program can be composed as follows. Given some labeled programs \( \mathcal{P}_1, \ldots, \mathcal{P}_t \) and a function \( g : \mathcal{M}^t \rightarrow \mathcal{M} \), the composed program \( \mathcal{P}^* \) is obtained by evaluating \( g \) on the outputs of \( \mathcal{P}_1, \ldots, \mathcal{P}_t \), and it is denoted as \( \mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t) \). (The labeled inputs with the same label are grouped together and considered as a unique input of \( \mathcal{P}^* \).) Let \( f_{\mathcal{I}_t} : \mathcal{M} \rightarrow \mathcal{M} \) be the identity function and \( l_i \in \{0,1\}^* \) be any label. We refer to \( \mathcal{I}_l = (f_{\mathcal{I}_t}, l) \) as an identity program with a label \( l \). Note that a program \( \mathcal{P} = (f, l_1, \ldots, l_t) \) can be expressed as the composition of \( t \) identity programs \( f(\mathcal{I}_{l_1}, \ldots, \mathcal{I}_{l_t}) \).

Depending on the definition of labeled programs, we can provide the syntax and its authentication correctness, evaluation correctness, and succinctness of MHEA as follows.

Definition 4 (Multi-Key Homomorphic Encrypted Authentication): An MHEA scheme MHEA consists of the following five PPT algorithms.

MHEA.Setup: The setup algorithm, given the security parameter \( ^1\lambda \), outputs a public parameter \( pp \). This parameter consists of a description of a tag space \( \mathcal{T} \), an identity space \( \mathcal{ID} \), a message space \( \mathcal{M} \), and a set of admissible functions \( \mathcal{F} \). Given \( \mathcal{T} \) and \( \mathcal{ID} \), a label space is defined as \( \mathcal{L} := \mathcal{ID} \times \mathcal{T} \). For a labeled program \( \mathcal{P} = (f, l_1, \ldots, l_t) \) with labels \( l_i := (id_i, \tau_i) \in \mathcal{L} \) for \( i \in [t] \), we use \( id \in \mathcal{P} \) as a compact notation for \( id \in \{id_1, \ldots, id_t\} \). The public parameter \( pp \) is an input to all of the following algorithms, even when not specified.

MHEA.KG: The key generation algorithm, given a public parameter \( pp \) and an identity \( id \), outputs a public evaluation key \( ek_{id} \) and a secret authentication key \( sk_{id} \).

MHEA.Auth: The authentication algorithm, given an authentication key \( sk_{id} \), a dataset identifier \( \Delta \), a message \( m \), a label \( \ell = (id, \tau) \), outputs an authentication \( \sigma_{\ell} \).

MHEA.Eval: The evaluation algorithm, given a \( n \)-input function \( f : \mathcal{M}^t \rightarrow \mathcal{M} \) and a set \( \{(\sigma_i, \text{EKS}_i)\}_{i \in [t]} \) where each \( \sigma_i \) is an authentication and each \( \text{EKS}_i \) is a set of evaluation keys, outputs an evaluated authentication \( \sigma^* \).

MHEA.Ver: The verification algorithm, given a labeled program \( \mathcal{P} = (f, l_1, \ldots, l_t) \), a dataset identifier \( \Delta \), a set of authentication keys \( \{sk_{id}\}_{id \in \mathcal{P}} \)
corresponding to the identities involved in the program \( P \), a message \( m \) and an authentication \( \sigma \), outputs 1 (meaning “accept”) or 0 (meaning “reject”).

1) AUTHENTICATION CORRECTNESS

We say that an MHEA scheme MHEA satisfies authentication correctness if we have

\[
\Pr[\text{MHEA.Ver}(I_i, \Delta, sk_{id}, m, \sigma) = 0] = \negl(\lambda),
\]

where \( pp \leftarrow \text{MHEA.Setup}(\lambda^2) \), \( id \in ID, \tau \in T, l = (id, \tau) \in L, m \in M, f_{id} \in F, I_i := (f_{id}, l) \), \( (ek_{id}, sk_{id}) \leftarrow \text{MHEA.KG}(pp, id) \), and \( \sigma \leftarrow \text{MHEA.Auth}(sk_{id}, \Delta, l, m) \).

2) EVALUATION CORRECTNESS

Intuitively, we say that an MHEA scheme MHEA satisfies evaluation correctness when we run the evaluation algorithm on signatures \((\sigma_1, \ldots, \sigma_t)\) such that each \( \sigma_i \) verifies for \( m_i \) as the output of a labeled program \( P_l \) over a dataset \( \Delta \), the output signature \( \sigma \) verifying for the message \( f(m_1, \ldots, m_t) \) which is the output of the composed program \( f(P_1, \ldots, P_t) \) over the dataset \( \Delta \).

More formally, let \( t := t(\lambda) \) be some polynomial in \( \lambda \), a public parameter \( pp \) from \( \text{MHEA.Setup}(\lambda^2) \), a set of key pairs \( \{(ek_{id}, sk_{id})\}_{id \in ID} \) for some \( \overline{ID} \subseteq ID \), a dataset \( \Delta \), a function \( g : M' \rightarrow M \in F \), and any set \( \{P_i, m_i, \sigma_i\}_{i \in [t]} \) such that \( \text{MHEA.Ver}(P_i, \Delta, \{sk_{id}\}_{id \in P_i}, m_i, \sigma_i) = 1 \) for all \( i \in [t] \). Let \( m^* = (m_1, \ldots, m_t) \), \( P^* = (P_1, \ldots, P_t) \), and \( \sigma^* \leftarrow \text{MHEA.Eval}(f, \{(\sigma_i, \text{EKS})\}_{i \in [t]}) \) where \( \text{EKS} = \{ek_{id}\}_{id \in P} \). We say that MHEA is evaluation correct if \( \Pr[\text{MHEA.Ver}(P^*, \Delta, \{sk_{id}\}_{id \in P^*}, m^*, \sigma^*) = 0] = \negl(\lambda) \) holds.

3) SUCCINCTNESS

Intuitively, an MHEA scheme MHEA is succinct if the size of every authentication depends only logarithmically on the size of a dataset. Here, we allow authentication to depend on the number of keys involved in the computation.

More formally, let \( pp \leftarrow \text{MHEA.Setup}(\lambda^2) \), \( P = (f, l_1^*, \ldots, l_t^*) \) with \( l_i = (id_i, \tau_i) \) for \( i \in [t] \), \( (ek_{id}, sk_{id}) \leftarrow \text{MHEA.KG}(pp, id) \) for all \( id \in P \), and \( \sigma_i \leftarrow \text{MHEA.Auth}(sk_{id}, \Delta, l, m_i) \) for all \( i \in [t] \). We say that MHEA is succinct if there is a fixed polynomial poly such that \( |\sigma^*| = \text{poly}(\lambda, q, \log t) \) where \( \sigma^* \leftarrow \text{MHEA.Eval}(g, \{(\sigma_i, \text{EKS})\}_{i \in [t]}) \) and \( q = |id \in P| \).

B. SECURITY DEFINITIONS

In this section, we provide the definitions of unforgeability and privacy for MHEA.

Definition 5 (Unforgeability for MHEA): Let MHEA be an MHEA scheme. We define the unforgeability game between a challenger C and an adversary A as follows.

1) C generates a public parameter \( pp \leftarrow \text{MHEA.Setup}(\lambda^2) \), sends \( pp \) to \( A \), and prepares lists \( L_{key} := \emptyset \) and \( L_{corr} := \emptyset \).

2) A may adaptively make polynomially many authentication queries, verification queries, and corruption queries.

Authentication Queries. When \( C \) receives \( (\Delta, l, m) \), where \( \Delta \) is a dataset identifier, \( l = (id, \tau) \) is a label in \( ID \times T \) and \( m \in M \), it answers as follows:

a) If \( (\Delta, l, m) \) is the first query for the dataset \( \Delta \), \( C \) initializes an empty list \( L_{\Delta} := \emptyset \) and proceeds as follows.

b) If \( (id, \cdot) \notin L_{key} \) holds, \( C \) generates keys \( (ek_{id}, sk_{id}) \leftarrow \text{MHEA.KG}(pp, id) \) (that are implicitly assigned to identity \( id \)), gives \( ek_{id} \) to \( A \), and updates the list \( L_{key} := L_{key} \cup \{(id, sk_{id})\} \).

c) If \( (\Delta, l, m) \) is such that \( (l, m) \notin L_{\Delta} \), \( C \) computes \( \sigma_l \leftarrow \text{MHEA.Auth}(sk_{id}, \Delta, l, m) \) (note that \( C \) has already generated keys for the identity \( id \)), returns \( \sigma_l \) to \( A \), and updates the list \( L_{\Delta} \leftarrow L_{\Delta} \cup \{(l, m)\} \).

d) If \( (\Delta, l, m) \) is such that \( (l, \cdot) \notin L_{\Delta} \) (which means that the adversary had already made a query \( (\Delta, l, m') \) for some message \( m' \)), \( C \) ignores the query.

Verification Queries. When \( C \) receives \( (P, \Delta, m, \sigma) \), it answers as follows:

a) If \( (id, \cdot) \notin L_{key} \) holds for some \( id \in P \), then \( C \) generates \( (ek_{id}, sk_{id}) \leftarrow \text{MHEA.KG}(pp, id) \) and updates the list \( L_{key} := L_{key} \cup \{(id, sk_{id})\} \).

b) \( C \) returns \( \nu \leftarrow \text{MHEA.Ver}(P, \Delta, \{sk_{id}\}_{id \in P}, m, \sigma) \) to \( A \) (using the keys in \( L_{key} \)).

Corruption Queries. When \( C \) receives an identity \( id \), it answers as follows:

a) If \( (id, \cdot) \notin L_{key} \) holds, then \( C \) generates \( (ek_{id}, sk_{id}) \leftarrow \text{MHEA.KG}(pp, id) \) and updates the list \( L_{key} := L_{key} \cup \{(id, sk_{id})\} \).

b) \( C \) returns \( sk_{id} \) to \( A \) (stored in \( L_{key} \)) and updates \( L_{corr} := L_{corr} \cup \{id\} \).

3) \( A \) outputs a tuple \( (P^*, \Delta^*, m^*, \sigma^*) \), where \( P^* = (f^*, l_1^*, \ldots, l_t^*) \). \( C \) outputs 1 if the tuple returned by \( A \) satisfies the conditions of an event \textbf{Forge} (defined below), and 0 otherwise.

In the above game, we say that an event \textbf{Forge} occurs if \( \text{MHEA.Ver}(P^*, \Delta^*, \{sk_{id}\}_{id \in P^*}, m^*, \sigma^*) = 1 \) for all \( id \in P^* \), \( id \notin L_{corr} \), and either one of the following conditions holds:

Type 1: \( L_{\Delta^*} \) has not been initialized during the game (i.e., the dataset \( \Delta^* \) was never queried).

Type 2: For all \( i \in [t] \), \( \exists (l_i^*, m_i) \in L_{\Delta^*}, \) but \( m_i \neq f^*(m_1, \ldots, m_t) \) (i.e., \( m_i \) is not the correct output of \( P^* \) when executed over previously authenticated messages.)

Type 3: There exists a label \( l_i^* \) such that \( (l_i^*, \cdot) \notin L_{\Delta^*} \) for some \( i \in [t] \). (i.e., \( A \) never made a query with the label \( l_i^* \).)
We say that an MHEA scheme MHEA satisfies unforgeability if for any PPT adversary \( \mathcal{A} \),

\[
\text{Adv}_{\text{MHEA}, \mathcal{A}}^{\text{unf}}(\lambda) := \text{Pr}[\text{Forge}] = \text{negl}(\lambda)
\]

holds.

Definition 6 (Privacy for MHEA): Let MHEA be an MHEA scheme. We define the privacy game between a challenger \( \mathcal{C} \) and an adversary \( \mathcal{A} \) as follows.

1. \( \mathcal{C} \) chooses a challenge bit \( b \) \( \leftarrow \{0, 1\} \), runs MHEA.Setup(\( \lambda \)) to obtain a public parameter \( pp \), sends \( pp \) to \( \mathcal{A} \), and prepares lists \( L_{\text{key}} := \emptyset \), \( L_{\text{ch}} := \emptyset \), and \( L_{\text{corr}} := \emptyset \).

2. \( \mathcal{A} \) can polynomially many challenge queries and corruption queries as follows:

    **Challenge Queries.** When \( \mathcal{C} \) receives a triple \( (\Delta, l, m_0, m_1) \), where \( \Delta \) is a dataset identifier, \( l = (id, \tau) \) is a label in \( TD \times T \), and \( m \in \mathcal{M} \), it proceeds as follows:
    a) If \( (\Delta, l, m_0, m_1) \) is the first query for the dataset \( \Delta \), \( \mathcal{C} \) initializes an empty list \( L_\Delta = \emptyset \) and proceeds as follows.
    b) If \( (\Delta, l, m_0, m_1) \) is the first query with identity \( id \) (that is, \( (id, \cdot) \not\in L_{\text{corr}} \)), \( \mathcal{C} \) generates \( (ek_{id}, sk_{id}) \leftarrow \text{MHEA.KG}(pp, id) \) (that are implicitly assigned to the identity \( id \)), gives \( ek_{id} \) to \( \mathcal{A} \), appends \( (id, sk_{id}) \) to \( L_{\text{key}} \), and proceeds as follows.
    c) If \( (\Delta, l, m_0, m_1) \) is such that \( (l, m) \not\in L_\Delta \), \( \mathcal{C} \) checks whether \( id \in L_{\text{corr}} \) holds. If this is the case, then \( \mathcal{C} \) returns \( \bot \) to \( \mathcal{A} \). Otherwise, \( \mathcal{C} \) computes \( \sigma_i \leftarrow \text{MHEA.Auth}(sk_{id}, \Delta, l, m) \) (note that \( \mathcal{C} \) has already generated keys for the identity \( id \)), gives \( \sigma_i \) to \( \mathcal{A} \), and updates the lists \( L_\Delta \leftarrow L_\Delta \cup \{(l, m)\} \) and \( L_{ch} \leftarrow L_{ch} \cup \{id\} \).
    d) If \( (\Delta, l, m_0, m_1) \) is such that \( (l, \cdot) \in L_\Delta \) (which means that \( \mathcal{A} \) had already made a query \( (\Delta, l', m') \) for some plaintext \( m' \)), \( \mathcal{C} \) ignores the query.

    **Corruption Queries.** When \( \mathcal{C} \) receives a corruption query \( id \), it answers as follows:
    a) \( \mathcal{C} \) checks whether \( id \in L_{ch} \) holds. If this is the case, then \( \mathcal{C} \) returns \( \bot \) to \( \mathcal{A} \).
    b) \( \mathcal{C} \) generates keys \( (ek_{id}, sk_{id}) \leftarrow \text{MHEA.KG}(pp, id) \), gives \( sk_{id} \) to \( \mathcal{A} \), and appends \( (id, sk_{id}) \) to \( L_{\text{key}} \) and \( id \) to \( L_{\text{corr}} \).

3. \( \mathcal{A} \) outputs a bit \( b' \in \{0, 1\} \).

In the above game, we define the advantage of the adversary \( \mathcal{A} \) as

\[
\text{Adv}_{\text{MHEA}, \mathcal{A}}^{\text{priv}}(\lambda) := 2 \cdot \left| \text{Pr}[b = b'] - \frac{1}{2} \right|.
\]

We say that MHEA satisfies privacy if for any PPT adversary \( \mathcal{A} \), \( \text{Adv}_{\text{MHEA}, \mathcal{A}}^{\text{priv}}(\lambda) = \text{negl}(\lambda) \) holds.

C. Instantiation

In this section, we show a candidate of instantiation of MHEA. An instantiation of MHEA can be obtained by applying the compiler “Matrioska” [14], which turns any (sufficiently expressive) single-key homomorphic authentication (HA) into a multi-key one, to the existing single-key fully HA scheme proposed by Gennaro and Wichs [19]. In the following, we explain about this instantiation.

As shown by Lai et al. [20], Gennaro et al.’s HA scheme satisfies privacy in the above sense, that is, their HA scheme is already a HEA scheme due to its construction. More precisely, their scheme is constructed based on a (single-key) fully homomorphic encryption (FHE) scheme and a pseudorandom function (PRF) \( F \). Roughly, an authentication \( \sigma \) on a message \( m \) and a label \( r \) generated in their scheme consists of \( (c_1, \ldots, c_i, \nu) \), where \( c_i \) is a FHE ciphertext of the message \( m \) for \( i \in [\lambda] \) and \( \nu = F(\tau) \) is a value independent of \( m \). Obviously, if the underlying FHE scheme satisfies standard IND-CPA security, then each \( c_i \) does not leak any information of \( m \), that is, their scheme satisfies privacy.

We note that, as mentioned in [19], the complexity of the verification algorithm of this HA scheme requires at least that of executing the evaluated program \( P \). That is, if we apply this HA scheme to verifiable delegating computation without any efficiency improvement on the verification complexity, clients are required to perform some heavy computation which is larger than the size of \( P \). This is not preferable to the setting of verifiable delegating computation. Gennaro et al. showed as a solution that their construction can be extended to an HA scheme whose verification complexity is independent of the size of \( P \) by utilizing succinct non-interactive arguments for polynomial-time deterministic computations (SNARGs for \( P \)). Roughly, using the fact that the heavy computation (included in the verification step) is independent of the (authenticated) message, their trick is to delegate even the above heavy computation to the server as a homomorphic evaluation based on SNARGs for \( P \). From the recent work [6], SNARGs for \( P \) can be constructed based on the learning with errors (LWE) assumption over lattices. Thus, we can say that this problem on efficiency is solved reasonably.

By applying the compiler Matrioska to the above (single-key) HEA scheme, we obtain the first MHEA scheme for all polynomial-sized circuits. In fact, due to its mechanism, Matrioska inherits the privacy of a single-key HEA scheme to the (obtained) MHEA scheme. We note that, due to the restriction due to the underlying single-key HEA scheme and Matrioska, our MHEA scheme (i) is resilient to only fixed a-priori bounded number of verification queries and (ii) supports only constant number of users. These restrictions on our MHEA scheme are inherited to our MVHE scheme given in Section IV. We leave as an interesting open problem to examine how to construct a fully secure HEA scheme for all polynomial-sized circuits overcoming these restrictions.
IV. MULTI-KEY VERIFIABLE HOMOMORPHIC ENCRYPTION

In this section, we introduce our target cryptographic primitive called multi-key verifiable homomorphic encryption (MVHE). Roughly, an MVHE scheme is a multi-key variant of verifiable homomorphic encryption (VHE). An MVHE scheme allows each user $U_i$ (in $[n]$) to generate a ciphertext $c_i$ of its message $m_i$. Then, receiving a polynomial-sized function $f$ and their ciphertexts $(c_i)_{1 \leq i \leq n}$, anyone can perform homomorphic evaluations and get an evaluated ciphertext $c_f$ encrypting the value $y = f(m_1, \ldots, m_n)$. By using all of the secret keys $(sk_i)_{1 \leq i \leq n}$, we can decrypt the evaluated ciphertext $c_f$ and verify that the evaluated value $y$ is correct as well as get the value $y$.

In the following, in Section IV-A, we provide a formal syntax and security definitions (privacy and unforgeability) of MVHE. Then, in Section IV-B, we give our construction of MVHE based on an MHE scheme and an MHEA scheme. Finally, in Section IV-C, we provide the security proofs for our MVHE scheme.

A. FORMALIZATION

In this section, we provide the formal definition of MVHE (based on the definition of labeled programs). Firstly, we give the syntax of MVHE. (For convenience, in Table 1, we give a table of our notations used in Section IV.)

Definition 7 (Multi-Key Verifiable Homomorphic Encryption): An MVHE scheme MVHE consists of the following PPT algorithms.

MVHE.Setup: The setup algorithm, given the security parameter $1^\lambda$, outputs a public parameter $pp$. This parameter consists of a description of a tag space $\mathcal{T}$, an identity space $\mathcal{I}$, a plaintext space $\mathcal{M}$, and a set of admissible functions $\mathcal{F}$. Given $\mathcal{I}$ and $\mathcal{D}$, a label space is defined as $\mathcal{L} := \mathcal{I} \times \mathcal{T}$. For a labeled program $P = (f, l_1, \ldots, l_t)$ with labels $l_i := (id_i, \tau_i)$ for any $i \in [t]$, we use $id \in \mathcal{P}$ as compact notation for $id \in \{id_1, \ldots, id_t\}$. The public parameter $pp$ is an input to all of the following algorithms, even when not specified.

MVHE.KG: The key generation algorithm, given a public parameter $pp$ and an identity id, outputs an evaluation key $ek_{id}$ and a secret key $sk_{id}$.

MVHE.Enc: The encryption algorithm, given a secret key $sk$, a dataset identifier $\Delta$, a label $l = (id, \tau)$, and a message $m \in \mathcal{M}$, outputs a ciphertext $c$.

MVHE.Dec: The decryption algorithm, given a labeled program $P$, a dataset identifier $\Delta$, a set of secret keys $(sk_{id})_{id \in \mathcal{P}}$, and a ciphertext $c$, outputs a message $m \in \mathcal{M} \cup \{\bot\}$.

12Without loss of generality, when we do not need to write an identity id explicitly, we simply denote $(ek, sk) \leftarrow MVHE.KG(pp)$. Moreover, we assume that $ek$ can be computed from $sk$ efficiently.

MVHE.Eval: The homomorphic evaluation algorithm, given a function $f : \mathcal{M}^t \rightarrow \mathcal{M}$ and a set $\{(c_i, EK_i)\}_{i \in [t]}$, outputs an evaluated ciphertext $c_f$ where each $EK_i$ is a set of evaluation keys.

As the basic properties for MVHE, we require that MVHE satisfies ordinary correctness, evaluation correctness, and succinctness.

1) ORDINARY CORRECTNESS

We say that an MVHE scheme MVHE satisfies ordinary correctness if

$$\Pr[MVHE.Dec(\mathcal{I}_l, \Delta, sk_{id}) \neq m] = negl(\lambda)$$

holds, where $\lambda \in \mathbb{N}$, $pp \leftarrow MVHE.Setup(1^\lambda)$, $m \in \mathcal{M}$, $id \in \mathcal{I}$, $\tau \in \mathcal{T}$, $l = (id, \tau)$, $\Delta$, $(ek_{id}, sk_{id}) \leftarrow MVHE.KG(pp, id)$, and $\mathcal{I}_l = (f_{id}, l)$ is an identity program.

2) EVALUATION CORRECTNESS

Let $\lambda \in \mathbb{N}$, $t := t(\lambda)$ be some polynomial in $\lambda$, $pp \leftarrow MVHE.Setup(1^\lambda)$, a set of tuples of (honest) evaluation keys and secret keys $\{(ek_{id}, sk_{id})_{id \in \mathcal{I}}\}_{i \in [t]}$ holds for some $\mathcal{I} \subseteq \mathcal{I}$, and any set of triples of programs, plaintexts, and ciphertexts $\{(P_i, m_i, c_i)_{id \in \mathcal{P}}\}_{i \in [t]}$ such that $m_i = MVHE.Dec(P_i, \Delta, (sk_{id})_{id \in \mathcal{P}}, c_i)$ holds for all $i \in [t]$. Then, $f : \mathcal{M}^t \rightarrow \mathcal{M}$ be some function, $m^* = f(m_1, \ldots, m_t)$, $P^* = f(P_1, \ldots, P_t)$, and $c^* \leftarrow MVHE.Eval(f, \{(c_i, EK_i)\}_{i \in [t]})$ where each $EK_i = (ek_{id})_{id \in \mathcal{P}}$. We say that an MVHE scheme MVHE satisfies evaluation correctness if

$$\Pr[MVHE.Dec(P^*, \Delta, (sk_{id})_{id \in \mathcal{P}^*}, c^*) \neq m^*] = negl(\lambda)$$

holds.

3) SUCCINCTNESS

Let $t := t(\lambda)$ be some polynomial in $\lambda$, $pp \leftarrow MVHE.Setup(1^\lambda)$, $P = (f, l_1, \ldots, l_t)$ with $l_i = (id_i, \tau_i)$ for all $i \in [t]$ and $(ek_{id}, sk_{id}) \leftarrow MVHE.KG(pp, id)$ for any $id \in \mathcal{P}$. Let $m_i \in \mathcal{M}$ and $c_i \leftarrow MVHE.Enc(\Delta, l, m_i)$ for all $i \in [t]$. We say that MVHE satisfies succinctness if there exists a fixed polynomial $poly$ such that $|c| = poly(\lambda, q, \log t)$ where $q = |(id \in \mathcal{P})|$ and $c \leftarrow MVHE.Eval(f, \{(c_i, EK_{id_i})\}_{id \in [t]})$. (We note that, similarly to the succinctness of MHEA, the size of all ciphertexts depend only logarithmically on the size of a dataset but linearly on the number of keys involved in the evaluation.)

Next, we provide the security definitions of MVHE: privacy and unforgeability.

Definition 8 (Privacy for MVHE): Let MVHE be an MVHE scheme. We define the privacy game between a challenger $C$ and an adversary $A$.

1) $C$ chooses a challenge bit $b \leftarrow \{0, 1\}$, generates a public parameter $pp \leftarrow MVHE.Setup(1^\lambda)$, gives $pp$
to A, and prepares lists $L_{\text{key}} := \emptyset$, $L_{\text{ch}} := \emptyset$, and $L_{\text{corr}} := \emptyset$.

2) A can make polynomially many challenge queries and corruption queries as follows:

**Challenge Queries.** When C receives a challenge query $(\Delta, l = (id, \tau), m_0, m_1)$ (where $|m_0| = |m_1|$), it answers as follows:

a) If $(\Delta, l, m_0, m_1)$ is the first query for the dataset $\Delta$, C initializes an empty list $L_{\Delta} := \emptyset$ and proceeds as follows.

b) If $(\Delta, l, m)$ is the first query with identity id (that is, $(id, \ldots, \not\in L_{\text{key}})$), C generates $(ek_{id}, sk_{id}) \leftarrow \text{MVHE.KG}(pp, id)$ (that are implicitly assigned to the identity id), gives ek$_{id}$ to A, and updates the list $L_{\text{key}} \leftarrow L_{\text{key}} \cup \{(id, sk_{id})\}$.

If $(\Delta, l, m)$ is such that $(l, m) \not\in L_{\Delta}$, C computes $c_l \leftarrow \text{MVHE.Enc}(sk_{id}, \Delta, l, m)$ (note that C has already generated keys for the identity id), returns $c_l$ to A, and updates the list $L_{\Delta} \leftarrow L_{\Delta} \cup \{(l, m)\}$.

If $(\Delta, l, m)$ is such that $(l, \cdot) \in L_{\Delta}$ (which means that the adversary had already made a query $(\Delta, l, m')$ for some message $m'$), C ignores the query.

**Corruption Queries.** When C receives a corruption query $(\tau, \Delta, \ell, id)$, it answers as follows:

a) C checks whether id $\in L_{\text{ch}}$ holds. If this is the case, then C returns $\perp$ to A. Otherwise, C computes $c_l \leftarrow \text{MVHE.Enc}(sk_{id}, \Delta, l, m_0)$ (note that C has already generated keys for the identity id), gives $c_l$ to A, and updates the lists $L_{\Delta} \leftarrow L_{\Delta} \cup \{(l, m)\}$ and $L_{\text{ch}} \leftarrow L_{\text{ch}} \cup \{id\}$.

b) If $(\Delta, l, m_0, m_1)$ is such that $(l, \cdot) \in L_{\Delta}$ (which means that A had already made a query $(\Delta, l, m')$ for some plaintext $m'$), C ignores the query.

In this game, we define the advantage of the adversary A as

$$\text{Adv}_{\text{MVHE.A}}^{\text{priv}}(\lambda) := 2 \cdot \text{Pr}[b = b'] - \frac{1}{2}.$$ 

We say that MVHE satisfies privacy if for any PPT adversary A, $\text{Adv}_{\text{MVHE.A}}^{\text{priv}}(\lambda) = \text{negl}(\lambda)$ holds.

**Definition 9 (Unforgeability for MVHE):** Let MVHE be an MVHE scheme. We define the unforgeability game between a challenger C and an adversary A as follows.

1) C generates a public parameter $pp \leftarrow \text{MVHE.Setup}(\lambda)$, gives $pp$ to A, and prepares lists $L_{\text{key}} := \emptyset$ and $L_{\text{corr}} := \emptyset$.

2) A can adaptively make polynomially many authentication queries, verification queries, and corruption queries as follows:

**Authentication Queries.** When C receives an authentication query $(\Delta, l, m)$, where $\Delta$ is a dataset identifier, $l = (id, \tau)$ is a label in $\mathcal{ID} \times \mathcal{T}$, and $m \in \mathcal{M}$, it answers as follows:

- If $(\Delta, l, m)$ is the first query for the dataset $\Delta$, C initializes an empty list $L_{\Delta} := \emptyset$ and proceeds as follows.
- If $(id, \cdot) \not\in L_{\text{key}}$ holds, then C generates keys $(ek_{id}, sk_{id}) \leftarrow \text{MVHE.KG}(pp, id)$ (that are implicitly assigned to identity id), gives $ek_{id}$ to A and updates the list $L_{\text{key}} \leftarrow L_{\text{key}} \cup \{(id, sk_{id})\}$.
- If $(\Delta, l, m)$ is such that $(l, m) \not\in L_{\Delta}$, C computes $c_l \leftarrow \text{MVHE.Enc}(sk_{id}, \Delta, l, m)$ (note that C has already generated keys for the identity id), returns $c_l$ to A, and updates the list $L_{\Delta} \leftarrow L_{\Delta} \cup \{(l, m)\}$.
- If $(\Delta, l, m)$ is such that $(l, \cdot) \in L_{\Delta}$ (which means that the adversary had already made a query $(\Delta, l, m')$ for some message $m'$), C ignores the query.

**Verification Queries.** When C receives a verification query $(\mathcal{P}, \Delta, m, \sigma)$, it answers as follows:

a) If $(id, \cdot) \not\in L_{\text{key}}$ holds for some id $\in \mathcal{P}$, then C generates $(ek_{id}, sk_{id}) \leftarrow \text{MVHE.KG}(pp, id)$ and updates the list $L_{\text{key}} := L_{\text{key}} \cup \{(id, sk_{id})\}$.

b) C gives 1 to A if MVHE.Dec($\mathcal{P}, \Delta, \{sk_{id}\}_{id \in \mathcal{P}}$, $m, \sigma$) $\not\in \perp$ holds. Otherwise, C gives 0 to A.

**Corruption Queries.** When C receives a corruption query $(\tau, \Delta, \ell, id)$, it answers as follows:

a) If $(id, \cdot) \not\in L_{\text{key}}$ holds, then C generates $(ek_{id}, sk_{id}) \leftarrow \text{MVHE.KG}(pp, id)$ and updates the list $L_{\text{key}} := L_{\text{key}} \cup \{(id, sk_{id})\}$.

b) C gives $sk_{id}$ to A and appends id to $L_{\text{corr}}$.

c) A outputs a forgery $(P^{*} = (f^{*}, l_1, \ldots, l_T), \Delta^{*}, m^{*}, \sigma^{*})$.

In the above game, we say that an event Forge occurs if MVHE.Dec($\mathcal{P}^{*}, \Delta^{*}, \{vk_{id}\}_{id \in \mathcal{P}^{*}}, m^{*}, \sigma^{*}$) $\not\in \perp$ for all id $\in \mathcal{P}^{*}$, id $\in L_{\text{corr}}$, and either one of the following conditions holds:

- **Type 1:** $L_{\Delta^{*}}$ has not been initialized during the game.
- **Type 2:** For all $i \in [t]$, $3(l_i^{*}, m_i) \in L_{\Delta^{*}}$, but $m_i \neq f^{*}(m_1, \ldots, m_t)$.
- **Type 3:** There exists a label $l_i^{*}$ such that $(l_i^{*}, \cdot) \not\in L_{\Delta^{*}}$ for some $i \in [t]$.

We say that an MVHE scheme MVHE satisfies unforgeability if for any PPT adversary A,

$$\text{Adv}_{\text{MVHE.A}}^{\text{unf}}(\lambda) := \text{Pr}[\text{Forge}] = \text{negl}(\lambda).$$

**B. CONSTRUCTION**

In this section, we provide our construction of MVHE based on MHE and MHEA. Let MHE = (MHE.Setup, MHE.KG, MHE.Enc, MHE.Dec, MHE.Eval, MHE.Auth, MHE.Eval, MHE.Arch) be an MHE scheme and MHEA = (MHEA.Setup, MHEA.KG, MHEA.Auth, MHEA.Eval, MHEA.Arch, MHEA.Ver) an MHEA scheme. We assume that MHE.Setup and MHEA.Setup support the same plaintext space $\mathcal{M}$, function family $\mathcal{F} : \mathcal{M}^{t} \rightarrow \mathcal{M}$, and identity space $\mathcal{T}$. Using MHE and MHEA, our MVHE scheme MVHE = (MVHE.Setup, MVHE.KG, ...
MVHE.Enc, MVHE.Dec, MVHE.Eval) whose plaintext space $\mathcal{M}$ and identity space $\mathcal{ID}$ is described as follows.

**MVHE.Setup**$(1^\lambda)$:
- Generate $pp_{MHE} \leftarrow$ MHE.Setup$(1^\lambda)$.
- Generate $pp_{MHEA} \leftarrow$ MHEA.Setup$(1^\lambda)$.
- Return $pp := (pp_{MHE}, pp_{MHEA})$.

**MVHE.KG**(pp, id):
- Parse $pp := (pp_{MHE}, pp_{MHEA})$.
- Generate $(ek_{id}, sk_{id}) \leftarrow$ MHE.KG$(pp_{MHE}, id)$.
- Generate $(ek_{MHEA}, sk_{MHEA}) \leftarrow$ MHEA.KG$(pp_{MHEA}, id)$.
- Return $ek_{id} := (ek_{id}, ek_{MHEA})$ and $sk_{id} := (sk_{id}, sk_{MHEA})$.

**MVHE.Enc**(sk_{id}, $\Delta$, l, m):
- Parse $sk_{id} := (sk_{MHE}, sk_{MHEA})$ and $l := (id, \tau)$.
- Compute $c_{MHE} \leftarrow$ MHE.Enc$(sk_{MHE}, m)$.
- Compute $c_{MHEA} \leftarrow$ MHEA.Enc$(sk_{MHEA}, m)$.
- Return $c := (c_{MHE}, c_{MHEA})$.

**MVHE.Dec**($P$, $\Delta$, (sk_{id})_{id\in P}, c):
- Parse $c := (c_{MHE}, c_{MHEA})$.
- For all $id \in P$, parse $sk_{id} := (sk_{MHE}, sk_{MHEA})$.
- Compute $m \leftarrow$ MHEA.Dec$(P, (sk_{MHEA})_{id\in P}, c_{MHEA})$.
- If MHEA.Ver$(P, \Delta, (sk_{id})_{id\in P}, m, c_{MHEA}) = 1$ then return $m$. Otherwise, return $\bot$.

**MVHE.Eval**(f, ((c_i, EKS_i))_{i\in[l]}):
- For all $i \in [l]$, parse $c_i := (c_{MHE}, c_{MHEA})$.
- For all $i \in [l]$, parse $EKS_{i} := (ek_{id})_{id\in P_i} = (ek_{id}, ek_{MHEA})_{id\in P_i}$.
- For all $i \in [l]$, set $EKS_{MHE} := (ek_{id})_{id\in P_i}$ and $EKS_{MHEA} := (ek_{id})_{id\in P_i}$.
- Compute $c_{f} := (ek_{id})_{id\in P_i}$.
- Compute $c_{f} \leftarrow$ MHE.Eval$(f, ((c_{i}^{MHE}, EKS_{i}^{MHE}))_{i\in[l]})$.
- Compute $c_{f} \leftarrow$ MHEA.Eval$(f, ((c_{i}^{MHE}, EKS_{i}^{MHEA}))_{i\in[l]})$.
- Return $c_{f} := (c_{f}^{MHE}, c_{f}^{MHEA})$.

We can easily see that the ordinary correctness, evaluation correctness, and succinctness of MVHE are followed due to the ordinary correctness, evaluation correctness, and succinctness of MHE and MHEA.

**C. SECURITY PROOF**

In this section, we show that our MVHE scheme MVHE satisfies privacy (Theorem 1) and unforgeability (Theorem 2).

**Theorem 1**: If MHE satisfies IND-CPA security and MHEA satisfies privacy, then MVHE satisfies privacy.

**Proof of Theorem 1**: Let $Q_{\text{key}} := Q_{\text{key}}(\lambda)$ be an arbitrary polynomial that denotes the number of key pairs generated in the privacy game. Let $\mathcal{A}$ be any PPT adversary that attacks the privacy of MVHE. We proceed the proof via a sequence of games by introducing the following games: Game$_0$ for $i \in [0, 2]$.

**Game$_0$**: This is the original privacy game for MVHE conditioned on $b = 0$. The detailed description is as follows:

1. The challenger $\mathcal{C}$ proceeds as follows:
   a) $\mathcal{C}$ generates $pp_{MHE} \leftarrow$ MHE.Setup$(1^\lambda)$ and $pp_{MHEA} \leftarrow$ MHEA.Setup$(1^\lambda)$ and sets $pp := (pp_{MHE}, pp_{MHEA})$.
   b) $\mathcal{C}$ gives $pp$ to $\mathcal{A}$ and prepares lists $L_{key} := \emptyset$, $L_{ch} := \emptyset$, and $L_{corr} := \emptyset$.
2. When $\mathcal{A}$ makes challenge queries and corruption queries, $\mathcal{C}$ answers as follows:

   **Challenge Queries.** When $\mathcal{C}$ receives a challenge query $(\Delta, l = (id, \tau), m_0, m_1)$, it proceeds as follows:
   a) If $(\Delta, l_0, m_1)$ is the first query for the dataset $\Delta$, $\mathcal{C}$ initializes an empty list $L_{\Delta} := \emptyset$.
   b) If $(id, \cdot) \notin L_{key}$ holds, $\mathcal{C}$ proceeds as follows:
      i) $\mathcal{C}$ generates $(ek_{id}, sk_{id}) \leftarrow$ MHEA.KG$(pp_{MHEA}, id)$.
      ii) $\mathcal{C}$ generates $(ek_{id}, sk_{id}) \leftarrow$ MHEA.KG$(pp_{MHEA}, id)$.
      iii) $\mathcal{C}$ sets $ek_{id} := (ek_{id}, ek_{MHEA})$ and $sk_{id} := (sk_{id}, sk_{MHEA})$, gives $ek_{id}$ to $\mathcal{A}$, and appends $(id, sk_{id})$ to $L_{key}$.
   c) If $(\Delta, l_0, m_1)$ is such that $(l, \cdot) \notin L_{\Delta}$, $\mathcal{C}$ proceeds as follows:
      i) $\mathcal{C}$ checks whether $id \in L_{corr}$ holds. If this is the case, then $\mathcal{C}$ returns $\bot$ to $\mathcal{A}$.
      ii) $\mathcal{C}$ computes $c_{MHE} \leftarrow$ MHE.Enc$(sk_{MHE}, m_0)$ (using $sk_{MHE}$ included in $L_{key}$).
      iii) $\mathcal{C}$ computes $c_{MHEA} \leftarrow$ MHEA.Auth$(sk_{MHE}, \Delta, l, m_0)$ (using $sk_{MHEA}$ included in $L_{key}$).
      iv) $\mathcal{C}$ sets $c := (c_{MHE}, c_{MHEA})$, gives $c$ to $\mathcal{A}$, and appends $(l, id)$ to $L_{\Delta}$ and $id$ to $L_{ch}$.
   d) If $(\Delta, l_0, m_1)$ is such that $(l, \cdot) \in L_{\Delta}$, $\mathcal{C}$ ignores the query.

   **Corruption Queries.** When $\mathcal{C}$ receives a corruption query $id$, it answers as follows:
   a) $\mathcal{C}$ checks whether $id \in L_{ch}$ holds. If this is the case, then $\mathcal{C}$ returns $\bot$ to $\mathcal{A}$.
   b) $\mathcal{C}$ generates $(ek_{id}, sk_{id}) \leftarrow$ MHEA.KG$(pp_{MHEA}, id)$.
   c) $\mathcal{C}$ generates $(ek_{id}, sk_{id}) \leftarrow$ MHEA.KG$(pp_{MHEA}, id)$.
   d) $\mathcal{C}$ sets $sk_{id} := (sk_{id}, sk_{MHEA})$, gives $sk_{id}$ to $\mathcal{A}$, and appends $id$ to $L_{corr}$. 
3) \( A \) outputs a bit \( b' \in \{0, 1\} \).

**Game1**: This game is identical to \( \text{Game}_0 \) except that \( C \) computes \( c^{\text{MHE}} \leftarrow \text{MHE.Enc}(sk^{\text{MHE}}_{id}, m_1) \) instead of \( c \leftarrow \text{Enc}(sk_{id}, m_0) \) when responding to the challenge queries \((\Delta, l = (id, \tau), m_0, m_1)\).

**Game2**: This game is identical to \( \text{Game}_1 \) except that \( C \) computes \( \sigma^{\text{MHEA}} \leftarrow \text{MHEA.Auth}(sk^{\text{MHEA}}_{id}, \Delta, l, m_1) \) instead of \( \sigma^{\text{MHEA}} \leftarrow \text{MHEA.Auth}(sk^{\text{MHEA}}_{id}, \Delta, l, m_0) \) when responding to the challenge queries \((\Delta, l = (id, \tau), m_0, m_1)\). Note that this game is the same as the original privacy game for MVHE conditioned on \( b = 1 \).

For \( i \in [0, 2] \), let \( \text{Win}_i \) be the event that \( A \) outputs \( b' = 0 \) in \( \text{Game}_i \). By using triangle inequality, we have

\[
\text{Adv}_{\text{MVHE} \cdot A}^\text{priv}(\lambda) = 2 \cdot \left| \Pr[b = b'] - \frac{1}{2} \right|
= \left| \Pr[b' = 0 | b = 0] - \Pr[b' = 0 | b = 1] \right|
= \left| \Pr[\text{Win}_0] - \Pr[\text{Win}_2] \right|
\leq \sum_{i=0} \left| \Pr[\text{Win}_i] - \Pr[\text{Win}_{i+1}] \right|.
\]

It remains to show how each \( |\Pr[\text{Win}_i] - \Pr[\text{Win}_{i+1}]| \) is upper-bounded. To this end, we will show the following lemmata.

- There exists an adversary \( B_1 \) against the IND-CPA security of \( \text{MHE} \) such that \( |\Pr[\text{Win}_0] - \Pr[\text{Win}_1]| = \text{Adv}_{\text{ind-CPA} \cdot B_1}^\text{mhe}(\lambda) \) (Lemma 1).
- There exists an adversary \( B_2 \) against the privacy of \( \text{MHEA} \) such that \( |\Pr[\text{Win}_1] - \Pr[\text{Win}_2]| = \text{Adv}_{\text{MHEA} \cdot B_2}^\text{priv}(\lambda) \) (Lemma 2).

**Lemma 1**: There exists an adversary \( B_1 \) against the IND-CPA security of \( \text{MHE} \) such that \( |\Pr[\text{Win}_0] - \Pr[\text{Win}_1]| = \text{Adv}_{\text{ind-CPA} \cdot B_1}^\text{mhe}(\lambda) \).

Proof of Lemma 1:

We construct an adversary \( B_1 \) that attacks the IND-CPA security of \( \text{MHE} \) so that \( |\Pr[\text{Win}_0] - \Pr[\text{Win}_1]| = \text{Adv}_{\text{ind-CPA} \cdot B_1}^\text{mhe}(\lambda) \), using the adversary \( A \) as follows.

1) Upon receiving a public parameter \( pp^{\text{MHE}} \) and a set of evaluation keys \( \{ek_{\text{MHE}}^c \}_{c \in \{0, 1\}} \) from the challenger, \( B_1 \) proceeds as follows:
   a) \( B_1 \) generates \( pp^{\text{MHEA}} \leftarrow \text{MHEA.Setup}(\lambda) \) and sets \( pp := pp^{\text{MHE}}, pp^{\text{MHEA}} \).
   b) \( B_1 \) gives \( pp \) to \( A \) and initializes a list \( L_{\text{key}} := \emptyset \) and a counter \( \text{cnt} := 1 \).

2) When \( A \) makes challenge queries and corruption queries, \( B_1 \) answers as follows:

**Challenge Queries**. When \( B_1 \) receives a challenge query \((\Delta, l = (id, \tau), m_0, m_1)\), it proceeds as follows:
   a) If \((\Delta, l, m_0, m_1)\) is the first query for the dataset \( \Delta \), \( B_1 \) initializes an empty list \( L_{\Delta} := \emptyset \).
   b) If \((id, \tau) \notin L_{\text{key}} \) holds, \( B_1 \) proceeds as follows:
      i) \( B_1 \) sets \( ek_{\text{cnt}}^{\text{MHE}} := ek_{\text{cnt}}^{\text{MHE}} \). \((ek_{\text{cnt}}^{\text{MHE}}) \) is associated with an identity \( id \) and updates \( \text{cnt} := \text{cnt} + 1 \).
      ii) \( B_1 \) generates \( ek_{\text{cnt}}^{\text{MHE}, sk^{\text{MHE}}_{id}} \leftarrow \text{MHEA.KG}(pp^{\text{MHEA}}) \).
      iii) \( B_1 \) sets \( ek_{\text{id}} := (ek_{\text{id}}^{\text{MHE}, sk^{\text{MHE}}_{id}}) \), gives \( ek_{\text{id}} \) to \( A \) and appends \((id, ek_{\text{id}}, sk^{\text{MHE}}_{id})\) to \( L_{\text{key}} \).
   c) If \((\Delta, l, m_0, m_1)\) is such that \((l = (id, \tau), \cdot) \notin L_{\Delta}, B_1 \) proceeds as follows:
      i) \( B_1 \) checks whether \( id \in L_{\text{corr}} \) holds. If this is the case, then \( B_1 \) returns \( \perp \) to \( A \).
      ii) \( B_1 \) makes a challenge query \((i, m_0, m_1)\) to its challenger, where \( i \) is an index assigned to \( id \) in the previous step, and gets a ciphertext \( c^{\text{MHE}} \).
      iii) \( B_1 \) computes \( \sigma^{\text{MHEA}} \leftarrow \text{MHEA.Auth}(sk^{\text{MHEA}}_{id}, \Delta, l, m_0) \).
      iv) \( B_1 \) sets \( c := (c^{\text{MHE}}, \sigma^{\text{MHEA}}) \), gives \( c \) to \( A \), and appends \((l, m) \) to \( L_{\text{key}} \) and \( id \) to \( L_{\text{corr}} \).
   d) If \((\Delta, l, m_0, m_1)\) is such that \((\cdot, \cdot) \in L_{\Delta}, C\) ignores the query.

**Corruption Queries**. When \( B_1 \) receives a corruption query \( id \), it proceeds as follows:
   a) \( B_1 \) checks whether \( id \in L_{\text{ch}} \) holds. If this is the case, then \( B_1 \) returns \( \perp \) to \( A \).
   b) \( B_1 \) makes a corruption query \( \text{cnt} \) to its challenger, gets a secret key \( sk^{\text{MHE}}_{\text{cnt}} \), sets \( sk^{\text{MHE}}_{\text{cnt}} := sk^{\text{MHE}}_{\text{cnt}} \) (\( ek_{\text{cnt}}^{\text{MHE}} \) is associated with an identity \( id \)) and updates \( \text{cnt} := \text{cnt} + 1 \).
   c) \( B_1 \) generates \( sk^{\text{MHEA}}_{id} \leftarrow \text{MHEA.KG}(pp^{\text{MHEA}}) \).
   d) \( B_1 \) sets \( sk_{\text{id}} := (sk^{\text{MHEA}}_{id}, sk^{\text{MHEA}}_{id}) \), gives \( sk_{\text{id}} \) to \( A \), and appends \( id \) to \( L_{\text{corr}} \).

3) When \( A \) outputs a bit \( b' \in \{0, 1\} \) and terminates, \( B_1 \) outputs \( b' := 0 \) to the challenger and terminates if \( b' = 0 \) holds. Otherwise, \( B_1 \) outputs \( b' := 1 \) to the challenger and terminates.

In the following, we let \( \beta \) be the challenge bit for \( B_1 \) in the IND-CPA game. We can see that \( B_1 \) perfectly simulates \( \text{Game}_0 \) for \( A \) if it receives the challenge ciphertext \( c^{\text{MHE}} \leftarrow \text{MHE.Enc}(sk^{\text{MHE}}_{id}, m_0) \) from its challenger. This ensures that the probability that \( B_1 \) outputs 0 when \( \beta = 0 \) is exactly the same as the probability that \( \text{Win}_0 \) happens in \( \text{Game}_0 \). That is, \( \Pr[b' = 0 | \beta = 0] = |Pr[\text{Win}_0]| \) holds.

On the other hand, we can see that \( B_1 \) perfectly simulates \( \text{Game}_1 \) for \( A \) if it receives the challenge ciphertext \( c^{\text{MHE}} \leftarrow \text{MHE.Enc}(sk^{\text{MHE}}_{id}, m_1) \) from its challenger. This ensures that the probability that \( B_1 \) outputs 0 when \( \beta = 1 \) is exactly the same as the probability that \( \text{Win}_1 \) happens in \( \text{Game}_1 \). That is, \( \Pr[b' = 0 | \beta = 1] = |Pr[\text{Win}_1]| \) holds. Therefore, we have

\[
\text{Adv}_{\text{MHE} \cdot B_1}^\text{ind-CPA}(\lambda) = |\Pr[b' = 0 | \beta = 0] - \Pr[b' = 0 | \beta = 1]| = |Pr[\text{Win}_0] - Pr[\text{Win}_1]|.
\]

\( \square \) (Lemma 1)
Lemma 2: There exists an adversary $B_2$ against the privacy of MHEA such that $|Pr[W_{1}] - Pr[W_{2}]| = \text{Adv}_{\text{MHEA}, B_2}^\text{priv}(\lambda)$. 

Proof of Lemma 2:

We construct an adversary $B_2$ that attacks the privacy of MHEA so that $|Pr[W_{1}] - Pr[W_{2}]| = \text{Adv}_{\text{MHEA}, B_2}^\text{priv}(\lambda)$, using the adversary $A$ as follows:

1. Upon receiving a public parameter $pp^{\text{MHEA}}$, $B_2$ generates $pp^{\text{MHE}} \leftarrow \text{MHE.Setup}(1^\lambda)$, gives $pp := (pp^{\text{MHE}}, pp^{\text{MHEA}})$ to $A$, and initializes a list $L_{\text{key}} := \emptyset$.

2. When $A$ makes challenge queries and corruption queries, $B_2$ answers as follows:

   Challenge Queries. When $B_2$ receives a challenge query $(\Delta, l = (id, \tau), m_0, m_1)$, it proceeds as follows:
   
   a) If $(\Delta, l, m_0, m_1)$ is the first query for the dataset $\Delta$, $B_2$ initializes an empty list $L_{\Delta} := \emptyset$.
   
   b) If $(\Delta, l, m_0, m_1)$ is such that $(l = (id, \tau), \cdot) \notin L_{\Delta}$, $B_2$ proceeds as follows:
      
      i) $B_2$ makes a challenge query $(\Delta, l = (id, \tau), m_0, m_1)$ to its challenger and gets an authentication $\sigma^{\text{MHEA}}$. If $\sigma^{\text{MHEA}} = \perp$ holds, then $B_2$ returns $\perp$ to $A$.
      
      ii) If $(\Delta, l, m)$ is the first query with identity $id$ that is, $(id, \cdot) \notin L_{\text{key}}$, $B_2$ gets $ek_{id}^{\text{MHEA}}$ from its challenger, computes $(ek_{id}^{\text{MHE}}, sk_{id}^{\text{MHE}}) \leftarrow \text{MHE.KG}(pp^{\text{MHE}}, id)$, gives $ek_{id} := (ek_{id}^{\text{MHE}}, ek_{id}^{\text{MHEA}})$ to $A$, and appends $(id, sk_{id}^{\text{MHE}})$ to $L_{\text{key}}$.
      
      iii) $B_2$ computes $c^{\text{MHE}} \leftarrow \text{MHE.Enc}(sk_{id}^{\text{MHE}}, m_1)$ (using a secret key $sk_{id}^{\text{MHE}} \in L_{\text{key}}$).
      
      iv) $B_2$ sets $c := (c^{\text{MHE}}, \sigma^{\text{MHEA}})$, gives $c$ to $A$, and appends $(l, m)$ to $L_{\Delta}$.
      
   c) If $(\Delta, l, m_0, m_1)$ is such that $(l, \cdot) \in L_{\Delta}$, $C$ ignores the query.

   Corruption Queries. When $B_2$ receives a corruption query $id$, it proceeds as follows:

   a) $B_2$ makes a corruption query $id$ to its challenger and gets a secret key $sk_{id}^{\text{MHEA}}$. If $sk_{id}^{\text{MHEA}} = \perp$ holds, then it also gives $\perp$ to $A$.
   
   b) $B_2$ generates $(ek_{id}^{\text{MHE}}, sk_{id}^{\text{MHE}}) \leftarrow \text{MHE.KG}(pp^{\text{MHE}}, id)$. $B_2$ sets $sk_{id} := (sk_{id}^{\text{MHE}}, sk_{id}^{\text{MHEA}})$ and gives $sk_{id}$ to $A$.

3. When $A$ outputs a bit $b' \in \{0, 1\}$ and terminates, $B_2$ outputs $b' = 0$ to the challenger and terminates if $b' = 0$ holds. Otherwise, $B_2$ outputs $b' := 1$ to the challenger and terminates.

In the following, we let $\beta$ be the challenge bit for $B_2$ in the privacy game (of MHEA). We can see that $B_2$ perfectly simulates $\text{Game}_1$ for $A$ if it receives the challenge authentication $\sigma^{\text{MHEA}} \leftarrow \text{MHE.Auth}(sk_{id}^{\text{MHE}}, \Delta, l, m_0)$ from its challenger. This ensures that the probability that $B_2$ outputs 0 when $\beta = 0$ is exactly the same as the probability that $Win_1$ happens in $\text{Game}_1$. That is, $Pr[\beta' = 0|\beta = 0] = Pr[Win_1]$ holds.

On the other hand, we can see that $B_2$ perfectly simulates $\text{Game}_2$ for $A$ if it receives the challenge authentication $\sigma^{\text{MHEA}} \leftarrow \text{MHE.Auth}(sk_{id}^{\text{MHE}}, \Delta, l, m_1)$ from its challenger. This ensures that the probability that $B_2$ outputs 0 when $\beta = 1$ is exactly the same as the probability that $Win_2$ happens in $\text{Game}_2$. That is, $Pr[\beta' = 0|\beta = 1] = Pr[Win_2]$ holds. Therefore, we have

$$\text{Adv}_{\text{MHEA}, B_2}^\text{priv}(\lambda) = |Pr[\beta' = 0|\beta = 0] - Pr[\beta' = 0|\beta = 1]|$$

$= |Pr[W_{1}] - Pr[W_{2}]|$

$\square$ (Lemma 2)

Putting everything together, we obtain

$$\text{Adv}_{\text{MVHE}, A}^\text{ind-cpa}(\lambda) \leq \text{Adv}_{\text{MHEA}, \text{B}_1}^\text{priv}(\lambda) + \text{Adv}_{\text{MHEA}, B_2}^\text{priv}(\lambda).$$

Since MHE satisfies IND-CPA security and MHEA satisfies privacy, for any PPT adversary $A$, $\text{Adv}_{\text{MVHE}, A}^\text{ind-cpa}(\lambda) = \text{negl}(\lambda)$ holds. Therefore, MVHE satisfies privacy.

$\square$ (Theorem 1)

Theorem 2: If MHEA satisfies unforgeability, then MVHE satisfies unforgeability.

Proof of Theorem 2: Let $A$ be any PPT adversary that attacks the unforgeability of MVHE. The original unforgeability game is described as follows.

1. The challenger $C$ proceeds as follows:
   a) $C$ generates $pp^{\text{MHE}} \leftarrow \text{MHE.Setup}(1^\lambda)$ and $pp^{\text{MHEA}} \leftarrow \text{MHEA.Setup}(1^\lambda)$ and sets $pp := (pp^{\text{MHE}}, pp^{\text{MHEA}})$.
   b) $C$ gives $pp$ to $A$ and prepares lists $L_{\text{key}} := \emptyset$, $L_{\text{ch}} := \emptyset$, and $L_{\text{cor}} := \emptyset$.

2. When $A$ makes authentication queries, verification queries, and corruption queries, $C$ answers as follows:

   Authentication Queries. When $C$ receives an authentication query $(\Delta, l = (id, \tau), m)$, it proceeds as follows:
   a) If $(\Delta, l, m)$ is the first query for the dataset $\Delta$, $C$ initializes an empty list $L_{\Delta} := \emptyset$.
   b) If $(id, \cdot, \cdot) \notin L_{\text{key}}$ holds, $C$ proceeds as follows:
      
      i) $C$ generates $(ek_{id}^{\text{MHE}}, sk_{id}^{\text{MHE}}) \leftarrow \text{MHE.KG}(pp^{\text{MHE}}, id)$.
      
      ii) $C$ generates $(ek_{id}^{\text{MHE}}, sk_{id}^{\text{MHE}}) \leftarrow \text{MHEA.KG}(pp^{\text{MHEA}}, id)$.
      
      iii) $C$ sets $ek_{id} := (ek_{id}^{\text{MHE}}, ek_{id}^{\text{MHEA}})$ and $sk_{id} := (sk_{id}^{\text{MHE}}, sk_{id}^{\text{MHEA}})$, gives $ek_{id}$ to $A$, and appends $(id, sk_{id})$ to $L_{\text{key}}$.
      
   c) If $(\Delta, l, m)$ is such that $(l, m) \notin L_{\Delta}$, $C$ proceeds as follows:
      
      i) $C$ computes $c^{\text{MHE}} \leftarrow \text{MHE.Enc}(sk_{id}^{\text{MHE}}, m)$ (using $sk_{id}^{\text{MHE}}$ included in $L_{\text{key}}$).
      
      ii) $C$ computes $\sigma^{\text{MHEA}} \leftarrow \text{MHEA.Auth}(sk_{id}^{\text{MHE}}, \Delta, l, m)$ (using $sk_{id}^{\text{MHEA}}$ included in $L_{\text{key}}$).
iii) $\mathcal{C}$ sets $c := (c^{MHE}, \sigma^{MHE})$, gives $c$ to $\mathcal{A}$, and appends $(l, m)$ to $L_{\Delta}$.

d) If $(\Delta, l, m)$ is such that $(l, \cdot) \in L_{\Delta}$, $\mathcal{C}$ ignores the query.

**Verification Queries.** When $\mathcal{C}$ receives a verification query $(\mathcal{P}, \Delta, m, c)$, it proceeds as follows:

a) $\mathcal{C}$ parses $c := (c^{MHE}, \sigma^{MHE})$ and $\mathcal{P} := (f, i_{d1}, \ldots, i_{d1})$.

b) If $(id, \cdot) \notin L_{\text{key}}$ holds for some $id \in \mathcal{P}$, then $\mathcal{C}$ proceeds as follows:

i) $\mathcal{C}$ generates $(ek^{MHE}_{id}, sk^{MHE}_{id}) \leftarrow MHE.KG(pp^{MHE}_{id}, id)$.

ii) $\mathcal{C}$ generates $(ek^{MHE}_{id}, sk^{MHE}_{id}) \leftarrow MHEA.KG(pp^{MHEA}_{id}, id)$.

iii) $\mathcal{C}$ sets $ek_{id} := (ek^{MHE}_{id}, ek^{MHEA}_{id})$ and $sk_{id} := (sk^{MHE}_{id}, sk^{MHEA}_{id})$, gives $ek_{id}$ to $\mathcal{A}$, and appends $(id, sk_{id})$ to $L_{\text{key}}$.

c) $\mathcal{C}$ computes $m \leftarrow MHE.Dec(\mathcal{P}, (sk^{MHE}_{id})_{id \in \mathcal{P}}, c^{MHE})$.

d) $\mathcal{C}$ checks whether $\text{MHEA.Ver(}\mathcal{P}, \Delta, (sk^{MHEA}_{id})_{id \in \mathcal{P}}, m, \sigma^{MHEA}) = 1$ holds. If this is the case, then $\mathcal{C}$ returns 1 to $\mathcal{A}$. Otherwise, $\mathcal{C}$ returns 0 to $\mathcal{A}$.

**Corruption Queries.** When $\mathcal{C}$ receives a corruption query $id$, it answers as follows:

a) $\mathcal{C}$ checks whether $(id, \cdot) \notin L_{\text{key}}$ holds. If this is the case, then $\mathcal{C}$ proceeds as follows:

i) $\mathcal{C}$ generates $(ek^{MHE}_{id}, sk^{MHE}_{id}) \leftarrow MHE.KG(pp^{MHE}_{id}, id)$.

ii) $\mathcal{C}$ generates $(ek^{MHE}_{id}, sk^{MHE}_{id}) \leftarrow MHEA.KG(pp^{MHEA}_{id}, id)$.

iii) $\mathcal{C}$ sets $ek_{id} := (ek^{MHE}_{id}, ek^{MHEA}_{id})$ and $sk_{id} := (sk^{MHE}_{id}, sk^{MHEA}_{id})$, gives $ek_{id}$ to $\mathcal{A}$, and appends $(id, sk_{id})$ to $L_{\text{key}}$.

b) $\mathcal{C}$ gives $sk_{id}$ to $\mathcal{A}$ and appends $id$ to $L_{\text{corr}}$.

3) $\mathcal{A}$ outputs a forgery $(\mathcal{P^{*}} = (f^{*}, l_{1}^{*}, \ldots, l_{n}^{*}), \Delta^{*}, m^{*}, \sigma^{*})$.

In the following, we show that there exists an adversary $B$ against the unforgeability of MHEA under the adversary $A$ as follows.

1) Upon receiving a public parameter $pp^{MHEA}$, $B$ generates $pp^{MHE} \leftarrow \text{MHE.Setup}(1^{\lambda})$, gives $pp := (pp^{MHE}, pp^{MHEA})$ to $\mathcal{A}$, and initializes a list $L_{\text{key}} := \emptyset$.

2) When $\mathcal{A}$ makes authentication queries, verification queries, and corruption queries, $B$ answers as follows:

**Authentication Queries.** When $\mathcal{B}$ receives an authentication query $(\Delta, l = (id, \tau), m)$, it proceeds as follows:

a) If $(\Delta, l, m)$ is the first query for the dataset $\Delta$, $\mathcal{B}$ initializes an empty list $L_{\Delta} := \emptyset$.

b) If $(\Delta, l, m)$ is such that $(l = (id, \tau), m) \notin L_{\Delta}$, $\mathcal{B}$ proceeds as follows:

i) $\mathcal{B}$ makes an authentication query $(\Delta, l = (id, \tau), m)$ to its challenger and gets an authentication $\sigma^{MHEA}$.

ii) If $(id, \cdot) \notin L_{\text{key}}$ holds, $B$ gets $ek_{id}^{MHEA}$ from its challenger, generates $(ek^{MHE}_{id}, sk^{MHE}_{id}) \leftarrow MHE.KG(pp^{MHE}_{id}, id)$, gives $ek_{id}^{MHEA} := (ek^{MHE}_{id}, ek^{MHEA}_{id})$ to $\mathcal{A}$, and appends $(id, sk_{id})$ to $L_{\text{key}}$.

iii) $\mathcal{B}$ computes $c^{MHE} \leftarrow MHE.Enc(sk^{MHEA}_{id}, m)$ (using a secret key $sk^{MHEA}_{id} \in L_{\text{key}}$).

iv) $\mathcal{B}$ sets $c := (c^{MHE}, \sigma^{MHEA})$, gives $c$ to $\mathcal{A}$, and appends $(l, m)$ to $L_{\Delta}$.

c) If $(\Delta, l, m)$ is such that $(l, \cdot) \in L_{\Delta}$, $\mathcal{C}$ ignores the query.

**Verification Queries.** When $\mathcal{B}$ receives a verification query $(\mathcal{P}, \Delta, m, c)$, it proceeds as follows:

a) $\mathcal{B}$ parses $c := (c^{MHE}, \sigma^{MHEA})$ and $\mathcal{P} := (f, i_{d1}, \ldots, i_{d1})$.

b) If $(id, \cdot) \notin L_{\text{key}}$ holds for some $id \in \mathcal{P}$, then $\mathcal{B}$ proceeds as follows:

i) $\mathcal{B}$ generates $(ek^{MHE}_{id}, sk^{MHE}_{id}) \leftarrow MHEA.KG(pp^{MHEA}_{id}, id)$.

ii) $\mathcal{B}$ sets $ek_{id} := (ek^{MHE}_{id}, ek^{MHEA}_{id})$, gives $ek_{id}$ to $\mathcal{A}$, and appends $(id, sk_{id})$ to $L_{\text{key}}$.

c) $\mathcal{B}$ computes $m \leftarrow MHEA.Dec(\mathcal{P}, (sk^{MHE}_{id})_{id \in \mathcal{P}}, c^{MHE})$.

$\mathcal{B}$ makes a verification query $(\Delta, m, c^{MHEA})$ to its challenger, gets a result $v$, and returns $v$ to $\mathcal{A}$.

**Corruption Queries.** When $\mathcal{B}$ receives a corruption query $id$, it proceeds as follows:

a) $\mathcal{B}$ makes a corruption query $id$ to its challenger and gets a secret key $sk^{MHEA}_{id}$. If $sk^{MHEA}_{id} \notin \mathcal{L}$ holds, then it also gives $\perp$ to $\mathcal{A}$.

b) $\mathcal{B}$ generates $(ek^{MHE}_{id}, sk^{MHE}_{id}) \leftarrow MHEA.KG(pp^{MHEA}_{id}, id)$.

c) $\mathcal{B}$ sets $sk_{id} := (sk^{MHE}_{id}, sk^{MHEA}_{id})$ and gives $sk_{id}$ to $\mathcal{A}$.

3) When $\mathcal{A}$ outputs a forgery $(\mathcal{P^{*}} = (f^{*}, l_{1}^{*}, \ldots, l_{n}^{*}), \Delta^{*}, m^{*}, \sigma^{*})$ and terminates, $\mathcal{B}$ proceeds as follows:

a) $\mathcal{B}$ parses $c^{*} := (c^{MHEA}, \sigma^{MHEA^{*}})$ and $\mathcal{P}^{*} := (f^{*}, i_{d1}^{*}, \ldots, i_{d1}^{*})$.

b) $\mathcal{B}$ computes $m^{*} \leftarrow MHEA.Dec(\mathcal{P^{*}}, (sk^{MHEA}_{id})_{id \in \mathcal{P^{*}}}, c^{MHE})$.

c) $\mathcal{B}$ outputs $(\mathcal{P^{*}}, \Delta^{*}, m^{*}, \sigma^{MHEA^{*}})$ to its challenger and terminates.

We can see that $B$ perfectly simulates the unforgeability game for $\mathcal{A}$. In the following, we show that $\mathcal{B}$ can output a valid forgery $(\mathcal{P^{*}}, \Delta^{*}, m^{*}, \sigma^{MHEA^{*}})$ if $\mathcal{A}$ outputs a valid forgery $(\mathcal{P^{*}}, \Delta^{*}, c^{*})$. Firstly, if $\mathcal{A}$ outputs a valid forgery $(\mathcal{P^{*}}, \Delta^{*}, c^{*})$, due to the construction of MHEA,
\[ \downarrow \neq m^* \rightarrow \text{MHE.DEC}(P^*, \Delta^*, (sk_{id})_{id \in P^*}, \text{MHE}^*) \]

1 \rightarrow \text{MHE.A.VER}(sk_{id})_{id \in P^*}, m^*, P^*, \sigma_{\text{MHE}^*}) \text{ and } id \notin L_{\text{corr}} \text{ hold for all } id \in P^*. \text{ Next, if } (P^*, \Delta^*, c^*) \text{ satisfies at least one of the condition of Type } X (X \in \{1, 2, 3\}). \text{ For any } X \in \{1, 2, 3\}, \text{ we can see that if } (P^*, \Delta^*, c^*) \text{ satisfies the condition of Type } X, \text{ then } (P^*, \Delta^*, m^*, \sigma_{\text{MHE}^*}) \text{ (output by B) also satisfies the condition of Type } X \text{ in the game of unforgeability for MHEA. Thus, if } A \text{ outputs a valid forgery, then } B \text{ can output a valid forgery. That is, we have } \text{Adv}_{\text{MVHE-A}}(\lambda) = \text{Adv}_{\text{MHEA-B}}(\lambda). \text{ Since MHEA satisfies unforgeability, for any PPT adversary } A, \text{ } \text{Adv}_{\text{MVHE-A}}(\lambda) = \text{negl}(\lambda) \text{ holds. Therefore, MVHE satisfies unforgeability.}

\Box \text{(Theorem 2)}

V. CONCLUSION

In this paper, we propose a new notion of multi-key verifiable homomorphic encryption (MVHE) as a core cryptographic primitive for realizing private and verifiable delegating computation in the multi-user setting as an application. Concretely, we provide a formal syntax and security definitions for MVHE and the generic construction based on an MHE scheme and an MHEA scheme. As mentioned in Section III-C, due to the restriction for our MHEA scheme occurred by the previous frameworks, our MVHE scheme (i) is resilient to only fixed a-priori bounded number of verification queries and (ii) supports only constant number of users. Then, we leave as an interesting open problem to examine how to construct a fully secure VHE scheme for all polynomial-sized circuits overcoming these restrictions. Also, since our work provides only a generic construction and its theoretical feasibility, we also leave to provide an efficient concrete construction of MVHE as a future work.

REFERENCES

[1] P. Ananth, A. Jain, Z. Jin, and G. Malavolta, “Multi-key fully-homomorphic encryption in the plain model,” in Theory of Cryptography (Lecture Notes in Computer Science). Heidelberg, Germany: Springer, Mar. 2020, pp. 28–57, doi: 10.1007/978-3-030-64375-1_2.

[2] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, “From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again,” in Proc. 3rd Innov. Theor. Comput. Sci. Conf. (ITCS), 2012, pp. 326–349, doi: 10.1145/2090236.2090263.

[3] A. Boi, I. Cascudo, D. Fiore, and D. Kim, “Flexible and efficient verifiable computation on encrypted data,” in Public-Key Cryptography (Lecture Notes in Computer Science). Heidelberg, Germany: Springer, 2021, pp. 528–538, doi: 10.1007/978-3-030-75248-4_19.

[4] M. Ben-Or and A. Wigderson, “Completeness theorems for non-cryptographic fault-tolerant distributed computation,” in 26th Ann. ACM Symp. Theory Comput. (STOC), 1988, pp. 1–10.

[5] Z. Brakerski and R. Perlman, “Lattice-based fully dynamic multi-key FHE with short ciphertexts,” in Advances in Cryptology (Lecture Notes in Computer Science), vol. 9814, M. Robshaw and J. Katz, Eds. Heidelberg, Germany: Springer, May 2016, pp. 735–763, doi: 10.1007/978-3-662-49896-5_26.

[10] D. Fiore, R. Gennaro, and V. Pastro, “Efficiently verifiable computation on encrypted data,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Nov. 2014, pp. 844–855, doi: 10.1145/2660267.2660566.

[11] D. Fiore, A. Minaev, L. Nicosia, and E. Pagnin, “Multi-key homomorphic authentications,” in Advances in Cryptology (Lecture Notes in Computer Science), vol. 10032, J. Hee and C. T. Takagi, Eds. Heidelberg, Germany: Springer, Dec. 2016, pp. 499–530, doi: 10.1007/978-3-662-53890-6_17.

[12] D. Fiore and A. Nitulessu, “On the (in)security of SNARKs in the presence of oracles,” in Theory of Cryptography (Lecture Notes in Computer Science), vol. 9985, M. H. Adam and D. Smith, Eds. Heidelberg, Germany: Springer, Oct./Nov. 2016, pp. 108–138, doi: 10.1007/978-3-319-70503-3_20.

[13] R. Gennaro, C. Gentry, and B. Parno, “Non-interactive verifiable computing: Outsourcing computation to untrusted workers,” in Advances in Cryptology (Lecture Notes in Computer Science), vol. 6223, T. Rabin, Ed. Heidelberg, Germany: Springer, Aug. 2010, pp. 465–482, doi: 10.1007/978-3-642-14623-7_25.

[14] D. Fiore and E. Pagnin, “Matrioska: A compiler for multi-key homomorphic signatures,” in Security and Cryptography for Networks (Lecture Notes in Computer Science), vol. 11035, D. Catalano and R. De Prisco, Eds. Heidelberg, Germany: Springer, Sep. 2018, pp. 43–62, doi: 10.1007/978-3-319-98113-0_3.

[15] S. Gorbunov, V. Vaikuntanathan, and D. Wichs, “Leveled fully homomorphic encryption from standard lattices,” in Advances in Cryptology (Lecture Notes in Computer Science), vol. 8043, A. Zeldovich, “How to run Turing machines on encrypted data,” in Advances in Cryptology (Lecture Notes in Computer Science), vol. 9093, D. Catalano and J. A. Garay, Eds. Heidelberg, Germany: Springer, Aug. 2013, pp. 536–553, doi: 10.1007/978-3-642-40084-1_30.

[16] D. Goldwasser, S. Micali, and A. Wigderson, “How to play any mental game or a completeness theorem for protocols with honest majority,” in Proc. 19th Annu. ACM Symp. Theory Comput., Jun. 1987, pp. 218–229, doi: 10.1145/28395.28420.

[17] S. Mukherjee and D. Wichs, “Two round multiparty computation via multikey fully homomorphic encryption,” in Advances in Cryptology (Lecture Notes in Computer Science), vol. 9666, R. G. Matthew and J.-S. Coron, Eds. Heidelberg, Germany: Springer, May 2016, pp. 735–763, doi: 10.1007/978-3-319-70503-3_20.

[18] S. Gorbunov, V. Vaikuntanathan, and D. Wichs, “Leveled fully homomorphic signatures from standard lattices,” in Proc. 47th Annu. ACM Symp. Theory Comput., Jun. 2015, pp. 469–477, doi: 10.1145/2746539.2746576.

[19] R. Gennaro and D. Wichs, “Fully homomorphic message authentications,” in Advances in Cryptology (Lecture Notes in Computer Science), vol. 8270, K. Sako and P. Sarkar, Eds. Heidelberg, Germany: Springer, Dec. 2013, pp. 301–320, doi: 10.1007/978-3-642-42045-0_16.

[20] J. Lai, H. R. Deng, H. Pang, and J. Weng, “Verifiable computation on outsourced encrypted data,” in Computer Security (Lecture Notes in Computer Science), vol. 8712, M. Kutylowski and J. Vaidya, Eds. Heidelberg, Germany: Springer, Sep. 2014, pp. 273–291, doi: 10.1007/978-3-319-11203-9_16.

[21] A. López-Alt, E. Tromer, and V. Vaikuntanathan, “On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption,” in Proc. 44th ACM STOC, H. J. Karloff and T. Pitassi, Eds., May 2012, pp. 1219–1234, doi: 10.1145/2213977.2214086.

[22] W. F. R. Lai, K. H. R. Tai, W. H. H. Wong, and S. M. S. Chow, “Multi-key homomorphic signatures unforgeable under insider corruption,” in Advances in Cryptology (Lecture Notes in Computer Science), vol. 11275, T. Peyrin and S. Galbraith, Eds. Heidelberg, Germany: Springer, Dec. 2018, pp. 465–492, doi: 10.1007/978-3-030-33329-3_16.

[23] P. Mukherjee and D. Wichs, “Two round multiparty computation via multi-key FHE,” in Advances in Cryptology (Lecture Notes in Computer Science), vol. 9666, M. Fischlin and J.-S. Coron, Eds. Heidelberg, Germany: Springer, May 2016, pp. 735–763, doi: 10.1007/978-3-319-70503-3_20.
[24] B. Parno, M. Raykova, and V. Vaikuntanathan, “How to delegate and verify in public: Verifiable computation from attribute-based encryption,” in *Theory of Cryptography* (Lecture Notes in Computer Science), vol. 7194, R. Cramer, Ed. Heidelberg, Germany: Springer, Mar. 2012, pp. 422–439, doi: 10.1007/978-3-642-28914-9_24.

[25] C. Peikert and S. Shiehian, “Multi-key FHE from LWE, revisited,” in *Theory of Cryptography* (Lecture Notes in Computer Science), vol. 9986, M. H. Adam and D. Smith, Eds. Heidelberg, Germany: Springer, Nov. 2016, pp. 217–238, doi: 10.1007/978-3-662-53644-5_9.

**YI LU** received the bachelor’s degree from the School of Computer Science and Technology, Dalian University of Technology, in 2019, and the master’s degree from the Department of Mathematical and Computing Science, Tokyo Institute of Technology, in 2021, where she is currently pursuing the Ph.D. degree. She is also a Research Assistant with the National Institute of Advanced Industrial Science and Technology (AIST).

**KEISUKE HARA** received the bachelor’s degree from the Department of Information Science, Tokyo Institute of Technology, in 2017, and the M.S. and Ph.D. degrees from the Department of Mathematical and Computing Science, Tokyo Institute of Technology, in 2019 and 2022, respectively. He is currently a Research Fellow of the Japan Society for the Promotion of Science (JSPS).

**KEISUKE TANAKA** received the B.E. degree from Yamanashi University, in 1992, and the M.S. and Ph.D. degrees from the Japan Advanced Institute of Science and Technology, in 1994 and 1997, respectively. For each degree, he majored in computer science. Before joining the Tokyo Institute of Technology, he was a Research Engineer at NTT Information Sharing Platform Laboratories. He is currently a Professor with the Department of Mathematical and Computing Sciences, Tokyo Institute of Technology.

***