A Constructive View of Homomorphic Encryption and Authenticator

Ganyuan Cao[a]

EPFL, Lausanne, Switzerland

Abstract. Homomorphic Encryption (HE) is a cutting-edge cryptographic technique that enables computations on encrypted data to be mirrored on the original data. This has quickly attracted substantial interest from the research community due to its extensive practical applications, such as in cloud computing and privacy-preserving machine learning.

In addition to confidentiality, the importance of authenticity has emerged to ensure data integrity during transmission and evaluation. To address authenticity, various primitives have been developed including Homomorphic Authenticator (HA). Corresponding security notions have also been introduced by extending the existing notions to their homomorphic versions.

Despite these advancements, formalizing the security of HE and HA remains challenging due to the novelty of these primitives and complexity of application scenarios involving message evaluation. It is inclusive which definitions in this zoo of notions are insufficient or overly complex. Moreover, HE and HA are designed to be combined to construct a secure communication channel that ensures both confidentiality and authenticity. However, the security of such compositions is not always clear when game-based notions are used to formalize security.

To bridge this gap, we conduct a constructive analysis through the lens of composable security. This method enables us to examine the security properties of each primitive in isolation and to more effectively evaluate their security when integrated into a larger system. We introduce the concepts of a confidential channel and an authenticated channel to specify the security requirements for HE and HA, respectively. We make a comparison with existing game-based notions to determine whether they adequately capture the intended security objectives.

We then analyze whether the composition of HE and HA constructs a Homomorphic Authenticated Encryption (HAE) that provides both confidentiality and authenticity in presence of message evaluation. Specifically, we examine a serial composition of HE and HA, corresponding to Encrypt-then-MAC (EtM) composition for constructing classical AE.

Keywords: Homomorphic Encryption · Homomorphic Authenticator · Composable Security · Constructive Cryptography · Provable Security

Contents

E-mail: ganyuan.cao@epfl.ch (Ganyuan Cao)

[a]The author is jointly affiliated at EPFL and ETH Zürich

Date of this document: 2024-08-10.

1 Introduction

1.1 Background and Motivation

Homomorphic Encryption (HE), first introduced by Rivest and Adleman as “privacy homomorphism” in [RAD+78], enables computations on encrypted data, producing results that can be directly translated back to the original data. This capability opens up significant opportunities for real-world applications, such as privacy-preserving machine learning [BP23], cloud computing [ZLL14], and multiparty computation (MPC) [BDOZ11]. Over the years, various HE schemes [C+09, BGV12, CGGI20] have been developed to improve efficiency and support a broader range of operations, moving towards full homomorphism.

In addition to ensuring confidentiality, the cryptographic community has increasingly focused on maintaining authenticity within a homomorphic framework. Since adversaries may modify messages before or after evaluation to produce false results, it is crucial to address this vulnerability. Agrawal and Boneh proposed homomorphic MACs in [AB09], and Johnson introduced homomorphic signatures in [JMSW02], approaching the problem from symmetric and public-key cryptography perspectives, respectively. Further research has aimed to provide authenticity for HE through verifiable control of the evaluation algorithm using techniques like SNARKs [Via23]. Various notions have also been proposed to address different adversarial scenarios that may arise during message evaluation.

However, game-based proof framework by Bellare and Rogaway [BR06] may not be the most effective tool for formalizing security in this context. The complication of defining a game or adversarial behavior increases with the presence of message evaluation. Moreover, these primitives are intended for use in large, complex systems, yet the composition property of these primitives in such systems remains unclear with game-based notions, as noted by Maurer in [Mau11]. To bridge this gap, our goal is to present an analysis of HE and Homomorphic Authenticator (HA) from a constructive perspective, capturing the inherent security of HE and HA while facilitating future research on the security of large-scale systems incorporating these primitives. This approach particularly allows us to offer insights into the construction of Homomorphic Authenticated Encryption (HAE) through the composition of HE and HA.

1.2 Related Work

Confidentiality. The primary security requirement of homomorphic encryption (HE) is *confidentiality*, characterized by the IND-CPA notion, as described by Gentry in [C^{+}09]. In addition to basic confidentiality, Gentry also explored *circular security* in [C^{+}09], where an adversary can access a "cycle of keys," meaning a key is encrypted under itself or another key, encapsulating security involving the *bootstrapping* operation. Further confidentiality requirements include *circuit privacy*, which ensures that an adversary cannot distinguish between the evaluated ciphertext and the encryption of the evaluated plaintext. Moreover, Li et al. introduced the concept of IND-CPA$^{\mathrm{D}}$ in [LM21], specifically for approximate HE schemes like CKKS [CKKS17], demonstrating its equivalence to IND-CPA for HE schemes with exact decryption.

Authenticity. For authenticity, Agrawal and Boneh proposed the concept of homomorphic MAC in [AB09], identifying two types of forgeries. Gennaro and Wichs further formalized this security by introducing the concept of a labeled program in [GW13], later termed HomoUF-CMA security in [CF13]. Based on this, in [JY14], Joo and Yun further formalized the notions of plaintext integrity INT-PTXT and ciphertext integrity INT-CTXT within the homomorphic setting by also taking the honest execution of the evaluation algorithm into consideration.

Towards CCA Security. Additionally, there are studies focused on constructing CCA-secure HE schemes that also consider authenticity. Prabhakaran and Mike Rosulek introduced HCCA in [PR08], capturing the idea that a scheme may be homomorphic for specific functions but must remain non-malleable with respect to all other operations. Joo and Yun first introduced *homomorphic authenticated encryption* (HAE) in [JY14] and defined IND-CCA security for HAE. They further demonstrated that the implication from IND-CPA plus INT-CTXT to IND-CCA holds in the homomorphic context. Akavia et al. introduced FuncCPA in [AGHV22], an intermediate notion between CPA and CCA2, to capture security for FHE schemes against attacks in the context of client-aided outsourcing. Manulis and Nguyen introduced the vCCA notion in [MN24], which is strictly stronger than CCA1, to address schemes where modifying the challenge ciphertext using homomorphism before querying it to its decryption oracle is detectable under certain conditions.

1.3 Contribution

In this work, we present a constructive analysis of homomorphic encryption (HE) and homomorphic authenticators (HA). From a composable perspective, our study not only elucidates the natural security objectives that HE and HA schemes should achieve but also evaluates their security composability within larger systems.

We treat HE and HA as symmetric primitives in a classical setting where a client outsources computation to a server, with an adversary positioned between them. We demonstrate that a confidential channel can be constructed using HE, and an authenticated channel can be created using HA from an initially insecure communication channel where an adversary can observe and inject messages into the channel.

Specifically, in the confidential channel, an adversary can only view the length of transmitted messages but can inject its own messages into the channel at will. This ensures that no message content is disclosed to the adversary during transmission or evaluation. Conversely, in the authenticated channel, an adversary can see all transmitted content but can only relay or delete messages between the client and server. This ensures that any results from the server remain unmodified by the adversary. We compare our construction of these channels to the existing game-based notions established for HE and HA, assessing whether they meet the desired security requirements.

We proceed to examine the serial composition of these channels to construct a secure channel that ensures both security and authentication, aligning with the Encrypt-then-MAC (EtM) composition method used in constructing Authenticated Encryption (AE) as outlined in [BN08]. This allows us to formally show that homomorphic authenticated encryption (HAE) can also be constructed through the generic composition of Homomorphic Encryption (HE) and Homomorphic Authentication (HA).

2 Preliminaries

2.1 Notation

We introduce the following notations for use throughout this paper. Let $\mathbb{N} = \{1, 2, \ldots\}$ denote the set of natural numbers. For each $n \in \mathbb{N}$, we define the set $[n] := \{1, \ldots, n\}$. Given a set S, we denote the set of all non-empty sequences of length at least n over S by $S^{\geq n} := \bigcup_{i \geq n} S^i$, and we define $S^+ := S^{\geq 1}$. Let $x = (x_1, \cdots, x_\ell) \in S^+$ with $\ell \in \mathbb{N}$ be a sequence. The length of x is denoted by $|x| := \ell$. For another sequence $y = (y_1, \ldots, y_{\ell'}) \in S^+$ with $\ell' \in \mathbb{N}$, the concatenation of x and y is defined as $x \parallel y = (x_1, \ldots, x_\ell, y_1, \ldots, y_{\ell'})$. When $S = \{0, 1\}$, such sequences are referred to as bit strings. Let $i \in \{0, 1, \ldots\}$; we denote the ℓ-bit string representation of i as $[i]_\ell$. The notation $S[a..b]$ represents the substring of S that includes indices from a to b. We use ε to denote the empty string, where $|\varepsilon| = 0$. We use $\vec{x} = \langle x_1, \ldots, x_\ell \rangle$ to denote a vector. A subvector $\vec{y}$ of $\vec{x}$ is denoted as $\vec{y} \subseteq \vec{x}$. The appendment of an element (or a vector) after $\vec{x}$ is denoted as $\vec{x} \bowtie \langle x' \rangle = \langle x_1, \ldots, x_\ell, x' \rangle$. Unless specified otherwise, we assume a vector $\vec{x}$ has length $|\vec{x}| = n$ and $\vec{x} = \langle x_1, \ldots, x_n \rangle$.

Let S be a finite set. We define the notation $x \leftarrow\!\!\$\, S$ to represent the selection of a value from the set S uniformly at random, which we then assign to the variable x. For an algorithm $\mathcal{A}$, we use the notation $y \leftarrow \mathcal{A}^{\mathrm{O}_1, \mathrm{O}_2, \cdots}$ to denote running $\mathcal{A}$ given access to oracles $\mathrm{O}_1, \mathrm{O}_2, \ldots$, and then assigning of the output of $\mathcal{A}$ to y. We use $y \twoheadleftarrow \mathcal{A}$ to indicate that y is the output of a probabilistic algorithm $\mathcal{A}$.

2.2 Game-Based Proof

In this work, we discuss some security notions that follow the code-based game-playing framework of Bellare and Rogaway [BR06]. This framework employs a game G composed of an *Initialization* procedure (Init), a *Finalization* procedure (Finalize), and a set of oracle procedures, whose number varies depending on the specific game. An adversary $\mathcal{A}$ interacts with these oracles, receiving responses to its queries through return statements specified in the oracle codes.

A game G begins with the Init procedure, followed by the adversary's interaction with the oracles. After making a series of oracle queries, the adversary halts and produces an *adversary output*. Subsequently, the Finalize procedure is executed to generate a *game output*. If no explicit finalization procedure is defined, the *adversary output* is considered the *game output*. We denote $\Pr[\mathcal{A}^{\textsc{Init}, \mathrm{O}_1, \mathrm{O}_2, \cdots} \Rightarrow b]$ as the probability that the adversary

$\mathcal{A}$ outputs a value b after the Init procedure and queries to oracles $\mathrm{O}_1, \mathrm{O}_2, \cdots$. We denote $\Pr[\mathrm{G}(\mathcal{A}) \Rightarrow b]$ as the probability that game G outputs b when adversary $\mathcal{A}$ plays game G. For simplicity, we define $\Pr[\mathrm{G}(\mathcal{A})] := \Pr[\mathrm{G}(\mathcal{A}) \Rightarrow 0]$. To simplify notation, we interchangeably use $\Delta_{\mathcal{A}}(\mathrm{O}_L; \mathrm{O}_R)$ and

$$\Delta_{\mathcal{A}}\begin{pmatrix}\mathrm{O}_L \\ \mathrm{O}_R\end{pmatrix} := \Pr[\mathcal{A}^{\mathrm{O}_L} \Rightarrow 0] - \Pr[\mathcal{A}^{\mathrm{O}_R} \Rightarrow 0]$$

to denote $\mathcal{A}$'s advantage in distinguishing between the oracles O_L and O_R.

We let $\mathbf{Adv}^{\mathrm{x}}_{\Pi}(\mathcal{A}_x)$ denote adversary $\mathcal{A}_x$'s advantage in breaking security notion X of a scheme Π. We say security notion X implies security notion Y, denoted $\mathsf{X} \rightarrow \mathsf{Y}$, if $\mathbf{Adv}^{\mathrm{y}}_{\Pi}(\mathcal{A}_y) \leq c \cdot \mathbf{Adv}^{\mathrm{x}}_{\Pi}(\mathcal{A}_x)$ for some constant $c > 0$.

3 Homomorphic Encryption and Authenticator

3.1 Labeled Program

Following the definitions presented in [GW13] by Gennaro and Wichs, we introduce the concept of a *labeled program*. This syntax helps in specifying which data is encrypted/authenticated and which data a program P should process, thereby simplifying our discussion.

Definition 1 (Labeled Program)**.** A labeled program is a tuple $P = (f, \lambda_1, \ldots, \lambda_n)$ where $f : \mathcal{M}^n \rightarrow \mathcal{M}$ is a *circuit*, and λ_i for $i \in [n]$ are *labels* for the inputs to f. The program P evaluates the circuit f on the inputs $(m_1, \ldots, m_n)$ associated with the labels $(\lambda_1, \ldots, \lambda_n)$, outputting $m_\star = f(m_1, \ldots, m_n) \in \mathcal{M}$.

Given labeled programs $P_1, \ldots, P_t$ and a circuit $g : \mathcal{M}^t \rightarrow \mathcal{M}$, a *composed program* $P_\star = g(P_1, \ldots, P_t)$ evaluates the circuit g using the outputs of $P_1, \ldots, P_t$.

For an input label λ and a canonical identity function g_{id}, the *identity program* with label λ is denoted as $I_\lambda = (g_{id}, \lambda)$. Any program $P = (f, \lambda_1, \ldots, \lambda_n)$ can be expressed as a composed program of n identity programs i.e., $P = f(I_{\lambda_1}, \ldots, I_{\lambda_n})$.

Next, we define when a program is *well-defined*. Informally, a program P is well-defined with respect to a set $\mathcal{Q}$ either if all its labels are in $\mathcal{Q}$, or if there exists a label not in $\mathcal{Q}$, the inputs associated with these labels are ignored by f, and the evaluation remains unaffected.

Definition 2 (Well-Definedness)**.** Let $\mathcal{Q} \subseteq \mathcal{L} \times \mathcal{M}$ be a set, and let $P = (f, \lambda_1, \ldots, \lambda_n)$ be a labeled program. Consider the following conditions:

1. For every $\lambda_i \in P$, there exists $(\lambda_i, m_i) \in \mathcal{Q}$ for some message $m_i \in \mathcal{M}$.
2. If there exists $\lambda_j \in P$ such that $(\lambda_j, m_j) \notin \mathcal{Q}$ for all $m_j \in \mathcal{M}$, then

   $$f(\{m_i\}_{(\lambda_i, m_i) \in \mathcal{Q}}) = f(\{m_i\}_{(\lambda_i, m_i) \in \mathcal{Q}} \cup \{m_j\}_{(\lambda_j, m_j) \notin \mathcal{Q}})$$

   for all $m_j \in \mathcal{M}$.

We say that P is well-defined with respect to $\mathcal{Q}$ if either condition 1 or 2 is satisfied, denoted by $\omega_{\mathcal{Q}}(P) = 1$.

3.2 Syntax

We first present the definition of a homomorphic encryption (HE) scheme in Definition 3. Note that we define an HE scheme as a symmetric primitive.

Definition 3 (Homomorphic Encryption)**.** A homomorphic encryption scheme with secret key space $\mathcal{SK}$, evaluation key space $\mathcal{EK}$, circuit space $\mathcal{F}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$, is a tuple $\mathsf{HE} = (\mathtt{Enc}, \mathtt{Dec}, \mathtt{Eval})$ of the following probabilistic polynomial time (PPT) algorithms:

- $\mathtt{Enc} : \mathcal{SK} \times \mathcal{M} \to \mathcal{C}$ encrypt a message m with a secret key sk and outputs a ciphertext $c \leftarrow \mathsf{HE}.\mathtt{Enc}_{sk}(m)$.
- $\mathtt{Dec} : \mathcal{SK} \times \mathcal{C} \to \mathcal{M}$ decrypts a ciphertext c and outputs the corresponding plaintext $m \leftarrow \mathsf{HE}.\mathtt{Dec}_{sk}(c)$.
- $\mathtt{Eval} : \mathcal{EK} \times \mathcal{F} \times \mathcal{C}^n \to \mathcal{C}$ evaluates a vector of ciphertext $\langle c_1, \ldots, c_n \rangle$ over a circuit f and outputs the evaluated ciphertext $c_\star \leftarrow \mathsf{HE}.\mathtt{Eval}^f_{ek}(\langle c_1, \ldots, c_n \rangle)$.

We define the following correctness for an HE scheme:

- *Encryption Correctness*: For all $m \in \mathcal{M}$, it has
$$\Pr[\mathsf{HE}.\mathtt{Dec}_{sk}(c) = m \mid (sk, ek) \leftarrow\$\ \mathcal{SK} \times \mathcal{EK}, m = \mathsf{HE}.\mathtt{Enc}_{sk}(m)] = 1.$$
- *Evaluation Correctness*: Fix any pair of keys $(sk, ek) \in \mathcal{SK} \times \mathcal{EK}$. Fix any circuit $f : \mathcal{M}^t \to \mathcal{M} \in \mathcal{F}$ and any tuple of message/ciphertext $\{(m_i, c_i)\}_{i=1}^t$ such that $c_i = \mathsf{HE}.\mathtt{Enc}_{sk}(m_i)$. If $m_\star = f(m_1, \ldots, m_t)$, and $c_\star = \mathsf{HE}.\mathtt{Eval}^f_{ek}(\langle c_1, \ldots, c_n \rangle)$, then it holds that $\mathsf{HE}.\mathtt{Dec}_{sk}(c_\star) = m_\star$.

Definition 4 (Homomorphic Authenticator)**.** A homomorphic authenticator (HA) with secret key space $\mathcal{SK}$, evaluation key space $\mathcal{EK}$, label space $\mathcal{L}$, circuit space $\mathcal{F}$, program space $\mathcal{P}$, message space $\mathcal{M}$, and tag space $\mathcal{T}$, is a tuple $\mathsf{HA} = (\mathtt{Tag}, \mathtt{Vfy}, \mathtt{Eval})$ of the following PPT algorithms:

- $\mathtt{Tag} : \mathcal{SK} \times \mathcal{L} \times \mathcal{M} \to \mathcal{T}$ produces a tag $\tau \leftarrow \mathsf{HA}.\mathtt{Tag}^\lambda_{sk}(m)$ that authenticates m under the label λ.
- $\mathtt{Vfy} : \mathcal{SK} \times \mathcal{P} \times \mathcal{M} \times \mathcal{T} \to \{0, 1\}$ checks if τ authenticates that m is the evaluation on previously authenticated labeled data over the program P and returns $b \leftarrow \mathsf{HA}.\mathtt{Vfy}^P_{sk}(m, \tau) \in \{0, 1\}$.
- $\mathtt{Eval} : \mathcal{EK} \times \mathcal{F} \times \mathcal{T}^n \to \mathcal{T}$ evaluates a vector of tags $\langle \tau_1, \ldots, \tau_n \rangle$ over a circuit f and produces a tag $\tau_\star \leftarrow \mathsf{HA}.\mathtt{Eval}^f_{ek}(\langle \tau_1, \ldots, \tau_n \rangle)$.

We define the following correctness for an HA scheme:

- *Authentication Correctness*: For any message-label tuple $(\lambda, m) \in \mathcal{L} \times \mathcal{M}$, it has
$$\Pr[\mathsf{HA}.\mathtt{Vfy}^{I_\lambda}_{sk}(m, \tau) = 1 \mid (sk, ek) \leftarrow\$\ \mathcal{SK} \times \mathcal{EK}, \tau = \mathsf{HA}.\mathtt{Tag}^\lambda_{sk}(m)] = 1$$
where I_λ is the identity program with respect to λ.
- *Evaluation Correctness*: Fix any pair of keys $(sk, ek) \in \mathcal{SK} \times \mathcal{EK}$. Fix any circuit $f : \mathcal{M}^t \to \mathcal{M} \in \mathcal{F}$ and any tuple of message/program/tag $\{(m_i, P_i, \tau_i)\}_{i=1}^n$ such that $\mathsf{HA}.\mathtt{Vfy}^{P_i}_{sk}(m_i, \tau_i) = 1$. If $m_\star = f(m_1, \ldots, m_n)$, $P_\star = f(P_1, \ldots, P_n)$, and $\tau_\star = \mathsf{HA}.\mathtt{Eval}^g_{ek}(\langle \tau_1, \ldots, \tau_n \rangle)$, then it holds that $\mathsf{HA}.\mathtt{Vfy}^{P_\star}_{sk}(m_\star, \tau_\star) = 1$.

4 Constructive Cryptography

4.1 Framework

In this work, we utilize the composable framework of *constructive cryptography* (CC) introduced by Maurer in [Mau11]. CC allows us to make statements about the construction of one resource from another. A *resource* $\mathbf{R}$ is a system with *interfaces* through which it interacts with its environment, typically assigned to specific parties. For our purposes, we consider three interfaces for all resources: a client A, a server B, and an adversary E positioned between them.

Security is modeled by the advantage of a distinguisher $\mathbf{D}$ in distinguishing between two resources $\mathbf{R}$ and $\mathbf{S}$, expressed as:

$$\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S}) = \Pr[\mathbf{DR} \Rightarrow 1] - \Pr[\mathbf{DS} \Rightarrow 1]$$

where $\Pr[\mathbf{DR} \Rightarrow 1]$ represents the probability that $\mathbf{D}$ outputs 1 when connected to the resource $\mathbf{R}$. Specifically, $\mathbf{DR}$ is a random experiment where $\mathbf{D}$ repeatedly interacts with one of the interfaces A, B, or E, observes the responses, and then decides on its output bit. In this work, we model communication channels between the client and the server using resources.

A *converter* cvt is a system connected to a resource's interfaces, modifying inputs and outputs at that interface and thus converting the resource into another resource. We denote this as $\mathsf{cvt}^{\mathsf{A}}\mathbf{R}$ when a converter is attached to the interface A of a resource $\mathbf{R}$. A converter has an *inner interface* in connected to a resource and an *outer interface* out which becomes the new connection point of the converted resource towards the environment.

Definition 5 (Construction)**.** Let $\mathbf{R}$ and $\mathbf{S}$ be resources, and let $\perp^{\mathsf{E}}$ be a converter representing when the adversary E performs no attack. Let sim be a simulator converter. A protocol, defined as a pair of converters $(\mathsf{cvt}_1, \mathsf{cvt}_2)$, securely constructs a resource $\mathbf{R}$ from a resource $\mathbf{S}$ if:

$$\Delta^{\mathbf{D}}(\mathsf{cvt}_1^{\mathsf{A}}\mathsf{cvt}_2^{\mathsf{B}}\perp^{\mathsf{E}}\mathbf{R}, \perp^{\mathsf{E}}\mathbf{S}) \leq \varepsilon(\mathbf{D}) \qquad (\textit{Availability})$$

and

$$\Delta^{\mathbf{D}}(\mathsf{cvt}_1^{\mathsf{A}}\mathsf{cvt}_2^{\mathsf{B}}\mathbf{R}, \mathsf{sim}^{\mathsf{E}}\mathbf{S}) \leq \varepsilon(\mathbf{D}) \qquad (\textit{Security})$$

where $\varepsilon(\mathbf{D}) \in [-1, 1]$ represents the distinguisher $\mathbf{D}$'s advantage in distinguishing between the two environments. We denote this construction as $\mathbf{R} \overset{(\mathsf{cvt}_1,\mathsf{cvt}_2,\varepsilon)}{\longrightarrow} \mathbf{S}$.

Given several constructions, we can define composability with respect to these constructions, specifically serial and parallel composability.

Theorem 1 ([Mau11, Theorem 1])**.** *The following composability properties hold for two constructions:*

1. *(*Serial Composability*)*

$$\mathbf{R} \overset{(\mathsf{cvt}_1,\mathsf{cvt}_2,\varepsilon)}{\longmapsto} \mathbf{S} \ \wedge \ \mathbf{S} \overset{(\mathsf{cvt}_1',\mathsf{cvt}_2',\varepsilon')}{\longmapsto} \mathbf{T} \implies \mathbf{R} \overset{(\mathsf{cvt}_1 \circ \mathsf{cvt}_1', \mathsf{cvt}_2 \circ \mathsf{cvt}_2', \varepsilon+\varepsilon')}{\longmapsto} \mathbf{T}$$

2. *(*Parallel Composability*)*

$$\mathbf{R} \overset{(\mathsf{cvt}_1,\mathsf{cvt}_2,\varepsilon)}{\longmapsto} \mathbf{S} \ \wedge \ \mathbf{R}' \overset{(\mathsf{cvt}_1',\mathsf{cvt}_2',\varepsilon')}{\longmapsto} \mathbf{S}' \implies \mathbf{R}||\mathbf{R}' \overset{(\mathsf{cvt}_1||\mathsf{cvt}_1', \mathsf{cvt}_2||\mathsf{cvt}_2', \varepsilon+\varepsilon')}{\longmapsto} \mathbf{S}||\mathbf{S}'$$

3. *(*Identity Composability*)*

$$\mathbf{R} \overset{(\mathsf{id},\mathsf{id},0)}{\longmapsto} \mathbf{R}$$

4.2 Converters & Insecure Channel

Insecure Channel. We consider a classical setting where HE and HA are used. In this setup, a client wishes to outsource computations to a server without revealing the data to be processed and the evaluated result from the server (to a third party). We assume the communication between the client and the server takes place over an insecure channel

```
Resource INS

Initalize
    $\mathcal{Q}_A, \mathcal{Q}_B \leftarrow$ empty FIFO queues

Interface A
    Input: $(\texttt{snd}, M_A)$
        $\mathcal{Q}_A.\texttt{enqueue}(M_A)$
        Output $M_A$ at E

Interface B
    Input: $(\texttt{snd}, M_B)$
        $\mathcal{Q}_B.\texttt{enqueue}(M_B)$
        Output $M_B$ at E

Interface E
    Input: dlv-vec
        $M_A \leftarrow \mathcal{Q}_A.\texttt{dequeue}()$
        Output $M_A$ at B
    Input: dlv-val
        $M_A \leftarrow \mathcal{Q}_B.\texttt{dequeue}()$
        Output $M_B$ at A
    Input: $(\texttt{inj-vec}, M_E)$
        $\mathcal{Q}_A.\texttt{replace}(M_A, M_E)$
        Output $M_E$ at B
    Input: $(\texttt{inj-val}, M_E)$
        $\mathcal{Q}_B.\texttt{replace}(M_B, M_E)$
        Output $M_E$ at B
```

Figure 1: An insecure channel (**INS**) resource. We use $\mathcal{Q}.\texttt{replace}(M, M_E)$ to represent M is first dequeued from a queue $\mathcal{Q}$ then M_E is enqueued to $\mathcal{Q}$.

INS, as shown in Figure 1. Here, A represents the client and B represents the server. Within this insecure channel, an adversary E can intercept all messages between A and B. Moreover, E can inject its own messages to "replace" those sent by A and B.

In this work, we assume that the channel interfaces are invoked in the sequence "A $\rightleftharpoons$ E $\rightleftharpoons$ B". Specifically, A first sends a message to E, then E sends a message to B, followed by B sending a message back to E, and finally, E sends a message back to A. We refer to this entire sequence as one *round* of communication. This simplifies our analysis, thus we do not address stateful security issues related to out-of-sync ciphertext delivery. In the following discussion, we will demonstrate how to construct a secure channel from this insecure one, ensuring both confidentiality and authenticity by combining HE and HA.

Key Resource. In Figure 2, we introduce $\mathbf{KEY}_{\mathcal{SK}\times\mathcal{EK}}$ as a resource that distributes the secret key *sk* to the client A and the (public) evaluation key *ek* to the server B and the adversary E. We assume that $\mathbf{KEY}_{\mathcal{SK}\times\mathcal{EK}}$ is both confidential and authenticated, meaning the secret key *sk* is not known by the adversary E, and the evaluation key *ek* cannot be replaced by it. For notation simplicity, we use **KEY** to represent this key resource.

```
Resource $\mathbf{KEY}_{\mathcal{SK}\times\mathcal{EK}}$

Initalization
    $(sk, ek) \leftarrow\!\$\; \mathcal{K}$

Interface A
    Input: getkey
        Output $sk$ at A

Interface E
    Input: getkey
        Output $ek$ at E

Interface B
    Input: getkey
        Output $ek$ at B
```

Figure 2: A key resource **KEY** that distributes the secret key *sk* to the client A and the evaluation (public) key *ek* to the server B and the adversary E.

Converters with HE and HA. We introduce client converters $\mathsf{cli}_{\mathrm{he}}$ and $\mathsf{cli}_{\mathrm{ha}}$, and server converters $\mathsf{srv}_{\mathrm{he}}$ and $\mathsf{srv}_{\mathrm{ha}}$ using homomorphic encryption HE and homomorphic authenticator HA respectively, as depicted in Figures 3 and 4.

- *Converters* $\mathsf{cli}_{\mathrm{he}}$ *and* $\mathsf{srv}_{\mathrm{he}}$: In the client converter $\mathsf{cli}_{\mathrm{he}}$, the outer interface out accepts a command $\mathtt{snd}$ to encrypt a vector of messages $\vec{m}$ to a vector of ciphertext $\vec{c}$ where $c_i = \mathsf{HE}.\mathtt{Enc}_{sk}(m_i)$ for $i \in [n]$. The converter then sends a circuit f and $\vec{c}$ to the channel **INS**. The inner interface in receives the evaluated ciphertext $c_\star$ from **INS**, decrypts it to obtain $m_\star$, and outputs $m_\star$ at interface A via its outer interface out.

  In the server converter $\mathsf{srv}_{\mathrm{he}}$, a circuit f and a ciphertext vector $\vec{c}$ are received from **INS** at its inner interface in. The converter evaluates $c_\star = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$, and outputs $\vec{c}$ to its interface out. The interface out then outputs a command $\mathtt{snd}$ to the interface B to output $c_\star$ to channel **INS**.

- *Converters* $\mathsf{cli}_{\mathrm{ha}}$ *and* $\mathsf{srv}_{\mathrm{ha}}$: In the client converter, $\mathsf{cli}_{\mathrm{ha}}$, the outer interface of accepts a command $\mathtt{snd}$ to authenticate a vector of messages $\vec{m}$ with tags $\vec{\tau}$ under the labels $\vec{\lambda}$ such that $\tau_i = \mathsf{HA}.\mathtt{Tag}^{\lambda_i}_{sk}(m_i)$ for $i \in [n]$. We assume that each λ_i does not repeat. The inner interface in of $\mathsf{cli}_{\mathrm{ha}}$ receives $(f, \vec{\lambda}, m_\star, \tau_\star)$ from **INS**, parses $P = (f, \vec{\lambda})$ as a labeled program and verifies if $\tau_\star$ authenticates $m_\star$ as the output of P. If valid, $m_\star$ is outputted at the interface A via outer interface out. Otherwise, $\perp$ is outputted to indicated invalidity.

  In the server converter $\mathsf{srv}_{\mathrm{ha}}$, a circuit f, a message vector $\vec{m}$, a tag vector $\vec{\tau}$, and a label vector $\vec{\lambda}$ are received from **INS** at the interface in. The converter evaluates the message $m_\star = f(\vec{m})$, and the tag $\tau_\star = \mathsf{HA}.\mathtt{Eval}^{f}_{ek}(\vec{\tau})$, and outputs the result to its outer interface out. The interface out then outputs a command $\mathtt{snd}$ to interface B to output $(f, \vec{\lambda}, m_\star, \tau_\star)$ to channel **INS**.

```
Converter $\mathsf{cli}_{\mathrm{he}}$

Initalize
    Output getkey → KEY
    $sk \leftarrow$ KEY

Interface out
    Input: $(\mathtt{snd}, (f, \vec{m}))$
        for $i = 1 \ldots n$ do
            $c_i \leftarrow \mathsf{HE}.\mathtt{Enc}_{sk}(m_i)$
        Output $(\mathtt{snd}, (f, \vec{c})) \rightarrow$ INS

Interface in
    Input: $c_\star \leftarrow$ INS
        $m_\star \leftarrow \mathsf{HE}.\mathtt{Dec}_{sk}(c_\star)$
        Output $m_\star$ at out
```

```
Converter $\mathsf{srv}_{\mathrm{he}}$

Initalize
    Output getkey → KEY
    $ek \leftarrow$ KEY

Interface in
    Input: $(f, \vec{c}) \leftarrow$ INS
        $c_\star \leftarrow \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$
        Output $c_\star$ at out

Interface out
    Input: $c_\star \leftarrow$ in
        Output $(\mathtt{snd}, c_\star) \rightarrow$ INS
```

Figure 3: Converters $\mathsf{cli}_{\mathrm{he}}$ and $\mathsf{srv}_{\mathrm{he}}$ for a client and a server attached to the channel **INS** using a homomorphic encryption scheme HE. For the first part of proof of Theorem 2 and 4, we assume the dot-boxed part of $\mathsf{cli}_{\mathrm{he}}$ is not executed.

```
Converter $\mathsf{cli}_{\mathrm{ha}}$

Initalize
    Output getkey $\to$ KEY
    $sk \leftarrow$ KEY

Interface out
    Input: $(\mathtt{snd}, (f, \vec{\lambda}, \vec{m}))$
        for $i = 1 \dots n$ do
            $\tau_i \leftarrow \mathsf{HA.Tag}_{sk}^{\lambda_i}(m_i)$
        Output $(\mathtt{snd}, (f, \vec{\lambda}, \vec{m}, \vec{\tau})) \to$ INS

Interface in
    Input: $(f, \vec{\lambda}, m_\star, \tau_\star) \leftarrow$ INS
        $P \leftarrow (f, \vec{\lambda})$
        $b \leftarrow \mathsf{HA.Vfy}_{sk}^{P}(m_\star, \tau_\star)$
        if $b = 1$ then
            Output $m_\star$ at out
        else
            Output $\perp$ at out
```

```
Converter $\mathsf{srv}_{\mathrm{ha}}$

Initalize
    Output getkey $\to$ KEY
    $ek \leftarrow$ KEY

Interface out
    Input: $(f, \vec{\lambda}, m_\star, \tau_\star) \leftarrow$ in
        Output $(\mathtt{snd}, (f, \vec{\lambda}, m_\star, \tau_\star)) \to$ INS

Interface in
    Input: $(f, \vec{\lambda}, \vec{m}, \vec{\tau}) \leftarrow$ INS
        $m_\star \leftarrow f(\vec{m})$
        $\tau_\star \leftarrow \mathsf{HA.Eval}_{ek}^{f}(\vec{\tau})$
        Output $(f, \vec{\lambda}, m_\star, \tau_\star)$ at out
```

Figure 4: Converters $\mathsf{cli}_{\mathrm{ha}}$ and $\mathsf{srv}_{\mathrm{ha}}$ for a client and a server attached to the channel **INS** using a homomorphic authenticator HA.

5 Confidentiality

5.1 Indistinguishability with IND-CPA

We define the IND-CPA security for HE as *real-or-random* security i.e., indistinguishability between the encryption of queried plaintext, and a random bitstring, following the definition in [Shr04]. Consider the IND-CPA game in Figure 5, we then define the advantage in Definition 6.

Definition 6 (IND-CPA Advantage)**.**

$$\mathbf{Adv}_{\mathsf{HE}}^{\text{IND-CPA}}(\mathcal{A}) := \Pr[\mathrm{G}_{\mathsf{HE}}^{\text{IND-CPA-0}}(\mathcal{A})] - \Pr[\mathrm{G}_{\mathsf{HE}}^{\text{IND-CPA-1}}(\mathcal{A})]$$

In [JY14], the notion of IND-CPA is defined by granting the adversary access to an oracle that performs honest encryption and a challenge oracle that implements left-or-right encryption. This method is syntactically complex and deviates from the standard security definitions used in many other works. Therefore, we redefine the security notion as real-or-random (RoR) security, which is stronger than left-or-right (LoR) security, as noted by Rogaway in [Rog11]. In this framework, the adversary's task is to distinguish between two scenarios: one where real encryption is returned and another where a random bitstring is returned, starting from the first query to Enc. This notion allows us to consider two types of indistinguishability:

- *Encryption Indistinguishability*: This notion ensures that the encryption of each

```
$\mathrm{G}^{\text{IND-CPA-0}}_{\mathsf{HE}}$   $\boxed{\mathrm{G}^{\text{IND-CPA-1}}_{\mathsf{HE}}}$

procedure Init
    $(sk, ek) \leftarrow\!\!\$\ \mathcal{SK} \times \mathcal{EK}$

procedure Finalize
    $b \twoheadleftarrow \mathcal{A}^{\textsc{Enc}}(ek)$
    return $b$

Oracle $\textsc{Enc}(m)$
    $c \twoheadleftarrow \mathsf{HE}.\mathtt{Enc}_{sk}(m)$
    $\boxed{c \leftarrow\!\!\$\ \{0,1\}^{\psi(|m|)}}$
    return $c$
```

Figure 5: IND-CPA game for a homomorphic encryption scheme HE. We assume the randomness (or noise) $\varepsilon \leftarrow\!\!\$\ \chi$ is sampled by the scheme following a distribution χ instead of queried by the adversary. The dot-boxed part is exclusive to $\mathrm{G}^{\text{IND-CPA-1}}_{\mathsf{HE}}$.

message is indistinguishable from a random bitstring, as determined by the behavior of the oracle Enc.

- *Strong Homomorphism*: This notion extends security to indistinguishability between encryption of evaluated plaintexts and the evaluation of ciphertexts. Specifically, $c_i \twoheadleftarrow \textsc{Enc}(m_i)$ for $i \in [n]$ can be either real encryption or a random bitstring, and the adversary $\mathcal{A}$ can freely evaluate $c_\star = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\langle c_1, \ldots, c_n \rangle)$. With this notion, we require that $c_\star$ is indistinguishable from $c'_\star \twoheadleftarrow \textsc{Enc}(m_\star)$ made in another query where $m_\star = f(m_1, \ldots, m_n)$. This ensures the adversary cannot distinguish between these two worlds by querying Enc with $m_\star$, thus ensuring the indistinguishability between $c_\star = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\overrightarrow{c})$ and $c'_\star = \mathsf{HE}.\mathtt{Enc}_{sk}(m_\star)$.

Remark 1 (*Comments on RoR Security*). In the current literature, IND-CPA security is primarily defined using LoR security. However, we argue that RoR security is attainable by some HE schemes and should be preferred over LoR security since it is stronger and it is also easier to adapt this notion to simulation-based proof. For instance, Halevi demonstrated in [Hal17, Lemma 7] that a GSW-like leveled HE scheme achieves this security. Additionally, other schemes such as FHEW [DM15], TFHE [CGGI20], and FINAL [BIP+22] are also capable of achieving this, provided they do have not certain structures (e.g., ciphertext modulus) that could inadvertently reveal the computation's progress through the ciphertext's format.

5.2 Construction for Confidential Channel

To clarify our objective in terms of confidentiality, we illustrate a confidential channel, denoted as **CONF**, in Figure 6. In the **CONF** channel, the client A inputs a vector of messages $\langle m_1, \ldots, m_n \rangle$ and a circuit f into the channel. The server B sends an evaluated message $m_\star = f(m_1, \ldots, m_n)$ back to A. During this process, the adversary E can observe only the lengths of the messages $|m_1|, \ldots, |m_n|$, the length $|m_\star|$, and the circuit f. The adversary E might choose to relay $(f, \langle m_1, \ldots, m_n \rangle)$ from A to B, or inject its own messages $(f', \langle m'_1, \ldots, m'_n \rangle)$ into the channel and send them to B. The adversary may also relay the message $m_\star$ from B to A, or inject its message $m'_\star$ and send it to A.

Remark 2 (*Comment on channel* **CONF**). In channel **CONF**, we assume that the server B can view and evaluate the messages $\langle m_1, \ldots, m_n \rangle$ using the circuit f. However, since we consider homomorphic encryption (HE) as a symmetric-key primitive in this context, B should not actually be able to see these messages to prevent the server from accessing the information. Therefore, in Lines 1 to 4 of Figure 6, we only describe the behavior of B evaluating the circuit. In **CONF**, any adversarial actions are attributed to the adversary

```
Resource CONF

Initalize
    $\mathcal{Q}_A, \mathcal{Q}_B \leftarrow$ empty FIFO queues

Interface A
    Input: $(\texttt{snd}, f, \vec{m})$
        $\mathcal{Q}_A.\texttt{enqueue}((f, \vec{m}))$
        Output $(f, |m_1|, \ldots, |m_n|)$ at E

Interface B
    Input: $(f, \vec{m})$
        $m_\star \leftarrow f(\vec{m})$
        $\mathcal{Q}_B.\texttt{enqueue}(m_\star)$
        Output $|m_\star|$ at E

Interface E
    Input: dlv-vec
        $(f, \vec{m}) \leftarrow \mathcal{Q}_A.\texttt{dequeue}()$
        Output $(f, \vec{m})$ at B
    Input: dlv-val
        $m_\star \leftarrow \mathcal{Q}_B.\texttt{dequeue}()$
        Output $m_\star$ at A
    Input: $(\texttt{inj-vec}, f', \vec{m}')$
        $\mathcal{Q}_A.\texttt{replace}((f, \vec{m}), (f', \vec{m}'))$
        Output $(f', \vec{m}')$ at B
    Input: $(\texttt{inj-val}, m'_\star)$
        $\mathcal{Q}_B.\texttt{replace}(m_\star, m'_\star)$
        Output $m'_\star$ at B
```

Figure 6: A confidential channel (**CONF**) resource. We use $\mathcal{Q}.\texttt{replace}(m, m')$ to represent m is first dequeued from a queue $\mathcal{Q}$ then m' is enqueued to $\mathcal{Q}$.

E. Indeed, if we can construct this channel with a protocol such that the adversary E can only observe the message length, then we can categorize any party from whom we conceal information as being such an adversary.

Theorem 2. *The protocol* $(\mathsf{cli}_{\mathrm{he}}, \mathsf{srv}_{\mathrm{he}})$ *constructs resource* **CONF** *from* **INS**||**KEY** *with respect to* $(\mathsf{dlv}, \mathsf{dlv})$ *and simulator* sim *as defined in Figure 7. More specifically, for any distinguisher* $\mathbf{D}$ *and for any adversary* $\mathcal{A}$*,*

$$\Delta^{\mathbf{D}}\left(\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathsf{dlv}^{\mathsf{E}}\mathbf{INS}||\mathbf{KEY}, \mathsf{dlv}^{\mathsf{E}}\mathbf{CONF}||\mathbf{KEY}\right) = 0 \tag{1}$$

and

$$\Delta^{\mathbf{D}}\left(\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathbf{INS}||\mathbf{KEY}, \mathsf{sim}^{\mathsf{E}}\mathbf{CONF}||\mathbf{KEY}\right) \leq \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{HE}}(\mathcal{A}) \tag{2}$$

Proof. We start by proving the security condition (2) by analyzing the input-output behaviors of both systems involved. For this part, we temporarily assume that the interface in of the converter $\mathsf{cli}_{\mathrm{he}}$ is disabled, i.e., no decryption is executed upon receiving the evaluated ciphertext $c_\star$ from the channel **INS**, since we focus only on *encryption confidentiality* with respect to the channel **CONF**.

- **On input** $(\texttt{snd}, (f, \vec{m}))$ **at interface** A:
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathbf{INS}||\mathbf{KEY}$, the converter $\mathsf{cli}_{\mathrm{he}}$ encrypts each message m_i to obtain $c_i = \mathsf{HE}.\mathtt{Enc}_{sk}(m_i)$ for $i \in [n]$. It then outputs $(\texttt{snd}, (f, \vec{c}))$ where $\vec{c} = \langle c_1, \ldots, c_n \rangle$ at A, sending $(f, \vec{c})$ to the channel **INS**. Consequently, $(f, \vec{c})$ is delivered to E.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{CONF}||\mathbf{KEY}$, the message lengths $\ell_1, \ldots, \ell_n$ are delivered to E. The simulator sim then samples a vector of bitstrings $\vec{c} = \langle c_1, \ldots, c_n \rangle$ such that $|c_i| = \psi(\ell_i)$ for $i \in [n]$, where $\psi : \mathbb{N} \to \mathbb{N}$ maps plaintext lengths to ciphertext lengths. The simulator outputs $(f, \vec{c})$ at E via its interface out.
- **On input** $(\texttt{snd}, m_\star)$ **or** $(f, \vec{m})$ **at interface** B:

Converter sim

```
Initalize
  $\mathcal{Q}_{cli}, \mathcal{Q}_{srv} \leftarrow$ empty FIFO queues
  Output $\mathtt{getkey} \rightarrow \mathbf{KEY}$
  $ek \leftarrow \mathbf{KEY}$

Interface out
  Input: $\mathtt{dlv\text{-}vec}$
    $(g, \vec{c}) \leftarrow \mathcal{Q}_{cli}.\mathtt{dequeue}()$
    Output $(\mathtt{inj\text{-}vec}, g, \vec{c}) \rightarrow \mathbf{CONF}$
  Input: $\mathtt{dlv\text{-}val}$
    $c_\star \leftarrow \mathcal{Q}_{srv}.\mathtt{dequeue}()$
    Output $(\mathtt{inj\text{-}val}, c_\star) \rightarrow \mathbf{CONF}$
  Input: $(\mathtt{inj\text{-}vec}, f, \vec{c})$
    $g \leftarrow \mathtt{parse}(\mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\cdot))$
    Output $(\mathtt{inj\text{-}vec}, g, c_\star) \rightarrow \mathbf{CONF}$

Interface in
  Input: $(f, \ell_1, \dots, \ell_n)$
    for $i = 1, \dots, n$ do
      $c_i \leftarrow\!\!\$\ \{0,1\}^{\psi(\ell_i)}$
    $g \leftarrow \mathtt{parse}(\mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\cdot))$
    $\mathcal{Q}_{cli}.\mathtt{enqueue}((g, \vec{c}))$
    Output$(f, \vec{c})$ at $\mathsf{out}$
  Input: $\ell_\star$
    $(g, \vec{c}) \leftarrow \mathcal{Q}_{cli}.\mathtt{dequeue}()$
    $c_\star \leftarrow g(\vec{c})$
    $\mathcal{Q}_{srv}.\mathtt{enqueue}(c_\star)$
    Output $c_\star$ at $\mathsf{out}$
```

Figure 7: A simulator converter sim attached to the interface E of the resource **CONF**.

- In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathbf{INS}||\mathbf{KEY}$, when $(f, \vec{c})$ is received from **INS**, the converter $\mathsf{srv}_{\mathrm{he}}$ evaluates $c_\star = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$, and outputs $(\mathtt{snd}, c_\star)$ at B, sending $c_\star$ to **INS**. Therefore, $c_\star$ is delivered to E.
- In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{CONF}||\mathbf{KEY}$, upon receiving the length of the evaluated message $\ell_\star$, the simulator sim extracts $(g, \vec{c})$ from the simulated client queue $\mathcal{Q}_{cli}$, where $\vec{c}$ is a vector of random bitstrings, and $g : \mathcal{EK} \times \mathcal{C}^n \rightarrow \mathcal{C}$ is parsed from $\mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\cdot)$ at the interface in of sim upon receiving $\ell_1, \dots, \ell_n$. The simulator then evaluates $c_\star = g(\vec{c})$, adds $c_\star$ to a simulated server queue $\mathcal{Q}_{srv}$, and outputs $c_\star$ at E.

- **On input $\mathtt{dlv\text{-}vec}$ at interface** E:
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathbf{INS}||\mathbf{KEY}$, the tuple $(f, \vec{c})$ is extracted from the client queue $\mathcal{Q}_A$ and delivered at interface B. The converter $\mathsf{srv}_{\mathrm{he}}$ attached to B then evaluates $\mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{CONF}||\mathbf{KEY}$, this process is simulated by extracting $(g, \vec{c})$ from the simulated client queue $\mathcal{Q}_{cli}$, where $\vec{c}$ is a vector of random bitstrings sampled at the simulator sim's interface in, and g is the parsed circuit from $\mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\cdot)$. The simulator then outputs $(\mathtt{inj\text{-}vec}, g, \vec{c})$ at E to deliver $(g, \vec{c})$ at interface B. Interface B then evaluates $g(\vec{c}) = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$ as defined by **CONF**.

- **On input $\mathtt{dlv\text{-}val}$ at interface** E:
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathbf{INS}||\mathbf{KEY}$, the element $c_\star$ is extracted from the server queue $\mathcal{Q}_B$ and delivered at interface A. This $c_\star$ is the evaluated result outputted by the converter $\mathsf{srv}_{\mathrm{he}}$, i.e., $c_\star = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$, where $(f, \vec{c})$ was obtained from the queue $\mathcal{Q}_A$.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{CONF}||\mathbf{KEY}$, this process is simulated by extracting $c_\star$ from the simulated server queue $\mathcal{Q}_{srv}$, where $c_\star = g(\vec{c}) = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$ for $(g, \vec{c})$

from the simulated client queue $\mathcal{Q}_{cli}$. The simulator then outputs $(\texttt{inj-val}, c_\star)$ at E to deliver $c_\star$ at interface A.

- **On input** `inj-vec`:
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\text{he}}\mathsf{srv}^{\mathsf{B}}_{\text{he}}\mathbf{INS}||\mathbf{KEY}$, the interface E outputs $(f, \vec{c})$ at interface B. Subsequently, the converter srv_{he} performs the evaluation $\mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{CONF}||\mathbf{KEY}$, the simulator translates $\mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\cdot)$ into a circuit g. Then, $(g, \vec{c})$ is delivered to B, where B computes $g(\vec{c}) = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$.

  Note that in this case, sim provides a perfect simulation.

- **On input** `inj-val`: In both systems, any message injected by the adversary E is directly forwarded to the interfaces A. In this case, sim provides a perfect simulation.

Observe that the input-output behaviors of the two system differs on the inputs $(\texttt{snd}, f, \vec{m})$, $(\texttt{snd}, m_\star)$, `dlv-vec`, and `dlv-val`. Specifically, on inputs $(\texttt{snd}, f, \vec{m})$ and `dlv-vec`, the two systems differ on $c_i \twoheadleftarrow \mathsf{HE}.\mathtt{Enc}_{sk}(m_i)$ and $c'_i \leftarrow\!\!\$ \{0,1\}^{\psi(|m_i|}$ for $i \in [n]$. On inputs $(\texttt{snd}, m_\star)$ and `dlv-val`, the two systems differ on $c_\star \leftarrow \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$ and $c'_\star \leftarrow \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c}')$. This exactly describes our definition of IND-CPA game in Figure 5. Then if there is a distinguisher $\mathbf{D}$ that distinguishes between these two systems, then we can use it to construct an IND-CPA adversary $\mathcal{A}$ by forwarding queries to get $c_i \twoheadleftarrow \textsc{Enc}(m_i)$ for $i \in [n]$ and running the evaluation on circuit f. Thus, Condition (2) holds.

We now establish the availability condition (1) assuming the interface in of the converter cli_{he} is now enabled. In system $\mathsf{cli}^{\mathsf{A}}_{\text{he}}\mathsf{srv}^{\mathsf{B}}_{\text{he}}\mathsf{dlv}^{\mathsf{E}}\mathbf{INS}||\mathbf{KEY}$, when the converter dlv is connected to interface E, any output from the channel $\mathbf{INS}$ triggers the inputs `dlv-vec` and `dlv-val`. Notably, any $(f, \vec{c})$ input into $\mathbf{INS}$ from cli_{he} is promptly delivered to srv_{he}. Likewise, any $c_\star$ input into $\mathbf{INS}$ from srv_{he} is immediately conveyed to cli_{he}. Consequently, if the i-th input at interface A is $(\texttt{snd}, f, \vec{m})$, then the i-th output back at interface A is $m_\star = f(m_1, m_2, \ldots, m_n)$, as ensured by the correctness of the HE scheme defined in Definition 3. It is also evident that the same input-output behavior applies to the system $\mathsf{dlv}^{\mathsf{E}}\mathbf{CONF}||\mathbf{KEY}$.

□

6 Authenticity

6.1 Unforgeability and Circuit Integrity

Unforgeability. We follow [GW13, CF13] to formalize the unforgeability of an HA scheme with EUF-CMA notion. We illustrate the EUF-CMA game in a homomorphic context in Figure 8 and define the EUF-CMA advantage in Definition 7. Compared with the game for classical MAC scheme, we introduce an additional requirement (or rather an assumption) in Line 1 such that no label is used for authenticating more than one message. Notably, as indicated in [CF13], such assumption is implicitly present in the HA construction by Gennaro Wichs in [GW13], as well as in all previous works on homomorphic signatures.

We allow the adversary to adaptively make query to a tag oracle Tag and a verification oracle Vfy. In Line 2 – 3 of oracle Vfy, we specify the winning condition for the game. In addition to a valid tag, we require that either the program is not well-defined with respect to $\mathcal{Q}$, that is, there is a label that has not been used for authentication, or m is not the correct evaluation of the messages that have been authenticated via oracle Tag. Otherwise, the adversary trivially wins the game.

```
$\mathrm{G}_{\mathsf{HA}}^{\text{EUF-CMA}}$

procedure INIT
  $(sk, ek) \leftarrow\$ \mathcal{SK} \times \mathcal{EK}$
  $\mathcal{Q} \leftarrow \emptyset$
  $\mathsf{win} \leftarrow 0$

Oracle TAG$(\lambda, m)$
  if $(\lambda, \cdot) \in \mathcal{Q}$ then
    return $\lightning$
  $\tau \twoheadleftarrow \mathsf{HA}.\mathtt{Tag}_{sk}^{\lambda}(m)$
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\lambda, m)\}$
  return $\tau$

procedure VFY$(P, m, \tau)$
  $b \leftarrow \mathsf{HA}.\mathtt{Vfy}_{sk}^{P}(m, \tau)$
  if $b = 1 \wedge (\omega_{\mathcal{Q}}(P) = 0$
    $\vee\, m \neq f(\{m_i\}_{(\lambda_i, m_i) \in \mathcal{Q}}))$ then
    $\mathsf{win} \leftarrow 1$
  return $b$

procedure FINALIZE
  $\mathcal{A}^{\text{TAG,VFY}}$
  return $\mathsf{win}$
```

Figure 8: EUF-CMA game for a homomorphic authenticator scheme HA. We use $\omega_{\mathcal{Q}}(P)$ to represent whether the labeled program P is well-defined with respect to $\mathcal{Q}$. We let $(\lambda, \cdot) \in \mathcal{Q}$ denote if λ has been used in a previous query.

Definition 7 (EUF-CMA Advantage)**.**

$$\mathbf{Adv}_{\mathsf{HA}}^{\text{EUF-CMA}}(\mathcal{A}) := \Pr[\mathrm{G}_{\mathsf{HA}}^{\text{EUF-CMA}}(\mathcal{A}) \Rightarrow 1]$$

```
$\mathrm{G}_{\mathsf{HA}}^{\text{INT-CIRC}}$

procedure INIT
  $(sk, ek) \leftarrow\$ \mathcal{SK} \times \mathcal{EK}$
  $f \leftarrow\$ \mathcal{F}$

Oracle TAG$(\vec{\lambda}, \vec{m})$
  $\vec{\tau} \leftarrow \langle\,\rangle$
  for $i = 1, \ldots, n$ do
    $\tau_i \twoheadleftarrow \mathsf{HA}.\mathtt{Tag}_{sk}^{\lambda_i}(m_i)$
    $\vec{\tau} \leftarrow \vec{\tau} \bowtie \langle\tau\rangle$
  return $\vec{\tau}$

procedure FINALIZE
  $(f', \vec{\lambda}', \vec{m}', \vec{\tau}') \twoheadleftarrow \mathcal{A}^{\text{TAG}}(f)$
    // $\vec{\lambda}' \subseteq \vec{\lambda}, \vec{m}' \subseteq \vec{m}, \vec{\tau}' \subseteq \vec{\tau}$
  $\tau'_\star \leftarrow \mathsf{HE}.\mathtt{Eval}_{ek}^{f'}(\vec{\tau}')$
  $P' \leftarrow (f', \vec{\lambda}')$
  $b \leftarrow \mathsf{HE}.\mathtt{Vfy}_{sk}^{P'}(f'(\vec{m}'), \tau'_\star)$
  if $f'(\vec{m}') \neq f(\vec{m})$ then
    return $b$
  else
    return $0$
```

Figure 9: INT-CIRC game for a homomorphic authenticator scheme HA.

CIRCUIT INTEGRITY. Current integrity notions, such as EUF-CMA depicted in Figure 8, INT-PTXT (which is the same as EUF-CMA), and INT-CTXT as detailed in [JY14], primarily concentrate on whether an adversary can forge a valid tag or ciphertext concerning an adversary-chosen circuit. However, the fundamental goal of integrity is to ensure the *correct evaluation* of the message. Given a tuple $(\vec{\lambda}, \vec{m}, \vec{\tau})$ and a circuit f, an adversary may not need to forge anything at all; instead, they could simply select a different circuit f' and perform an honest evaluation to produce misleading results. Indeed, to counteract such trivial attacks, the client can retain a copy of the circuit f for verification. Nonetheless, there are scenarios where f is part of the message, particularly when f is a temporary or ephemeral circuit created for a specific or one-time computational task. In these cases, ensuring the integrity of f is vital since the client relies on the circuit f being returned to validate the results.

To address this, we propose a stronger security notion where a tag τ is associated with a circuit f that is to be evaluated. We introduce a new notion, INT-CIRC, which assesses whether an adversary can construct a circuit f' that produces a different output compared to a predefined random circuit f from the circuit space, as illustrated in Figure 9. Specifically, the adversary is allowed to *non-adaptively* query a tag oracle TAG with a vector of *non-repeating* labels $\vec{\lambda}$ and a vector of messages $\vec{m}$. The oracle TAG then returns the corresponding vector of tags $\vec{\tau}$. To distinguish this notion from unforgeability, the adversary selects *subvectors* $\vec{\lambda}' \subseteq \vec{\lambda}$, $\vec{m}' \subseteq \vec{m}$, and $\vec{\tau}' \subseteq \vec{\tau}$. The adversary wins if $f'(\vec{m}') \neq f(\vec{m})$ and $\tau'_\star = \mathsf{HE.Eval}^{f'}_{ek}(\vec{\tau}')$ authenticates $f'(\vec{m}')$ as the output of the program $P' = (f', \vec{\lambda}')$. The INT-CIRC advantage is then defined in Definition 8.

Definition 8 (INT-CIRC Advantage)**.**

$$\mathbf{Adv}^{\text{INT-CIRC}}_{\mathsf{HA}}(\mathcal{A}) := \Pr[\mathrm{G}^{\text{INT-CIRC}}_{\mathsf{HA}}(\mathcal{A}) \Rightarrow 1]$$

6.2 Construction for Authenticated Channel

We then define a authenticated channel **AUTH** as illustrated in Figure 10. In the channel **AUTH**, the client A inputs into the channel a vector of messages $\langle m_1, \ldots, m_n \rangle$ and a circuit f. The server B computes $m_\star = f(m_1, \ldots, m_n)$ and sends $m_\star$ back to A. The adversary E sees the message content $m_1, \ldots m_n$ and $m_\star$, and the circuit f. The adversary E can only deliver $(f, \langle m_1, \ldots, m_n \rangle)$ from A to B, and deliver $m_\star$ from B to A.

Similarly as in Remark 2, we use interface B to describe the behavior that the circuit is evaluated on the server side. We assume the server honestly evaluate the circuit, and any adversarial behavior is attributed to E. In **AUTH**, we only allow the adversary to deliver the honestly evaluated message, which express the security goal with **AUTH**.

```
Resource AUTH

Initalize
  $\mathcal{Q}_A, \mathcal{Q}_B \leftarrow$ empty FIFO queues

Interface E
  Input: dlv-vec
      $(f, \vec{m}) \leftarrow \mathcal{Q}_A$.dequeue()
      Output $(f, \vec{m})$ at B
  Input: dlv-val
      $m_\star \leftarrow \mathcal{Q}_B$.dequeue()
      Output $m_\star$ at A

Interface A
  Input: $(\mathtt{snd}, f, \vec{m})$
      $\mathcal{Q}_A$.enqueue($(f, \vec{m})$)
      Output $(f, \vec{m})$ at E

Interface B
  Input: $(f, \vec{m})$
      $m_\star \leftarrow f(\vec{m})$
      $\mathcal{Q}_B$.enqueue($m_\star$)
      Output $m_\star$ at E
```

Figure 10: An authenticated channel (**AUTH**) resource.

Theorem 3. *The protocol* $(\mathsf{cli}_{\mathrm{ha}}, \mathsf{srv}_{\mathrm{ha}})$ *constructs resource* **AUTH** *from* **INS**||**KEY** *with respect to* $(\mathsf{dlv}, \mathsf{dlv})$ *and a simulator* sim *as defined in Figure 11. More specifically, for any distinguisher* $\mathbf{D}$ *and any adversary* $\mathcal{A}$*,*

$$\Delta^{\mathbf{D}}\left(\mathsf{cli}^{\mathsf{A}}_{\mathrm{ha}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{ha}}\mathsf{dlv}^{\mathsf{E}}\mathbf{INS}||\mathbf{KEY}, \mathsf{dlv}^{\mathsf{E}}\mathbf{AUTH}\right) = 0 \qquad (3)$$

and

$$\Delta^{\mathbf{D}}\left(\mathsf{cli}^{\mathsf{A}}_{\mathrm{ha}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{ha}}\mathbf{INS}||\mathbf{KEY}, \mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}\right) \leq \mathbf{Adv}^{\text{INT-CIRC}}_{\mathsf{HA}}(\mathcal{A}_1) + \mathbf{Adv}^{\text{EUF-CMA}}_{\mathsf{HA}}(\mathcal{A}_2) \tag{4}$$

```
Converter sim

Initalize
  $\mathcal{Q}_{cli} \leftarrow$ empty FIFO queues

Interface in
  Input: $(f', \vec{m}')$
      $(f, \vec{m}) \leftarrow \mathcal{Q}_A$.dequeue()
      $\mathcal{Q}_{cli}$.enqueue($(f, \vec{m})$)
      $\mathcal{Q}_A$.enqueue($(f', \vec{m}')$)
      Output $(f', \vec{m}')$ at out

  Input: $m'_\star$
      $\mathcal{Q}_B$.dequeue()
      $(f, \vec{m}) \leftarrow \mathcal{Q}_{cli}$.dequeue()
      if $m'_\star \neq f(\vec{m})$ then
          $\mathcal{Q}_B$.enqueue($\perp$)
      else
          $\mathcal{Q}_B$.enqueue($m'_\star$)
      Output $m'_\star$ at out
```

Figure 11: A simulator converter sim attached to interface E of $\mathbf{AUTH}$.

Remark 3. We first explain the intuition of the simulator in Figure 11. At the in interface of sim, we consider the tuple $(f', \vec{m}')$ and $m'_\star$ as either originating from A and B, or being injected by the adversary E in $\mathbf{INS}$. If $(f', \vec{m}')$ comes from A, it holds that $(f, \vec{m}) = (f', \vec{m}')$, where $(f, \vec{m})$ is the input provided by A to the channel. Similarly, $m_\star = m'_\star$ if $m'_\star$ is from B. Otherwise, $(f', \vec{m}')$ and $m'_\star$ are messages injected by E.

Proof. We begin by proving the security condition (4) by analyzing the input-output behaviors of both systems involved. Note that in the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{ha}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{ha}}\mathbf{INS}||\mathbf{KEY}$, the tuples $(\vec{\lambda}, \vec{m}, \vec{\tau})$ and $(f, \vec{\lambda}, m_\star, \tau_\star)$ are transmitted as messages since the protocol $(\mathsf{cli}_{\mathrm{ha}}, \mathsf{srv}_{\mathrm{ha}})$ is attached. However, in the system $\mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}$, we can only focus on the messages $\vec{m}$ and $m_\star$. We analyze the indistinguishability of the two systems from the perspective of $m_\star$ since the goal of $\mathbf{AUTH}$ is to ensure the honest evaluated of the circuit f.

- **On input** $(\mathtt{snd}, (f, \vec{m}))$ **at interface** A:
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{ha}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{ha}}\mathbf{INS}||\mathbf{KEY}$, the converter $\mathsf{cli}_{\mathrm{ha}}$ generates tags $\tau_i = \mathsf{HA}.\mathtt{Tag}^{\lambda_i}_{sk}(m_i)$ for $i \in [n]$ with unique labels λ_i. The converter then outputs $(\mathtt{snd}, (f, \vec{\lambda}, \vec{m}, \vec{\tau}))$ at A, sending $(f, \vec{\lambda}, \vec{m}, \vec{\tau})$ to the channel $\mathbf{INS}$, which delivers $(f, \vec{\lambda}, \vec{m}, \vec{\tau})$ to E.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}$, it is trivial to see that the simulator sim also outputs the same circuit-message tuple $(f, \vec{m})$ at interface E.
- **On input** $(\mathtt{snd}, m_\star)$ **or** $(f, \vec{m})$ **at interface** B:
  - In the system $\mathsf{srv}^{\mathsf{A}}_{\mathrm{ha}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{ha}}\mathbf{INS}||\mathbf{KEY}$, the converter $\mathsf{srv}_{\mathrm{ha}}$ evaluates $m_\star = f(\vec{m})$ and $\tau_\star = \mathsf{HA}.\mathtt{Eval}^{f}_{ek}(\vec{\tau})$ upon receiving $(f, \vec{\lambda}, \vec{m}, \vec{\tau})$ at in. It then outputs $(\mathtt{snd}, (f, \vec{\lambda}, m_\star, \tau_\star))$ at B to send $(f, \vec{\lambda}, m_\star, \tau_\star)$ to $\mathbf{INS}$, which delivers them to interface E.

  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}$, the interface B computes $m_\star = f(\vec{m})$ upon receiving $(f, \vec{m})$. The simulator sim then directly outputs $m_\star$ at E.

- **On input** $(\texttt{inj-vec}, f', \vec{m}')$ **at interface E (of INS):**
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{ha}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{ha}}\mathbf{INS}||\mathbf{KEY}$, the tuple $(f, \vec{\lambda}, \vec{m}, \vec{\tau})$ from A is replaced with $(f', \vec{\lambda}', \vec{m}', \vec{\tau}')$ from E in the client queue $\mathcal{Q}_A$, and delivered to B.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}$, the simulator receives the tuple $(f', \vec{m}')$ injected by E at its interface in. The simulator dequeues $(f, \vec{m})$, which is the tuple sent by A, from $\mathcal{Q}_A$. It then enqueues $(f', \vec{m}')$ from E into $\mathcal{Q}_A$, and outputs `dlv-vec` at E to deliver $(f', \vec{m}')$ to B.

- **On input** $(\texttt{inj-val}, m'_\star)$ **at interface E (of INS):**
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{ha}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{ha}}\mathbf{INS}||\mathbf{KEY}$, the tuple $(f, \vec{\lambda}, m_\star, \tau_\star)$ from B is replaced with $(f', \vec{\lambda}', m'_\star, \tau'_\star)$ from E in the server queue $\mathcal{Q}_B$, and delivered to A. Depending on whether $\tau_\star$ authenticates $m_\star$ as the output of the program $P = (f', \vec{\lambda}')$, the converter $\mathsf{cli}_{\mathrm{ha}}$ outputs $m'_\star$ or $\perp$ at A.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}$, the simulator receives $m'_\star$ injected by in at its interface E. The simulator dequeues $(f, \vec{m})$, which is the original tuple sent by A, from $\mathcal{Q}_{cli}$, and checks if $m'_\star = f(\vec{m})$. If true, it enqueues $m'_\star$ into $\mathcal{Q}_B$, otherwise, it enqueues $\perp$. The simulator then outputs `dlv-val` at E to deliver $m'_\star$ or $\perp$ to A.

- **On input `dlv-vec` at interface E:**
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{ha}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{ha}}\mathbf{INS}||\mathbf{KEY}$, the tuple $(f, \vec{\lambda}, \vec{m}, \vec{\tau})$ is extracted from the client queue $\mathcal{Q}_A$ and delivered to B.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}$, the content of $\mathcal{Q}_A$ remains unchanged since the same tuple is first dequeued then enqueued to $\mathcal{Q}_A$. Thus when then simulator outputs `dlv-val` at E, the same tuple $(f, \vec{m})$ enqueued into $\mathcal{Q}_A$ at interface A is extracted and delivered at interface B.

- **On input `dlv-val` at interface E:**
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{ha}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{ha}}\mathbf{INS}||\mathbf{KEY}$, the tuple $(f, \vec{\lambda}, m_\star, \tau_\star)$ is extracted from the server queue $\mathcal{Q}_B$ and delivered to A. Depending whether $\tau_\star$ authenticates $m_\star$ with respect to the program $P = (f, \vec{\lambda})$, the converter $\mathsf{cli}_{\mathrm{ha}}$ outputs $m_\star$ or $\perp$ at interface A.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}$, the tuple $(f, \vec{m})$ that is sent by A is first extracted from the queue $\mathcal{Q}_{cli}$. Then depending on whether $m_\star = f(\vec{m})$ or not, the message $m_\star$ or $\perp$ is outputted at interface A.

First, we analyze the output at interface A of the system $\mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}$. Let $(f, \vec{m})$ be the message input by A, and let $m_\star = f(\vec{m})$. In $\mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}$, any injected tuple $(f', \vec{m}')$ where $m_\star \neq f'(\vec{m}')$, or any injected message $m'_\star \neq f(\vec{m})$, results in $\perp$ at interface A.

In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{ha}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{ha}}\mathbf{INS}||\mathbf{KEY}$, the output is either $m_\star$ or $\perp$, depending on whether $\tau_\star$ authenticates $m_\star$ as the output of a labeled program $P = (f, \vec{\lambda})$. Let $(f, \vec{\lambda}, \vec{m}, \vec{\tau})$ be the message input by A, and let $m_\star = f(\vec{m})$. We observe that this system differs from $\mathsf{sim}^{\mathsf{E}}\mathbf{AUTH}$ when an evaluated message $m'_\star \neq m_\star$ is output at interface A instead of $\perp$. We assume the adversary E does not attempt to inject $m'_\star = f(\vec{m})$ with an invalid $\tau'_\star$, as this is trivial to accomplish. Now we can list the behaviors at interface E of **INS** as the following cases:

(1) **Case 1** $(\texttt{inj-vec}, (f', \vec{\lambda}, \vec{m}, \vec{\tau}))$ *then* $\texttt{dlv-val}$: The adversary E injects a different circuit $f' \neq f$ such that $m'_\star = f'(\vec{m}) \neq m_\star$. The converter srv_{ha} evaluates $m'_\star = f'(\vec{m})$ and $\tau'_\star = \mathsf{HA}.\mathtt{Eval}^{f'}_{ek}(\vec{\tau})$, which produces a valid tag $\tau'_\star$. Consequently, the converter cli_{ha} outputs $m'_\star$ at A instead of $\perp$.

(2) **Case 2** $(\texttt{inj-vec}, (f, \vec{\lambda}', \vec{m}', \vec{\tau}'))$ *then* $\texttt{dlv-val}$: The adversary's injection is not well-defined with respect to $\{(\lambda_i, m_i)\}_{\lambda_i \in \vec{\lambda}, m_i \in \vec{m}}$, which corresponds to the condition in Line 2 of oracle VFY in Figure 8. Specifically, the adversary must inject $\vec{m}' \neq \vec{m}$ such that $f(\vec{m}') \neq f(\vec{m}) = m_\star$, resulting in a message $m' \in \vec{m}'$ but $m' \notin \vec{m}$. The adversary must forge a tag τ' for m' under a new label $\lambda' \notin \vec{\lambda}$. The converter srv_{ha} honestly evaluates $m'_\star = f(\vec{m}')$ and $\tau'_\star = \mathsf{HA}.\mathtt{Eval}^{f}_{ek}(\vec{\tau}')$, which produces a valid tag $\tau'_\star$. Consequently, a message $m'_\star \neq m_\star$ is outputted at interface A.

(3) **Case 3** $(\texttt{inj-vec}, (f', \vec{\lambda}', \vec{m}', \vec{\tau}'))$ *then* $\texttt{dlv-val}$: This behavior can be viewed as a combination of Case 1 and Case 2. Thus, if the adversary makes a successful injection in either Case 1 or Case 2 (resulting in $m'_\star \neq m_\star$ being outputted at interface A), then it can make a successful injection in this case as well.

(4) **Case 4** $\texttt{dlv-vec}$ *then* $(\texttt{inj-val}, (f', \vec{\lambda}', m'_\star, \tau'_\star))$: In the previous step, the tuple $(f, \vec{\lambda}, \vec{m}, \vec{\tau})$ is outputted at interface E. The adversary can then change to a different circuit, evaluating $m'_\star = f'(\vec{m}')$ and $\tau'_\star = \mathsf{HA}.\mathtt{Eval}^{f'}_{ek}(\vec{\tau}')$, where $\vec{\lambda}' \subseteq \vec{\lambda}, \vec{m}' \subseteq \vec{m}$, and $\vec{\tau}' \subseteq \vec{\tau}$. Since $\tau'_\star$ is a valid tag, the converter cli_{ha} outputs $m'_\star \neq m_\star$ at A instead of $\perp$.

(5) **Case 5** $\texttt{dlv-vec}$ *then* $(\texttt{inj-val}, (f, \vec{\lambda}', m'_\star, \tau'_\star))$: In this case, E does not inject a new circuit. Instead, E injects a different $m'_\star \neq f(\vec{m}) = m_\star$. This corresponds to the condition in Line 3 of oracle VFY in Figure 8. Note that the adversary may or may not choose to inject $\vec{\lambda}' = \vec{\lambda}$, as the winning condition is defined with an OR statement. Thus it is sufficient for the adversary to inject an $m'_\star \neq m_\star$ and forge a valid tag $\tau'_\star$.

(6) **Case 6** $\texttt{inj-vec}$ *then* $\texttt{inj-val}$: Similarly, this behavior can be viewed as a combination of Cases 1 or 2. Thus, if the adversary makes a successful injection in Cases 1 and 2, then it can make a successful injection in this case as well.

We can then observe that if the adversary E injects a different circuit that yields a different result, yet the evaluated tag remains correct (Cases 1 and 4), this corresponds to an INT-CIRC adversary $\mathcal{A}_1$. Conversely, if the adversary provides a (vector of) message with a (vector of) valid tag (Cases 2 and 5), this corresponds to an EUF-CMA adversary $\mathcal{A}_2$. Also, the adversary might perform both actions. Thus, by Union Bound, we can bound the security by $\mathbf{Adv}^{\text{INT-CIRC}}_{\mathsf{HA}}(\mathcal{A}_1)$ plus $\mathbf{Adv}^{\text{EUF-CMA}}_{\mathsf{HA}}(\mathcal{A}_2)$, thereby concluding the Condition (4).

We now establish the availability condition (3). In system $\mathsf{cli}^{\mathsf{A}}_{\text{ha}}\mathsf{srv}^{\mathsf{B}}_{\text{ha}}\mathsf{dlv}^{\mathsf{E}}\mathbf{INS}||\mathbf{KEY}$, when the converter dlv is connected to interface E, any output from the channel $\mathbf{INS}$ triggers the inputs $\texttt{dlv-vec}$ and $\texttt{dlv-val}$. Notably, any $(f, \vec{m}, \vec{\tau})$ input into $\mathbf{INS}$ from cli_{ha} is promptly delivered to srv_{ha}. Likewise, any $(m_\star, \tau_\star)$ input into $\mathbf{INS}$ from srv_{ha} is immediately conveyed to cli_{ha} with $\tau_\star = \mathsf{HA}.\mathtt{Eval}^{f}_{ek}(\vec{\tau})$. Consequently, if the i-th input at interface A is $(\texttt{snd}, f, \vec{m})$, then the i-th output back at interface A is $m_\star = f(m_1, m_2, \ldots, m_n)$ since $\tau_\star$ authenticates $m_\star$ thus $m_\star$ is outputted at the interface A instead of $\perp$, as ensured by the correctness of the HE scheme defined in Definition 4. It is also evident that the same input-output behavior applies to the system $\mathsf{dlv}^{\mathsf{E}}\mathbf{AUTH}$. □

```
Resource SEC

Initalize
  $\mathcal{Q}_A, \mathcal{Q}_B \leftarrow$ empty FIFO queues

Interface A
  Input: $(\texttt{snd}, f, \vec{m})$
    $\langle m_1, \ldots, m_n \rangle \leftarrow \vec{m}$
    $\mathcal{Q}_A.\texttt{enqueue}((f, \vec{m}))$
    Output $(f, |m_1|, \ldots, |m_n|)$ at E

Interface B
  Input: $(f, \vec{m})$
    $\langle m_1, \ldots, m_n \rangle \leftarrow \vec{m}$
    $m_\star \leftarrow f(m_1, \ldots, m_n)$
    $\mathcal{Q}_B.\texttt{enqueue}(m_\star)$
    Output $|m_\star|$ at E

Interface E
  Input: dlv-vec
    $(f, \vec{m}) \leftarrow \mathcal{Q}_A.\texttt{dequeue}()$
    Output $(f, \vec{m})$ at B
  Input: dlv-val
    $m_\star \leftarrow \mathcal{Q}_B.\texttt{dequeue}()$
    Output $m_\star$ at A
```

Figure 12: A secure channel resource **SEC**.

7 Security Composition

Given a confidential channel and an authenticated channel, in accordance with *serial composability* as per Theorem 1, we should be able to construct a secure channel ensuring both confidentiality and authenticity, as depicted in Figure 12. In the channel **SEC**, the client A inputs a vector of messages $\langle m_1, \ldots, m_n \rangle$ and a circuit f into the channel. The server B computes $m_\star = f(m_1, \ldots, m_n)$ and returns $m_\star$ to A. The adversary E observes the lengths of messages $|m_1|, \ldots, |m_n|$, $|m_\star|$, and the circuit f. E can only intercept $(f, \langle m_1, \ldots, m_n \rangle)$ from A to B, and intercept $m_\star$ from B to A.

Security of EtM. We revisit the Encrypt-then-MAC (EtM) composition as discussed in [BN00]. We make a slight modification to the server converter $\mathsf{srv}_{\mathrm{he}}$, as illustrated in Figure 13. Specifically, the converter parses the evaluation algorithm $\mathsf{HE}.\mathtt{Eval}^f_{ek}(\cdot)$ into a circuit $g : \mathcal{EK} \times \mathcal{C}^n \to \mathcal{C}$. This adjustment is necessary because the interface B of **AUTH** requires a circuit and a vector of messages as inputs. Consequently, when $(f, \vec{c})$ is outputted at interface B, the converter makes B execute the evaluation algorithm as $g(\vec{c}) = \mathsf{HE}.\mathtt{Eval}^f_{ek}(\vec{c})$.

```
Converter $\mathsf{srv}_{\mathrm{he}}$

Initalize
  Output getkey $\to$ KEY
  $ek \leftarrow$ KEY

Interface in
  Input: $(f, \vec{c}) \leftarrow$ AUTH
    $g \leftarrow \texttt{parse}(\mathsf{HE}.\mathtt{Eval}^f_{ek}(\cdot))$
    Output $(g, \vec{c}) \to$ AUTH
```

Figure 13: Converter $\mathsf{srv}_{\mathrm{he}}$ attached to the interface B of channel **AUTH**.

Theorem 4. *The protocol* $(\mathsf{cli}_{\mathrm{he}}, \mathsf{srv}_{\mathrm{he}})$ *with* $\mathsf{cli}_{\mathrm{he}}$ *defined in Figure 3 and* $\mathsf{srv}_{\mathrm{he}}$ *defined in Figure 13 constructs resource* $\mathbf{SEC}||\mathbf{KEY}$ *from* $\mathbf{AUTH}||\mathbf{KEY}$ *with respect to* $(\mathsf{dlv}, \mathsf{dlv})$ *and a simulator* sim *as defined in Figure 14. More specifically, for any distinguisher* $\mathbf{D}$ *and any adversary* $\mathcal{A}$*,*

$$\Delta^{\mathbf{D}}\left(\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathsf{dlv}^{\mathsf{E}}\mathbf{AUTH}||\mathbf{KEY}, \mathsf{dlv}^{\mathsf{E}}\mathbf{SEC}||\mathbf{KEY}\right) = 0 \tag{5}$$

and

$$\Delta^{\mathbf{D}}\left(\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathbf{AUTH}||\mathbf{KEY}, \mathsf{sim}^{\mathsf{E}}\mathbf{SEC}||\mathbf{KEY}\right) \leq \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{HE}}(\mathcal{A}) \tag{6}$$

```
Converter sim

Initalize
  $\mathcal{Q}_{cli}, \mathcal{Q}_{srv} \leftarrow$ empty FIFO queues
  Output getkey $\rightarrow$ KEY
  $ek \leftarrow$ KEY

Interface out
  Input: dlv-vec
      $Q_A \leftarrow \mathcal{Q}_{cli}$
      Output dlv-vec $\rightarrow$ SEC
  Input: dlv-val
      $Q_B \leftarrow \mathcal{Q}_{srv}$
      Output dlv-val $\rightarrow$ SEC

Interface in
  Input: $(f, \ell_1, \ldots, \ell_n)$
      for $i = 1, \ldots, n$ do
          $c_i \leftarrow\$ \{0,1\}^{\psi(\ell_i)}$
      $g \leftarrow \mathtt{parse}(\mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\cdot))$
      $\mathcal{Q}_{cli}.\mathtt{enqueue}((g, \vec{c}))$
      Output $(f, \vec{c})$ at out
  Input: $\ell_\star$
      $(g, \vec{c}) \leftarrow \mathcal{Q}_{cli}.\mathtt{dequeue}()$
      $c_\star \leftarrow g(\vec{c})$
      $\mathcal{Q}_{srv}.\mathtt{enqueue}(c_\star)$
      Output $c_\star$ at out
```

Figure 14: A simulator converter sim attached to interface E of channel $\mathbf{SEC}$.

Proof. We will demonstrate the security condition (6) by analyzing the input-output behaviors of both systems involved. For this analysis, similar to the proof of Theorem 2, we will temporarily disable the interface in of the converter $\mathsf{cli}_{\mathrm{he}}$.

- **On input** $(\mathtt{snd}, f, \vec{m})$ **at interface** A:
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathbf{AUTH}||\mathbf{KEY}$, the converter $\mathsf{cli}_{\mathrm{he}}$ encrypts the message components to obtain ciphertexts $c_i = \mathsf{HE}.\mathtt{Enc}_{sk}(m_i)$ for $i \in [n]$. The converter then outputs $(\mathtt{snd}, f, \vec{c})$ at A, which sends $(f, \vec{c})$ to the channel $\mathbf{AUTH}$. Consequently, $(f, \vec{c})$ is delivered at E.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{SEC}||\mathbf{KEY}$, the lengths of the messages $\ell_1, \ldots, \ell_n$ are provided at E. The simulator converter sim samples a vector of bitstrings $\vec{c}$ such that $|c_i| = \psi(\ell_i)$ for $i \in [n]$, where $\psi : \mathbb{N} \rightarrow \mathbb{N}$ maps plaintext length to ciphertext length. Then $(f, \vec{c})$ is output to E via the out interface of sim.
- **On input** $(f, \vec{m})$ **at interface** B:
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathbf{AUTH}||\mathbf{KEY}$, when $(f, \vec{c})$ is received from $\mathbf{AUTH}$, the converter $\mathsf{srv}_{\mathrm{he}}$ parses the evaluation algorithm $\mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$ as a circuit g. Then B evaluates $c_\star = g(\vec{c}) = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{c})$ and outputs $c_\star$ at E.

- In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{SEC}||\mathbf{KEY}$, upon receiving the length of the evaluated message $\ell_\star$, the simulator sim extracts $(g, \overrightarrow{c})$ from the simulated client queue $\mathcal{Q}_{cli}$, where $\overrightarrow{c}$ is a vector of random bitstrings sampled and g is the parsed circuit from $\mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\cdot)$. The simulator then evaluates $c_\star = g(\overrightarrow{c})$ and adds $c_\star$ to the simulated server queue $\mathcal{Q}_{srv}$. Finally, $c_\star$ is output at E.

- **On input `dlv-vec` at interface** E:
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathbf{AUTH}||\mathbf{KEY}$, the tuple $(f, \overrightarrow{c})$ is extracted from the client queue $\mathcal{Q}_A$ and delivered at interface B. Since $\mathsf{srv}_{\mathrm{he}}$ is connected to B, the interface B parses and evaluates $g(\overrightarrow{c}) = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\overrightarrow{c})$.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{SEC}||\mathbf{KEY}$, the simulated client queue $\mathcal{Q}_{cli}$ contains $(g, \overrightarrow{c})$, where $\overrightarrow{c}$ is a vector of random bitstrings sampled at the simulator sim's interface in and g is the circuit parsed from $\mathsf{HE}.\mathtt{Eval}$. The simulator copies the contents of $\mathcal{Q}_{cli}$ to the actual client queue $\mathcal{Q}_A$. It then outputs `dlv-vec` at interface E to deliver the tuple $(g, \overrightarrow{c})$ at interface B. Subsequently, the interface B evaluates $g(\overrightarrow{c}) = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\overrightarrow{c})$.

- **On input `dlv-val` at interface** E:
  - In the system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathbf{AUTH}||\mathbf{KEY}$, the element $c_\star$ is extracted from the server queue $\mathcal{Q}_B$ and delivered at interface A. This $c_\star$ is the evaluated result output by the converter $\mathsf{srv}_{\mathrm{he}}$, i.e., $c_\star = g(\overrightarrow{c}) = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\overrightarrow{c})$, where $(f, \overrightarrow{c})$ was from the queue $\mathcal{Q}_A$.
  - In the system $\mathsf{sim}^{\mathsf{E}}\mathbf{SEC}||\mathbf{KEY}$, the simulated server queue $\mathcal{Q}_{srv}$ contains $c_\star$, where $c_\star = g(\overrightarrow{c}) = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\overrightarrow{c})$ for $(g, \overrightarrow{c})$ from the simulated client queue $\mathcal{Q}_{cli}$. The simulator outputs `dlv-val` at interface E to deliver $c_\star$ at interface A.

We observe that the input-output behaviors of the two systems differ only in the values of $\overrightarrow{c}$ and $c_\star$. As in the proof of Theorem 2, if there exists a distinguisher $\mathbf{D}$ that can distinguish between the two systems, we can construct an IND-CPA adversary from $\mathbf{D}$.

We now establish the availability condition (5) assuming the interface in of $\mathsf{cli}_{\mathrm{he}}$ is enabled. In system $\mathsf{cli}^{\mathsf{A}}_{\mathrm{he}}\mathsf{srv}^{\mathsf{B}}_{\mathrm{he}}\mathsf{dlv}^{\mathsf{E}}\mathbf{AUTH}||\mathbf{KEY}$, when the converter dlv is connected to interface E, any output from the channel $\mathbf{INS}$ triggers the inputs `dlv-vec` and `dlv-val`. Notably, any $(f, \overrightarrow{c})$ input into $\mathbf{AUTH}$ from $\mathsf{cli}_{\mathrm{he}}$ is promptly delivered to B. Likewise, any $c_\star$ input into $\mathbf{AUTH}$ from B is immediately conveyed to $\mathsf{cli}_{\mathrm{he}}$. Consequently, if the i-th input at interface A is $(\mathtt{snd}, f, \overrightarrow{m})$, then the i-th output back at interface A is $m_\star = f(m_1, m_2, \ldots, m_n)$, as ensured by the correctness of the HE scheme defined in Definition 3. It is also evident that the same input-output behavior applies to the system $\mathsf{dlv}^{\mathsf{E}}\mathbf{SEC}||\mathbf{KEY}$. □

Remark 4 (Ciphertext Integrity). Note that in Line 2 of the converter $\mathsf{srv}_{\mathrm{he}}$ in Figure 13, the converter parses the evaluation algorithm $\mathsf{HE}.\mathtt{Eval}$ as another circuit $g : \mathcal{EK} \times \mathcal{C}^n \to \mathcal{C}$. We know that the channel $\mathbf{AUTH}$ guarantees the the honest evaluation of g. This mean this composition gurantees $c_\star = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\overrightarrow{c})$ where $(f, \overrightarrow{c})$ is sent from the client, capturing the homomorphic INT-CTXT security as discussed in [JY14].

8 Future Works

8.1 Analysis in Public-Key Setting:

In this work, we study Homomorphic Encryption (HE) and Homomorphic Authentication (HA) as symmetric primitives, which are more often used in real-world applications like cloud computing. Initially in [C+09], HE is defined as a public-key primitive where encryption is performed using a public key. However, it is essential to also examine these primitives within a public-key framework to gain insights into constructing *Homomorphic Secure Communication* (HSC) in both symmetric and public-key settings. This research specifically investigates the integration of HE and HA. In a public-key context, this should involve examining the combination of HE and Homomorphic Signature (HS) instead.

Specifically, in Section 6.2, we define the interface B, representing the server, to consistently compute $f(\vec{m})$ regardless of whether the input originates from the client A or an adversary E, since the server B cannot verify the authenticity of $\vec{m}$. Conversely, with HS, the authenticity of messages $\vec{m}$ can be publicly verified. Thus, when a converter $\mathsf{cli}_{\mathrm{hs}}$ is attached, the interface B should be capable of rejecting evaluations of messages that do not originate from A.

Furthermore, several studies, including [Via23, MN24], have explored the integration of HE with *succinct non-interactive argument of knowledge* (SNARK) to verifiably control the evaluation algorithm. The construction presented in [MN24] follows the Naor-Yung double encryption paradigm, aiming to build a CCA2-secure [BDJR97] FHE. It is worth investigating whether this construction has achieved, or can be extended to achieve, IND-CCA3 security [Shr04], which is equivalent to AE security. At a high level, the channel **AUTH** should be constructible with a SNARK from **INS**, and attaching the protocol $(\mathsf{cli}_{\mathrm{he}}, \mathsf{srv}_{\mathrm{he}})$ should similarly enable the construction of a secure channel.

8.2 Distribution of Circuit and Labels

In channel **INS**, the circuit f is part of the communication between the client A and the server B. Alternatively, f may be pre-shared between the client and the server, thus not included in the message transmission. Consequently, we should consider the channels $\widehat{\mathbf{CONF}}$, $\widehat{\mathbf{AUTH}}$, and $\widehat{\mathbf{SEC}}$, where only $\vec{m}$ and $m_\star$ are transmitted. To address this, we introduce an additional resource **CIRC**, which ensures both confidentiality and authenticity, to distribute the circuit f at interfaces A and B, similar to the resource **KEY** depicted in Figure 2. For confidentiality, if we have

$$\mathbf{INS}||\mathbf{KEY}||\mathbf{CIRC} \overset{(\mathsf{cli}_{\mathrm{he}},\mathsf{srv}_{\mathrm{he}})}{\longrightarrow} \widehat{\mathbf{CONF}}||\mathbf{KEY}$$

then the scheme HE should achieve *circuit privacy* as introduced in [C+09].

For authenticity, note that the channel $\widehat{\mathbf{AUTH}}$ is equivalent to the channel **AUTH** defined in Figure 10, as the adversary can only deliver the circuit f to the server. In this scenario, we need to consider a different protocol, $(\mathsf{cli}_{\mathrm{he}}, \mathsf{srv}_{\mathrm{he}})$, and a simulator, sim, since the adversary E can no longer inject a different circuit f' into the channel **INS**. Then the channel $\mathbf{AUTH}'$ should capture the equivalent security as EUF-CMA.

Additionally, observe that in the protocol $(\mathsf{cli}_{\mathrm{ha}}, \mathsf{srv}_{\mathrm{ha}})$, we transmit $\vec{\lambda}$ as part of the message. The purpose of this setting is to align with the EUF-CMA game. However, in practice, the labels $\vec{\lambda}$ do not need to be transmitted, similar to the circuit. This is because the server does not require $\vec{\lambda}$ for evaluation, and the client can verify by retrieving the labels locally. Such a protocol attached to **AUTH** captures a slightly weaker notion than EUF-CMA but is still practical in real-world applications.

8.3 More on Circuit Integrity

In Section 6, we discuss circuit integrity by focusing on the correctness of the evaluated result. We refer to this form of security as *Type-I Circuit Security.* However, there is a special case where a different circuit $f'(\vec{m}') = f(\vec{m})$. For example, if $\vec{m}' = (\vec{m}, \vec{0})$ and f' is obtained from f by adding extra `add` gates, or if $\vec{m}' = (\vec{m}, \vec{1})$ and f' is derived from f by adding extra `mult` gates. According to [CF13, JY14], such a forgery with adversary-chosen circuit f is not considered a win since a tag $\tau_\star = \mathsf{HE}.\mathtt{Eval}^{f}_{ek}(\vec{\tau})$ that authenticates $m_\star$ as output of $P = (f, \vec{\lambda})$ may also authenticate $\vec{m}'$ as the output of $P = (f, \vec{\lambda}')$.

In this case, an adversary might inject an alternative circuit $(f', \vec{m}')$ to produce a correctly evaluated result $m_\star$ for the server. Although the result remains correct, the computation could be more complex and time-consuming, potentially increasing the client's costs if computation time is correlated with cost. Therefore, ensuring that the *exact same* circuit is evaluated, particularly when the circuit has not been pre-shared, is also important. We refer to this security property as *Type-II Circuit Security.*

In [GW13], Gennaro and Wichs discussed the construction of a hash tree g^H for a circuit g, which is a *Merkle Tree* mirroring the circuit's structure, but replaces internal gates with a *collision-resistant* hash function H. Specifically, to authenticate a message m under a label λ, an input ν to g^H is computed as $\nu = F_K(\lambda)$, where $F_K(\cdot)$ is a *keyed* PRF, and ν is included as part of the tag τ. This approach intuitively supports Type-II circuit integrity because adding the inputs or gates to the circuit would necessitate access to the key K to generate valid inputs for g^H.

Nonetheless, an adversary could still alter the gates within the circuit (e.g., changing an `add` gate to a `mult` gate), thereby compromising Type-I circuit integrity, as illustrated in Case 1 of the proof of Theorem 3. Loftus et al., in [LMSV12], represent the message space as a *ring* $(\mathcal{M}, +, \times)$ and the ciphertext space as a ring $(\mathcal{C}, \oplus, \otimes)$. By incorporating gate information into the labels when computing the inputs, such as $\nu = F_K(+, \lambda)$ or $\nu = F_K(\times, \lambda)$, this issue might be mitigated.

Furthermore, partial evaluation of g' and selection of a subvector of $\vec{\lambda}$ may still allow the violation of Type-I circuit integrity, akin to Case 4 in the proof of Theorem 3. An enhancement could involve incorporating circuit information when computing each input ν, such as $\nu = F_K(g, \lambda)$. If an adversary injects a different circuit g', each input hash $\nu' = F_K(g', \lambda)$ will differ. Consequently, the root hash $g'^H(\nu_1, \dots, \nu_n)$ injected by the adversary will differ from the reconstructed root hash $g'^H(\nu'_1, \dots, \nu'_n)$, allowing for the detection of such circuit injections. We propose to investigate and design a *designated-circuit* HA scheme to address this issue in future work.

9 Conclusion

In this study, we revisited homomorphic encryption (HE) and homomorphic authenticators (HA) from a constructive perspective. This approach enabled us to identify the fundamental security goals that HE and HA should achieve without relying on complex game-based notions. By comparing these notions with our security channels, we demonstrated whether the game-based notions achieved the desired security or not.

Additionally, we analyzed the serial composition of HE and HA, corresponding to the Encrypt-then-MAC (EtM) composition. This analysis allowed us to formally demonstrate that EtM, as a generic composition, can also be used to construct homomorphic authenticated encryption (HAE) in the presence of message evaluation.

Our work highlights the importance of composable security in the design of cryptographic primitives. By treating HE and HA as building blocks, we illustrated how secure communication channels can be constructed, a process that is less clear with game-based notions. We also provide insights into future work, including analysis in the public-key setting with the composition of homomorphic encryption and homomorphic signatures (HS), aiming to construct homomorphic secure communication (HSC) in both symmetric and public-key settings.

In conclusion, this research bridges the gap between existing formalism on the composition property of homomorphic cryptographic primitives, offering insights into future work on constructing homomorphic secure communication through the composition of these primitives.

References

[AB09] Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-based integrity for network coding. pages 292–305, 2009. doi:10.1007/978-3-642-01957-9_18.

[AGHV22] Adi Akavia, Craig Gentry, Shai Halevi, and Margarita Vald. Achievable CCA2 relaxation for homomorphic encryption. pages 70–99, 2022. doi:10.1007/978-3-031-22365-5_3.

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. pages 394–403, 1997. doi:10.1109/SFCS.1997.646128.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. pages 169–188, 2011. doi:10.1007/978-3-642-20465-4_11.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. pages 309–325, 2012. doi:10.1145/2090236.2090262.

[BIP$^+$22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: Faster FHE instantiated with NTRU and LWE. pages 188–215, 2022. doi:10.1007/978-3-031-22966-4_7.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. pages 531–545, 2000. doi:10.1007/3-540-44448-3_41.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. 21(4):469–491, October 2008. doi:10.1007/s00145-008-9026-x.

[BP23] Michael Brand and Gaëtan Pradel. Practical privacy-preserving machine learning using fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/1320, 2023. https://eprint.iacr.org/2023/1320. URL: https://eprint.iacr.org/2023/1320.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. pages 409–426, 2006. doi:10.1007/11761679_25.

[C$^+$09] Gentry Craig et al. A fully homomorphic encryption scheme. *Diss. Stanford University*, 2009.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. pages 336–352, 2013. doi:10.1007/978-3-642-38348-9_21.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. 33(1):34–91, January 2020. doi:10.1007/s00145-019-09319-x.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. pages 409–437, 2017. doi:10.1007/978-3-319-70694-8_15.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. pages 617–640, 2015. doi:10.1007/978-3-662-46800-5_24.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. pages 301–320, 2013. doi:10.1007/978-3-642-42045-0_16.

[Hal17] Shai Halevi. Homomorphic encryption. In *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pages 219–276. Springer, 2017.

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. pages 244–262, 2002. doi:10.1007/3-540-45760-7_17.

[JY14] Chihong Joo and Aaram Yun. Homomorphic authenticated encryption secure against chosen-ciphertext attack. pages 173–192, 2014. doi:10.1007/978-3-662-45608-8_10.

[LM21] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. pages 648–677, 2021. doi:10.1007/978-3-030-77870-5_23.

[LMSV12] Jake Loftus, Alexander May, Nigel P. Smart, and Frederik Vercauteren. On CCA-secure somewhat homomorphic encryption. pages 55–72, 2012. doi:10.1007/978-3-642-28496-0_4.

[Mau11] Ueli Maurer. Constructive cryptography – a new paradigm for security definitions and proofs. In S. Moedersheim and C. Palamidessi, editors, *Theory of Security and Applications (TOSCA 2011)*, volume 6993 of *Lecture Notes in Computer Science*, pages 33–56. Springer-Verlag, 4 2011.

[MN24] Mark Manulis and Jérôme Nguyen. Fully homomorphic encryption beyond IND-CCA1 security: Integrity through verifiability. pages 63–93, 2024. doi:10.1007/978-3-031-58723-8_3.

[PR08] Manoj Prabhakaran and Mike Rosulek. Homomorphic encryption with CCA security. pages 667–678, 2008. doi:10.1007/978-3-540-70583-3_54.

[RAD+78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Rog11] Phillip Rogaway. Evaluation of some blockcipher modes of operation. *Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan*, 630, 2011.

[Shr04] Tom Shrimpton. A characterization of authenticated-encryption as a form of chosen-ciphertext security. Cryptology ePrint Archive, Report 2004/272, 2004. https://eprint.iacr.org/2004/272.

[Via23] Alexander Viand. *Useable Fully Homomorphic Encryption.* PhD thesis, ETH Zurich, 2023.

[ZLL14] Feng Zhao, Chao Li, and Chun Feng Liu. A cloud computing security solution based on fully homomorphic encryption. In *16th International Conference on Advanced Communication Technology*, pages 485–488, 2014. doi:10.1109/ICACT.2014.6779008.