Homomorphic signcryption with public plaintext-result checkability

Citation for the original published paper (version of record):

Li, S., Liang, B., Mitrokotsa, A. et al (2021). Homomorphic signcryption with public plaintext-result checkability. IET Information Security, 15(5): 333-350. http://dx.doi.org/10.1049/ise2.12026

Received: 24 April 2019 | Revised: 4 August 2020 | Accepted: 3 February 2021

DOI: 10.1049/ise2.12026

IET Information Security

ORIGINAL RESEARCH PAPER

Homomorphic signcryption with public plaintext-result checkability

Shimin Li[1,2] | **Bei Liang**[3] | **Aikaterini Mitrokotsa**[4,5] | **Rui Xue**[2,6]

[1]Westone Cryptologic Research Center, Westone Information Industry Inc., Beijing, China

[2]State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China

[3]Yanqi Lake Beijing Institute of Mathematical Sciences and Applications, Beijing, China

[4]Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden

[5]School of Computer Science, University of St. Gallen, St. Gallen, Switzerland

[6]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Correspondence

Bei Liang, Yanqi Lake Beijing Institute of Mathematical Sciences and Applications, Beijing, 101408, China.
Email: lbei@chalmers.se

Funding information

VR PRECIS; National Natural Science Foundation of China, Grant/Award Number: 61472414, 61772514, 61602061, 61972124; National Key R&D Programme of China, Grant/Award Number: 2017YFB1400700

Abstract

Signcryption originally proposed by Zheng (CRYPTO′97) is a useful cryptographic primitive that provides strong confidentiality and integrity guarantees. This article addresses the question whether it is possible to homomorphically compute arbitrary functions on signcrypted data. The answer is affirmative and a new cryptographic primitive, homomorphic signcryption (HSC) with public plaintext-result checkability is proposed that allows both to evaluate arbitrary functions over signcrypted data and makes it possible for anyone to publicly test whether a given ciphertext is the signcryption of the message under the key. Two notions of message privacy are also investigated: weak message privacy and message privacy depending on whether the original signcryptions used in the evaluation are disclosed or not. More precisely, the contributions are two-fold: (i) two different definitions of HSC with public plaintext-result checkability is provided for arbitrary functions in terms of syntax, unforgeability and message privacy depending on if the homomorphic computation is performed in a private or in a public evaluation setting, (ii) two HSC constructions are proposed: one for a public evaluation setting and another for a private evaluation setting and security is formally proved.

1 | INTRODUCTION

Often in secure communications, one may want to send a message that is not only concealed (i.e., readable only by the intended receiver) but also authenticated (i.e., possible to verify that the message originates from the indicated sender) and it has not been modified while being transferred. Signcryption originally proposed by Zheng [1] is a useful cryptographic primitive that can provide strong confidentiality as well as authentication guarantees. Unlike naively combining two different cryptographic primitives (encryption and digital signatures), signcryption results in faster computation and shorter message expansion. It can be viewed as the public-key version of the symmetric-key primitive known as authenticated encryption.

More precisely, let us consider an example for signcryption in the two-user setting. In this case, the sender (Alice) generates her own secret signing and public verification key pair $(\mathsf{sk}_S, \mathrm{pk}_S)$ and the receiver (Bob) generates his own secret decryption and public encryption key pair $(\mathsf{sk}_R, \mathrm{pk}_R)$. The sender performs the signcryption algorithm by taking as input its secret key sk_S, the receiver's public key pk_R, and a message m and outputs a signcryption ciphertext C to the receiver. After receiving the signcryption C, the receiver performs unsigncryption on C by using his secret key sk_R and the sender's public key pk_S, and outputs either m or $\perp$, which indicates that the message was not encrypted or signed properly.

Motivated by the ground-breaking work of how to homomorphically perform arbitrary computations over encrypted data, known as fully homomorphic encryption (FHE) (e.g. [2–5]) as well as how to compute functions on signed data known as homomorphic signatures (HS) (e.g. [6–9]); we address the question whether it is possible to homomorphically compute arbitrary functions on signcrypted data in this article.

1.1 | Our Motivation

Rezeibagha et al. [10] introduced the concept of an additive homomorphic signcryption and gave a concrete construction, which is proven to be secure against chosen plaintext attacks (IND-CPA) and achieves weak unforgeability under the DDH assumption and the CDH assumption, respectively. However, the notion of homomorphic signcryption (HSC) proposed in [10], is only limited in additive operations and is not general to capture homomorphic evaluations for any function. Furthermore, Rezaeibagha et al.'s additive HSC scheme allows only private verification of the derived (homomorphically computed) signcryptions, that is, the verification can only be performed by the intended receiver. Moreover, it only achieves weak unforgeability, which requires that the adversary is not allowed to perform signcryption queries to the challenger.

In this article, we go beyond the additive homomorphic signcryption notion and introduce new primitive homomorphic signcryptions for any function, which allows computations for any function (arbitrary circuits) on the signcrypted data and provides public plaintext-result checkability for the derived signcryptions. We also investigate how to define the message privacy notion in the homomorphic evaluation setting.

Informally, an example of performing computations on signcrypted data can be described as follows (depicted in Figure 1). The sender (Alice) has a dataset $m_1, \ldots, m_k$ of size k, for instance, the weights of k persons. For convenience, we write $\vec{m} \coloneqq (m_1, \ldots, m_k)$. She independently signcrypts each data m_i associated with a tag and an index using her secret key sk_S and the receiver's (Bob) public key sk_R. Namely, Alice signcrypts the triple ('weight', m_i, i) for $i = 1, \ldots, k$ and obtains k independent signcryptions $c_1, \ldots, c_k$, where i is the index of m_i in the dataset and 'weight' serves as a tag that names the dataset. For simplicity, we write $\vec{c} \coloneqq (c_1, \ldots, c_k)$. However, restricted by limited computational resources and space, Alice uploads her signcryptions $\vec{c}$ to a powerful server and delegates the (expensive and complicated) computation of the function f to the server using an evaluation key ek. To compute a function f, the server employs the algorithm $\mathsf{Evaluate}$ that uses $\vec{c}$ and f to derive a signcryption $c_{f,y}$ on the triple: $(\text{"}\mathsf{weight}\text{"}, y := f(m_1, \ldots, m_k), \langle f \rangle)$, where $\langle f \rangle$ is a string describing the function f uniquely. Note that $\mathsf{Evaluate}$ does not require access to the original dataset $\vec{m}$ but only works on the signcryptions themselves.

Then, the server may send the derived signcryption $c_{f,y}$ to Bob. Bob using his secret key sk_R together with Alice's public key pk_S can obtain the message y. Given the pair (y, $c_{f,y}$), anyone can check with the public verification key vk that the server correctly applied f to the dataset by verifying that $c_{f,y}$ is a valid signcryption on a given triple $\left(\text{"}\mathsf{weight}\text{"}, f(\vec{m}), \langle f \rangle\right)$, without

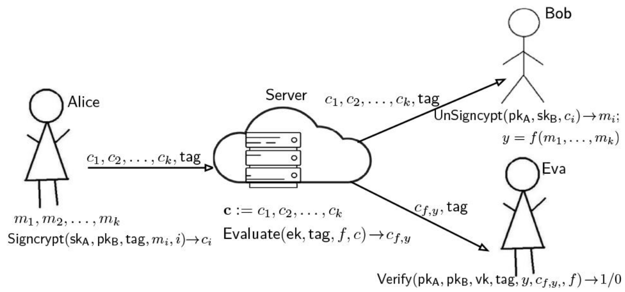

FIGURE 1 An example of using the homomorphic signcryption scheme

performing the expensive computations of the function f on the original dataset $\vec{m}$, to compute the value y. The verification on a pair (y, $c_{f,y}$) achieves public plaintext-result checkability by allowing anyone to test whether a derived signcryption $c_{f,y}$ is a signcryption of the result y which is the correct output of the computation f over Alice's data that is, $y = f(\vec{m})$. We describe the precise definitions of the syntax and the security later.

1.2 | Our contributions

In this article, we introduce for the first time the notion of homomorphic signcryption with public plaintext-result checkability that can evaluate arbitrary circuits over signcrypted data. We also define security notions in terms of both unforgeability and message privacy. We give two definitions for HSC depending on whether the homomorphic computation process is performed publicly (i.e., by anyone who has access to the public information) or privately (i.e., only by the one knowing the secret evaluation key), while in both cases the plaintext-result-checkability is performed publicly. For each case of public or private evaluation, we provide a construction and prove its security regarding unforgeability as well as message privacy.

Next, let us briefly describe the two important properties – namely, public plaintext-result-checkability and message privacy, which our homomorphic signcryption enjoys.

Public Plaintext-Result Checkability. It is to be noted that in a traditional signcryption scheme, by employing the unsigncryption algorithm (that uses the receiver's secret key) on the signcrypted data, either the original message $\vec{m}$ or $\perp$ is returned *. We notice that the signcryption allows a specific user having access to the receiver's secret key, to test whether a given signcryption originates from the given message under the sender's public key.

In our homomorphic signcryption primitive, we go beyond the plaintext-result checkability for a specified user, and provide a functionality of public plaintext-result checkability that anyone can test whether a (either derived or original) signcryption is the signcryption of a given message (possibly the outcome of a function) under the sender's public key. Looking ahead, we will see that such public plaintext-result checkability is achieved by a verification algorithm.

Message Privacy. Message privacy in the original (traditional) signcryption [1] means that even if the sender's private key is leaked to an attacker, the attacker cannot tell which message the signcryption challenge c^* is corresponding to, m_0 or m_1 that are chosen by the attacker. In our proposed HSC, we achieve a notion of message privacy, which guarantees that even if we publish the pair ($c_{f,y}$, y), where $c_{f,y}$ certifies y as the output of some computation f over a dataset $\overrightarrow{m}$, no information is revealed about the data $\overrightarrow{m}$, beyond what is revealed by y and f. Our definition for message privacy is presented in a way that given signcryptions on $\overrightarrow{m}_b$ (where $b \in \{0, 1\}$ and thus $\overrightarrow{m}_b$ is one of two different datasets $\overrightarrow{m}_0$, $\overrightarrow{m}_1$), the attacker cannot tell from which dataset the derived signcryptions $c_{f,y}$ came from, $f(\overrightarrow{m}_0)$ or $f(\overrightarrow{m}_1)$ for some function f known to the attacker.

However, to achieve a formal definition of message privacy for our HSC is a challenging problem because the public verification algorithm is able to operate on the signcryptions of messages for any function of the adversary's choice, which makes it easy for an adversary to find a function f^* such that $f^*(\overrightarrow{m}_0) \neq f^*(\overrightarrow{m}_1)$. More precisely, if the original signcryptions $\overrightarrow{c}_b$ over messages $\overrightarrow{m}_b$ are exposed to an adversary and the homomorphic operation and verification process are public algorithms, it is trivial for an adversary to distinguish the original dataset only from the information of $\overrightarrow{c}_b$ by performing a function f^* on $\overrightarrow{c}_b$ to get the derived signcryption $\widehat{c}$ on the message $f^*(\overrightarrow{m}_b)$. By testing the matching of $\widehat{c}$ with $f(\overrightarrow{m}_0)$ or $f(\overrightarrow{m}_1)$ via the public verification process, the attacker can tell which $\overrightarrow{c}_b$ comes from, $\overrightarrow{m}_0$ or $\overrightarrow{m}_1$.

In consequence to avoid such a trivial attack described above, we define two notions of message privacy–namely *weak* message privacy and standard message privacy for HSC with public verification. In the notion of *weak* message privacy, we assume that the original signcryptions $\overrightarrow{c}_b$ on the dataset $\overrightarrow{m}_b$ are not published (while the final signcryption is published), in which case the homomorphic operation is public. In the standard message privacy, we require $\overrightarrow{c}_b$ to be exposed to the adversary but the homomorphic operation is privately performed by the challenger, rather than being available to the adversary.

1.3 | Challenges in designing an HSC

To describe some challenges faced when designing an HSC scheme, let us consider a naive HSC scheme, which simply outputs the concatenation of a message x and a signature σ, that is, $x\|\sigma$ as the signcryption of x; where σ is a signature on x using an HS scheme. The publicly homomorphic evaluation for HSC on different elements $x\|\sigma$ proceeds by using the evaluation algorithm of HS to homomorphically compute the signature $\sigma_{f,y}$ for $y = f(\overrightarrow{x})$ in a straightforward manner. This appears to satisfy the unforgeability of HSC due to the unforgeability of HS. Furthermore, from the definition of the weak message privacy for HSC that the adversary will be given the derived signcryption $\sigma_{f,y}$ but not get access to the original signcryption $x\|\sigma$, it seems that the message x is not revealed. However, this is only true when the underlying HS is context-hiding, which means the derived signature $\sigma_{f,y}$ does not leak any information about $\overrightarrow{x}$. In other words, if the underlying HS is unforgeable and context-hiding, then the concatenation of a message and the corresponding signature is a trivial way to build a weak message private HSC scheme. Nevertheless, if given an HS that satisfies the basic homomorphism requirement, but not context hiding (which is an additionally advanced requirement, since to achieve it, either an additional assumption, i.e., existence of NIZKPoK (non-interactive zero-knowledge proof of knowledge), or a context-hiding homomorphic trapdoor function is needed [7]), a challenging question is how to address the issue of maintaining no leakage on $\overrightarrow{x}$ from $\sigma_{f,y}$. In this article, we propose a construction of HSC with weak message privacy in a public evaluation setting from HS scheme without the context-hiding property.

A similar intuition can be given for HSC schemes with private evaluation. In this case, we might consider a scheme similar to the one described above, but in which we set the signcryption to be the ciphertext of $x\|\sigma$ under a public key encryption (PKE) scheme, denoted as $c_{x,\sigma}$. Then, the evaluation key will correspond to the private key for the PKE, and the evaluator simply decrypts, and evaluates the HRs using the appropriate function f to obtain a derived signature $\sigma_{f,y}$, and then releases the result $y = f(\overrightarrow{x})$ concatenated with $\sigma_{f,y}$. Note that despite given access to the original signcrytion $c_{x,\sigma}$, the adversary still cannot learn the plaintext x since the employed PKE scheme is secure. Then as given above, this scheme appears to satisfy the normal message privacy for HSC with private evaluation. Unfortunately, such an assertion still relies on the context-hiding property of the underlying HS. The problem of building an HSC scheme with message privacy in a private evaluation setting arises when assuming the HS scheme is unforgeable but not context-hiding.

1.4 | Summary of our constructions

Let us now describe our solution for constructing HSC from an HS scheme without the context-hiding property. At first, we provide a construction of a homomorphic signcryption scheme with public plaintext-result-checkability in a public evaluation setting achieving unforgeability and weak message privacy, assuming the existence of HSsHS, PKE, indistinguishability obfuscation and statistical simulation-soundness non-interactive zero-knowledge proof (SSS-NIZK) for NP languages. The core idea of our construction is to apply the sequential composition method by signing the message using the underlying HS scheme first, then encrypting the concatenation of the message and the signature. The evaluation algorithm of the HS scheme naturally provides a way to homomorphically derive a signature $\sigma_{f,y}$ from the signature $\sigma_{\overrightarrow{x}}$ of the data $\overrightarrow{x}$, which certifies that y is the correct output of the computation f over the signed data $\overrightarrow{x}$. Nevertheless, the challenge is how to perform homomorphic evaluation over the encrypted signature (signcryption) $c_{\overrightarrow{x}} := \mathsf{Enc}(\sigma_{\overrightarrow{x}})$ of the data $\overrightarrow{x}$ without the decryption key. We employ indistinguishability obfuscation to resolve this problem by embedding the secret decryption key in an obfuscated programme, whose functionality is to decrypt the input signcryption so as to recover the underlying signatures first, and then from them to derive the signature σ_{fy} that corresponds to the message $y = f(\overrightarrow{x})$. Anyone can verify the tuple (f, y, σ_{fy}) using the HS's public verification algorithm, thus testing whether y is the correct output of the computation f over $\overrightarrow{x}$.

For an outsider adversary, who does not know the receiver's secret key, unforgeability can be achieved from the existential unforgeability property of the underlying HS scheme directly. Moreover, it is shown that our construction achieves weak message privacy, which requires that given not only the signcryptions on a number of messages derived from two different datasets for some function known to the attacker but also access to an unsigncryption oracle and the secret signcryption key, the attacker is not able to tell which dataset the derived signatures came from.

We then show how to use an HS together with a public key functional encryption (FE) to construct a homomorphic signcryption scheme with public plaintext-result-checkability in a private evaluation setting with unforgeability and message privacy. Employing the same sequential composition method of signing then encrypting, we instantiate with an HS scheme as before, while the underlying encryption is replaced with a public functional encryption scheme. To perform homomorphic evaluation over the *encrypted* signature (signcryption) $c_{\overrightarrow{x}} := \mathsf{Enc}(\sigma_{\overrightarrow{x}})$ of the data $\overrightarrow{x}$ in case of having no access to the decryption key, we take advantage of the functional secret key for a function whose functionality is the following: if the message-signature pair $(\overrightarrow{x}, \sigma_{\overrightarrow{x}})$ (i.e. underlay in $c_{\overrightarrow{x}}$) is verified, then it returns a function value y resulting from the function f applied on $\overrightarrow{x}$, as well as a signature $\sigma_{y,f}$ that is derived from the original signatures $\sigma_{\overrightarrow{x}}$ over the plaintext data $\overrightarrow{x}$ for a function f. This is achieved by employing the evaluation algorithm of HS and certifying the result value y. Furthermore, to publicly verify the validity of the signcryption $c_{y,f}$ on the data y, we use the function secret key for another function, whose functionality is to directly check the validity of (f, y, σ_{fy}) by using the HS's public verification algorithm.

For an outsider adversary, unforgeability can be achieved from the unforgeability of underlying HS directly. Moreover, we define the notion of message privacy of HSC in a private evaluation setting, which requires that given not only the signcryptions on a number of messages derived from two different datasets for some function known to the attacker, but also the original signcryption on both two datasets and access to the signcryption and evaluation oracles, the attacker is not able to tell which dataset the derived signatures came from. We prove that message privacy is preserved based on the IND-CPA security of the base functional encryption scheme.

Implementation and practical aspects. Due to the building blocks that our constructions are based on, it is inevitable to discuss the practicality of obfuscation and function encryption. As far as we know, there are several articles [11–16] investigating the practicality of programme obfuscation focussing on implementing and evaluating the performance of the obfuscators. Although the initial results required significant amounts of time to implement programme obfuscation [11, 12, 14], there are some recent advances that are very promising. More precisely, Cousins et al. [16] implemented obfuscation for conjunction programme satisfying distributional virtual black box (VBB) security. The obfuscation for a 64-bit conjunction programme takes 6.7 h and the evaluation takes only 3.5 s. Except software-only-based approaches to implement programme obfuscation, there are also some significant advances in hardware-based approaches. Nayak et al. [15] employed a hardware-based approach to implement it which achieves simulation-secure obfuscation for RAM programs, and improves performance significantly making an important step towards deploying obfuscation technology in practice.

Furthermore, Lewi et al. [13] also implemented the Boneh et al. [17] multi-input functional encryption scheme for 2-input comparison functions and 3-input DNF (3DNF) functions. Taking 3DNF function on 16-bit inputs with security parameter 80 as an example, it takes about 12 min to compute the encryption for three 16-bit inputs, yielding the overall ciphertexts with a size of 2.5 GB. Moreover, evaluating the 3DNF function value from the overall ciphertexts only takes 3 min.

Thus, in light of the promising future on the practicality of programme obfuscation and functional encryption, we believe that our proposed homomorphic signcryption schemes are very promising and advance significantly the area of signcryption.

1.5 | Application: certified data analysis

Working with sensitive data is often a balancing act between confidentiality and integrity concerns. One question on big data is how to release some socially beneficial results on private data, while minimising the information revealed about individuals. The nature of homomorphic signcryptions is that they provide public plaintext-result-checkability, confidentiality and integrity and they are very useful in a wide variety of settings involving data processing by untrusted entities.

As an example, consider the National Institute of Health (NIH) as the role of the curator which has some sensitive medical information from a set of subjects, which various research groups (analysts) wish to examine in detail to draw conclusions from. This separation between a curator and an analyst reflects a common application scenario in large-scale studies on sensitive data, where the raw data is often hosted by a single organisation, and the data may be used multiple times by different groups for different purposes. The curator signcrypts the collected data $\overrightarrow{m}$ and distributes the signcrypted data $\overrightarrow{c}$ to various analysts for processing, so that the underlying sensitive data is authenticated and remain confidential. Some of these groups may have the intention to lie about the outcomes of their analysis. However, using homomorphic signcryptions, they can publish their analysis strategy (a function f), the claimed outcome of the analysis $y = f(x)$, and a signcryption c_{fy} that certifies the correctness of the outcome. This information can be released publicly and verified by anyone using a verification key published by the NIH. When performing the verification, the verifier neither needs to have access to the underlying data nor needs to communicate with the NIH or the research groups that performed the computation. Furthermore, if such signcryptions are made to be messaged private, then it is assured that they do not reveal additional information about the underlying data beyond the analysis results.

1.6 | Related work

Homomorphic Signcryption. Homomorphic signcryption initially introduced by Rezaeibagha et al. [10, 18] was limited in capturing only additive homomorphic operations on the signcrypted messages. Thus, it did not address the problem of constructing a fully homomorphic signcryption, whose homomorphic operation is able to perform any function or circuit of polynomial size. On the other hand, Rezaeibagha et al. studied the homomorphic signcryption within the framework of standard signcryption, where the unsigncryption can be seen as a verification algorithm designed to be performed only by a specific user having the unsigncryption key to test whether a given ciphertext is the signcryption of a given message. Therefore, in Rezaeibagha et al.'s approach message privacy is defined in the same flavour of the original signcryption scheme. In contrast, our work focusses on settling these two weaknesses on Rezaeibagha et al.' homomorphic signcryption. We not only go beyond the plaintext-result checkability for a specified user, and provide a public verification algorithm to achieve public plaintext-result checkability such that anyone can test whether a given ciphertext is the signcryption of a given message under the sender's public key; but also beyond the additive homomorphic signcryption notion and present a homomorphic signcryption for any function.

Linearly Homomorphic Authenticated Encryption with Public Verifiability. Linearly homomorphic authenticated encryption with public verifiability (LAEPuV), introduced by Catalano et al. [19], is a primitive approach that allows to authenticate computations on encrypted data, with the additional property that the correctness of the computations can be publicly verified. Catalano et al. have also provided an instantiation for LAEPuV in the random oracle model based on Paillier's cryptosystem. Struck et al. [20] proposed some improvements on Catalano et al.'s instantiated scheme by avoiding false negatives during the verification algorithm.

Syntactically, our notion of homomorphic signcryption bears resemblance with the primitive of LAEPuV [19] on the aspect of allowing both the homomorphic operation on the encrypted data and the decryption of the ciphertext resulted from the homomorphic evaluation. However, using our homomorphic signcryption scheme, the data is signcrypted. The main difference between our homomorphic signcryption and LAEPuV relies on the verification algorithm. More precisely, in LAEPuV [19, 20] the verification algorithm is defined as $\mathsf{Verify}(\mathsf{vk}, \mathsf{id}, c, f)$ and requires that the condition $\mathsf{Verify}(\mathsf{vk}, \mathsf{id}, c, f) = 1$ is equivalent to $\exists m,\ s.t.\ \mathsf{Decrypt}\ (\mathsf{sk}, \mathsf{id}, c, f) = m$. Such a correctness requirement implies that if the verification is successful, the verifier is convinced that c is an encryption of some message, but without knowing to which message c corresponds to. In contrast, the verification algorithm in our homomorphic signcryption is defined as $\mathsf{Verify}(\mathsf{pk}, \mathsf{tag}, m', c', f)$, which is employed to check the matching of a message m' with the corresponding signcryption c'. This also explains why we have different security requirements from LAEPuV. The latter only considers IND-CCA security, since their verification algorithm does not need the message as input, and in some sense it resembles the security model of an encryption scheme. However in our case, the verification algorithm itself takes the message as input, thus, we should consider the plaintext checkable attack. Therefore, we define not only the privacy for the message but also the unforgeability for the signcryption.

Furthermore, although the two instantiations of LAEPuV [19, 20] are very efficient (since the homomorphic operation on the encrypted message in LAEPuV is a linear function), their security is proved in the random oracle model. On the contrary, the security of our proposed signcryption primitive is proved in the standard model and can be employed for a more general function, that is, for any polynomial size circuit, which leads to sacrificing some efficiency.

Publicly Verifiable Computation. Intuitively, it seems that our homomorphic signcryption scheme implies a publicly verifiable computation (VC) scheme that protects the secrecy of the used dataset towards the server and the verifier. In fact, to the best of our knowledge, there are no verifiable computation schemes that simultaneously achieve input privacy and provide public verifiability for arbitrary functions. Our homomorphic signcryption as a building block provides an optional method to publicly achieve VC with input privacy.

VC schemes [21–23] based on fully homomorphic encryption naturally offer input-output privacy, because both the inputs and outputs are encrypted. However, they do not provide public verifiability. The VC schemes using homomorphic authenticators are more restrictive with respect to the supported function class, although some solutions even provide input privacy. Homomorphic authenticator-based VC schemes are all privately verifiable and do not provide input privacy with an exception of Fiore et al.'s [24] scheme. Fiore et al. [24] showed how to combine the homomorphic MACs of [25] with a FHE scheme to construct a VC scheme for multi-variate polynomials of degree 2 that offers input privacy; however, it is privately verifiable.

Another line on studying VC is based on functional encryption or functional signatures. Parno et al. [26] showed a public VC for a class of Boolean functions $\mathcal{F}$, namely functions with one-bit output, that can be constructed from any attribute-based encryption (ABE) scheme for a related class of functions–namely, $\mathcal{F} \cup \mathcal{F}$ where $\mathcal{F}$ denotes the complement of the function $\mathcal{F}$. If the underlying ABE scheme is attribute hiding, then the VC scheme provides input privacy since the function's input is encoded in the attribute. However, the constructions for attribute hiding ABE is under way. Another very interesting approach is to build VC from functional signatures (FS) introduced by Boyle et al. [27]. Given an FS scheme with signature size $\delta(\lambda)$, and verification time $t(\lambda)$ (where λ is the security parameter), we can get a VC scheme in the preprocessing model with proof size $\delta(\lambda)$ and verification time $t(\lambda)$. However, this scheme provides no input privacy, since there is no encoded processing to be carried out on the input, which means that the input is sent as plaintext to the server.

2 | PRELIMINARIES

2.1 | Functional encryption

We provide the definition of functional encryption from the literature (e.g., [28–31]).

Definition 1 (Functional Encryption) *A functional encryption scheme* $\mathcal{FE}$ *over a message space* $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ *and a function space* $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ *consists of four algorithms* $(\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.KeyGen}, \mathsf{FE.Dec})$ *defined as follows:*

- $\mathsf{FE.Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$: on input the security parameter λ, the setup algorithm outputs a master public key mpk and a master secret key msk.
- $\mathsf{FE.KeyGen}(\mathsf{msk}, f) \to \mathsf{SK}_f$: on input the master secret key msk and a function $f \in \mathcal{F}_\lambda$, the key generation algorithm outputs a functional key SK_f.
- $\mathsf{FE.Enc}(\mathsf{mpk}, x) \to \mathsf{CT}$: on input the master public key mpk and a plaintext $x \in \mathcal{X}_\lambda$, the encryption algorithm outputs a ciphertext CT.
- $\mathsf{FE.Dec}(\mathsf{SK}_f, \mathsf{CT}) \to y$: on input the functional key SK_f corresponding to the function f and the ciphertext CT, the decryption algorithm outputs a value y.

Correctness. A functional encryption scheme $\mathcal{FE}$ is correct for a class of functions $\mathcal{F}$ if for any $f \in \mathcal{F}$, any pair of master keys $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$, any functional key $\mathsf{SK}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f)$, any $x \in \mathcal{X}$, and any $\mathsf{CT} \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, x)$, it holds that $\mathsf{FE.Dec}(\mathsf{SK}_f, \mathsf{CT}) = f(x)$ with all but a negligible probability over the internal randomness of the algorithms $\mathsf{FE.Setup}$, $\mathsf{FE.KeyGen}$, and $\mathsf{FE.Enc}$.***Adaptive* security.** A functional encryption scheme $\mathcal{FE}$ for a class of functions $\mathcal{F}$ is adaptively secure if for any probabilistic polynomial-time adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $negl(\lambda)$ such that

$$\mathsf{Adv}^{\mathsf{adp}}_{\mathcal{FE},\mathcal{A}}(\lambda) = \left| Pr[\mathsf{Exp}^{\mathsf{adp}}_{\mathcal{FE},\mathcal{A}}(\lambda) = 1] - 1/2 \right| \le negl(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$, where the random variable $\mathsf{Exp}^{\mathsf{adp}}_{\mathcal{FE},\mathcal{A}}(\lambda)$ is defined via the following experiment:

Experiment $\mathsf{Exp}^{\mathsf{adp}}_{\mathcal{FE},\mathcal{A}}(\lambda)$:
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$;
$(x_0^*, x_1^*, \mathsf{state}) \leftarrow \mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$, where $x_0^*, x_1^* \in \mathcal{X}$,
and for each $f \in \mathcal{F}$ which $\mathcal{A}_1$ queries to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$,
it holds that $f(x_0^*) = f(x_1^*)$;
$\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_b^*)$;
$b' \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{CT}^*, \mathsf{state})$, where for all $f \in \mathcal{F}$ that $\mathcal{A}_2$
queries to $\mathsf{KeyGen}\ (\mathsf{msk}, \cdot)$ it holds that $f(x_0^*) = f(x_1^*)$;
If $b' = b$, output 1, otherwise output 0.

2.2 | Homomorphic signature

In this subsection, we recall the syntax and security definition of homomorphic signatures as originally introduced by Boneh and Freeman [6]. We denote the message space by $\mathcal{M}$, and let $f : \mathcal{M}^k \to \mathcal{M}$ denote a function that takes k messages and outputs a result message in $\mathcal{M}$.

Definition 2 (Homomorphic Signature [6]**)** *A homomorphic signature scheme for the function family* $\mathcal{F}$ *is a tuple of probabilistic, polynomial-time algorithms* $\mathcal{HS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Evaluate})$ *as follows:*

- $\mathsf{Setup}(1^\lambda, k)$: Takes as inputs the security parameter λ and a maximum size k of a dataset whose messages can be signed. Outputs a public key pk and a secret key sk. The public key pk defines a message space $\mathcal{M}$, a signature space Σ, and a set $\mathcal{F}$ of functions $f : \mathcal{M}^k \to \mathcal{M}$.
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{tag}, m, i)$: Takes as inputs a secret key sk, a tag $\mathsf{tag} \in \{0,1\}^\lambda$, a message $m \in \mathcal{M}$ and its corresponding index $i \in [k]$, and outputs a signature $\sigma \in \Sigma$.
- $\mathsf{Evaluate}(\mathsf{pk}, \mathsf{tag}, f, \vec{\sigma})$: Takes as inputs a public key pk, a tag $\mathsf{tag} \in \{0,1\}^\lambda$, a function $f \in \mathcal{F}$, and a tuple of signatures $\vec{\sigma} \in \Sigma^k$, and outputs a signature $\sigma' \in \Sigma$.
- $\mathsf{Verify}(\mathsf{pk}, \mathsf{tag}, m, \sigma, f)$: Takes as inputs a public key pk, a tag $\mathsf{tag} \in \{0,1\}^\lambda$, a message $m \in \mathcal{M}$, a signature $\sigma \in \Sigma$, and a function $f \in \mathcal{F}$, and outputs either 0 (reject) or 1 (accept).

The tags are used to distinguish different datasets so that only signatures with matching tags can be employed to perform homomorphic evaluations. A tag is a bit-string of length λ that is randomly chosen from $\{0,1\}^\lambda$.

Definition 3 (Correctness [6]**)** *The* $\mathcal{F}$*-homomorphic signature* $\mathcal{HS}$ *is correct, if for any tag* $\mathsf{tag} \in \{0,1\}^\lambda$*, any function* $f \in \mathcal{F}$*, any tuple of messages* $\vec{m} = (m_1, \ldots, m_k) \in \mathcal{M}^k$*, and any index* $i \in [k]$*, we have*

$$\mathsf{Verify}(\mathsf{pk}, \mathsf{tag}, f(\vec{m}), \mathsf{Evaluate}(\mathsf{pk}, \mathsf{tag}, f, (\sigma_1, \ldots, \sigma_k)), f) = 1$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, k)$, $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{tag}, m, i)$ for $i \in [k]$. Note that the function f can also be a projection function P_i, that is, $P_i(m_1, \ldots, m_k) = m_i$, which means that the correctness also must hold for original signatures.

We use the notion of existential unforgeability under chosen dataset attacks in [6] to describe the unforgeable security definition of homomorphic signatures, which requires the adversary to query all the messages in a dataset at once.

Definition 4 (Unforgeability [6]**)** *An* $\mathcal{F}$*-homomorphic signature scheme* $\mathcal{HS} = (\mathsf{Setup}, \mathsf{Sign},$

$\mathsf{Verify}, \mathsf{Evaluate})$ *is unforgeable if for all k, no PPT adversary* $\mathcal{A}$ *can win the following defined experiment* $\mathsf{Expt}_{\mathcal{A}}^{UF}(1^\lambda)$ *with non-negligible probability.*

- The challenger runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, k)$ and sends pk to the adversary.
- Proceeding adaptively, $\mathcal{A}$ specifies a sequence of signature queries. Each query consists of:
  — a dataset given as a k-message vector $\overrightarrow{m}_i = \{m_{i,1}, \ldots, m_{i,k}\}$.
- For each i, the challenger sends back:
  — a randomly chosen dataset tag $\mathsf{tag}_i \in \{0,1\}^\lambda$;
  — a signature vector $\overrightarrow{\sigma}_i = \{\sigma_{ij}\}_{j\in[k]}$ where $\sigma_{ij} \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{tag}_i, m_{ij}, j)$ for $j \in [k]$
- $\mathcal{A}$ outputs a tag $\mathsf{tag}^* \in \{0,1\}^\lambda$, a message $m^* \in \mathcal{M}$, a function $f \in \mathcal{F}$, and a signature $\sigma^* \in \Sigma$.

The adversary wins if $\mathsf{Verify}(\mathsf{pk}, \mathsf{tag}^*, m^*, \sigma^*, f) = 1$ and either

1. (a type 1 forgery) $\mathsf{tag}^* \neq \mathsf{tag}_i$ for all i, or
2. (a type 2 forgery) $\mathsf{tag}^* = \mathsf{tag}_i$ for some i but $m^* \neq f(\overrightarrow{m}_i)$.

Context hiding, as a desirable privacy property for a homomorphic signature, requires that a signature that certifies y as the outcome of some computation f over original data should not reveal anything about the underlying data beyond the function value y. The full *context hiding* property of HS was first proposed by Ahn et al. [32] to capture a notion of privacy that the derived signature $\overrightarrow{\sigma}$ (i.e., signature produced homomorphically from the signatures $\sigma_1, \sigma_2, \ldots, \sigma_k$ corresponding to the data $m_1, m_2, \ldots, m_k$) is required to have the same distribution with the fresh signature σ_y generated by computing the signature of the message $y = f(m_1, m_2, \ldots, m_k)$, even if the original signatures $(\sigma_1, \sigma_2, \ldots, \sigma_k)$ are disclosed to the adversary. Then, Boneh et al. [6] defined a weak context-hiding property which captures the idea that the given signatures on a number of messages derived from two different datasets, the attacker cannot tell which dataset the derived signatures came from. They call it as a 'weak' context hiding since the original signatures on the dataset are not exposed to the adversary. Note that since the privacy property of homomorphic signature is not required in our article, here we omit its formal definition.

2.3 | Indistinguishability obfuscation

Here, we recall the notion of indistinguishability obfuscation which was originally proposed by Barak et al. [33]. The formal definition we present below is from [34].

Definition 5 (Indistinguishability obfuscation [34]**)** *A probabilistic polynomial time (PPT) algorithm* $i\mathcal{O}$ *is said to be an indistinguishability obfuscator for a circuits class* $\{\mathcal{C}_\lambda\}$, *if the following conditions are satisfied:*

- For all security parameters $\lambda \in \mathbb{N}$, for all $C \in C_\lambda$, for all inputs x, we have that

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1.$$

- For any (not necessarily uniform) PPT adversaries (Samp, D), there exists a negligible function $negl(\cdot)$ such that the following holds: if $\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda)] > 1 - negl(\lambda)$, then we have:

$$|\Pr[D(\sigma, i\mathcal{O}(\lambda, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda)] - \Pr[D(\sigma, i\mathcal{O}(\lambda, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow \mathsf{Samp}(1^\lambda)]| \leq negl(\lambda).$$

3 | HOMOMORPHIC SIGNCRYPTION WITH PUBLIC PLAINTEXT-RESULT CHECKABILITY IN a PUBLIC EVALUATION SETTING: DEFINITION AND BASIC CONSTRUCTION

Informally, a homomorphic signcryption scheme in a public evaluation setting consists of algorithms Setup, KGen_S, KGen_R, $\mathsf{Signcrypt}$, $\mathsf{UnSigncrypt}$ as well as two additional algorithms, that is, $\mathsf{Evaluate}$ and Verify. The $\mathsf{Evaluate}$ algorithm is able to transform the signcryptions on some original messages to a signcyrption on an outcome of the function applied to those original messages without using any secret keys. The Verify algorithm enables a verifier to test whether the (either original or derived) signcryption is a valid signcryption of a given message under the corresponding sender and receiver's public keys. If $\overrightarrow{c}$ is a valid set of signcryptions on the messages $\overrightarrow{m}$, then $\mathsf{Evaluate}(f, \overrightarrow{c})$ should be a valid signcryption for $f(\overrightarrow{m})$. An additional 'tag' is employed to distinguish different datasets, so that only signcryptions with matching tags can be computed homomorphically. A tag is a bit-string of length λ that is randomly chosen from $\{0,1\}^\lambda$.

Definition 6 (Homomorphic Signcryption in a public evaluation setting) *A homomorphic signcryption (HSC) scheme in a public evaluation setting is a tuple of probabilistic, polynomial-time algorithms* Setup, KGen_S, KGen_R, $\mathsf{Signcrypt}$, $\mathsf{Unsigncrypt}$, $\mathsf{Evaluate}$, Verify *as follows:*

- $\mathsf{Setup}(1^\lambda, k)$: It takes as inputs the security parameter λ and a maximum size k of a dataset, whose messages can be signcrypted. It outputs the public parameter pp and defines a message space $\mathcal{M}$, a signcryption space $\mathcal{C}$, and a set $\mathcal{F}$ of functions $f : \mathcal{M}^k \rightarrow \mathcal{M}$.
- $\mathsf{KGen}_S(\mathsf{pp})$: It takes as inputs the public parameter pp, and outputs a sender's key-pair $(\mathsf{pk}_S, \mathsf{sk}_S)$.

- $\mathsf{KGen}_R(\mathsf{pp})$: It takes as inputs the public parameter pp, and outputs a receiver's key-pair $(\mathsf{pk}_R, \mathsf{sk}_R)$.
- $\mathsf{Signcrypt}(\mathsf{pp}, \mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}, m, i)$: It takes as inputs the public parameter pp, sender's private key sk_S, receiver's public key pk_R, a tag $\mathsf{tag} \in \{0,1\}^\lambda$, a message $m \in \mathcal{M}$ and its corresponding index $i \in [k]$, and outputs a signcryption $c \in \mathcal{C}$.
- $\mathsf{UnSigncrypt}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_R, c)$: It takes as inputs the public parameter pp, the sender's public key pk_S, the receiver's secret key sk_R, and a signcryption $c \in \mathcal{C}$, and outputs a message $m \in \mathcal{M}$.
- $\mathsf{Evaluate}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, f, \overrightarrow{c})$: It takes as inputs the public parameter pp, the sender's public key pk_S, the receiver's public key pk_R, a tag $\mathsf{tag} \in \{0,1\}^\lambda$, a function $f \in \mathcal{F}$, and a tuple of signcryptions $\overrightarrow{c} \in \mathcal{C}^k$, and outputs a derived signcryption $c' \in \mathcal{C}$.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, m', c', f)$: It takes as inputs the public parameter pp, sender's public key pk_S, receiver's public key pk_R, a tag $\mathsf{tag} \in \{0,1\}^\lambda$, a message $m' \in \mathcal{M}$, a function $f \in \mathcal{F}$, and a signcryption $c' \in \mathcal{C}$, and outputs either 0 (reject) or 1 (accept).

Let $\{\Phi_i : \mathcal{M}^k \to \mathcal{M}\}$ be the function $\Phi_i(m_1, \ldots, m_k) = m_i$ that maps onto the i-th component and $\Phi_1, \ldots, \Phi_k \in \mathcal{F}$ for all pp output by $\mathsf{Setup}(1^\lambda, k)$.

Correctness For all $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, k)$, $(\mathsf{pk}_S, \mathsf{sk}_S) \leftarrow \mathsf{KGen}_S(\mathsf{pp})$ and $(\mathsf{pk}_R, \mathsf{sk}_R) \leftarrow \mathsf{KGen}_R(\mathsf{pp})$ we have:

1. For all tags $\mathsf{tag} \in \{0,1\}^\lambda$, $m \in \mathcal{M}$, and $i \in \{1, \ldots, k\}$, if $c \leftarrow \mathsf{Signcrypt}(\mathsf{pp}, \mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}, m, i)$, then with overwhelming probability it holds that $\mathsf{UnSigncrypt}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_R, c) = m$ and $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, m, c, \Phi_i) = 1$.
2. For all $\mathsf{tag} \in \{0,1\}^\lambda$, tuples $\overrightarrow{m} = (m_1, \ldots, m_k) \in \mathcal{M}^k$, and functions $f \in \mathcal{F}$, if $c_i \leftarrow \mathsf{Signcrypt}(\mathsf{pp}, \mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}, m_i, i)$ for $i = 1, \ldots, k$ and $c' \leftarrow \mathsf{Evaluate}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, f, (c_1, \ldots, c_k))$, then with overwhelming probability it holds $\mathsf{UnSigncrypt}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_R, c') = f(\overrightarrow{m})$ and $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, f(\overrightarrow{m}), c', f) = 1$.

We say that a signcryption scheme as above is $\mathcal{F}$-homomorphic, or homomorphic with respect to $\mathcal{F}$. For simplicity, we assume in our homomorphic signcryption systems all data sets are composed of exact k items.

3.1 | Unforgeability

The security model for homomorphic signcryptions in a public evaluation setting allows an adversary to get access to the signcryption oracle by submitting datasets of his choice which contains up to k messages and obtaining the signcryptions as responses along with a randomly selected tag tag for each dataset that is queried. Meanwhile, the adversary is allowed to have access to the unsigncryption oracle by issuing queries on the signcryptions of his choice and receiving the original message m back. The adversary wins the game if he outputs a message-signcryption pair (m^*, c^*) as well as a tag tag^* and a function f^* as a forgery. To win the game, the forgery should be one of the following two distinct types of forgeries. In a *type 1 forgery*, the signcryption c^* is a *valid* signcryption of a message m^* associated with the tag tag^*, which has *not* been chosen as a tag for the dataset queried to the signcryption oracle earlier. This corresponds to the usual notion of a signcryption forgery. In a *type 2 forgery*, the pair (m^*, c^*) verifies for some dataset that *has* been queried to the signcryption oracle, but for which m^* is not equal to the outcome of f^* applied to the queried messages.

Definition 7 (Unforgeability) *An $\mathcal{F}$-homomorphic signcryption scheme in a public evaluation setting* $\mathsf{HSC} = (\mathsf{Setup}, \mathsf{KGen}_S, \mathsf{KGen}_R, \mathsf{Signcrypt}, \mathsf{Unsigncrypt}, \mathsf{Evaluate}, \mathsf{Verify})$ *is unforgeable if for all k no PPT adversary $\mathcal{A}$ can win the following defined experiment* $\mathsf{Expt}^{\mathsf{UF}}_{\mathsf{HSC},\mathcal{A}}(1^\lambda)$ *with non-negligible probability.*

- The challenger runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, k)$, $(\mathsf{pk}_S, \mathsf{sk}_S) \leftarrow \mathsf{KGen}_S(\mathsf{pp})$ and $(\mathsf{pk}_R, \mathsf{sk}_R) \leftarrow \mathsf{KGen}_R(\mathsf{pp})$, and sends $\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R$ to the adversary $\mathcal{A}$.
- $\mathcal{A}$ proceeds with adaptive queries,
  - SIGNCRYPTION QUERIES. Each query consists of:
- a dataset given as a k-message vector $\overrightarrow{m}_i = \{m_{i,1}, \ldots, m_{i,k}\}$.
- For each i, the challenger sends back a randomly chosen tag $\mathsf{tag}_i \in \{0,1\}^\lambda$, and a signcryption vector $\overrightarrow{c}_i = \{c_{ij}\}_{j \in [k]}$ where $c_{ij} \leftarrow \mathsf{Signcrypt}(\mathsf{pp}, \mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}_i, m_{ij})$ for $j \in [k]$.
- a dataset given as a k-message vector $\overrightarrow{m}_i = \{m_{i,1}, \ldots, m_{i,k}\}$.
- For each i, the challenger sends back a randomly chosen tag $\mathsf{tag}_i \in \{0,1\}^\lambda$, and a signcryption vector $\overrightarrow{c}_i = \{c_{ij}\}_{j \in [k]}$ where $c_{ij} \leftarrow \mathsf{Signcrypt}(\mathsf{pp}, \mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}_i, m_{ij})$ for $j \in [k]$.
  - UNSIGNCRYPTION QUERIES. Each query consists of:
- a signcryption $c_\imath \in \mathcal{C}$
- For each $\imath$, the challenger sends back: a message $m_\imath \leftarrow \mathsf{UnSigncrypt}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_R, c_\imath)$.
- $\mathcal{A}$ outputs a tag $\mathsf{tag}^* \in \{0,1\}^\lambda$, a message $m^* \in \mathcal{M}$, a function $f^* \in \mathcal{F}$, and a signcryption c^*.

The adversary wins if $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}^*, m^*, c^*, f^*) = 1$ and $\mathsf{UnSigncrypt}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{sk}_R, c^*) = m^*$ and either

1. (a type 1 forgery) $\mathsf{tag}^* \neq \mathsf{tag}_i$ for all i or
2. (a type 2 forgery) $\mathsf{tag}^* = \mathsf{tag}_i$ for some i but $m^* \neq f^*(\overrightarrow{m}_i)$.

Remark 1 *Our security model requires that all k messages in a dataset is signcrypted at once.*

Remark 2 *While the adversary can always compute the message m^* by querying the unsigncryption oracle with c^*, if the adversary is able to forge a valid signcryption c^*, he always can obtain the message m^* by querying the unsigncryption oracle. Thus, we only require that for the type 1 forgery* $(\mathsf{tag}^*, m^*, c^*, f^*)$ *the* *corresponding tag* tag^* *of message* m^* *has never been chosen as a tag for the dataset queried to the signcryption oracle, while a signcrypted message* c^* *might has been queried to the unsigncryption oracle.*

3.2 | Weak message privacy

Our definition of message privacy for homomorphic signcryption is defined in a game. Given the access to the signcryption oracle as well as the unsigncryption oracle, the adversary has to come up with two equal-length messages $\overrightarrow{m}_0^*$ and $\overrightarrow{m}_1^*$ along with a sequence of functions $f_1, \ldots, f_s$. One of $(\overrightarrow{m}_0^*, \overrightarrow{m}_1^*)$ will be signcrypted at random, and the signcryption challenge $\widehat{c}_i^*$ will be generated by the homomorphic evaluation algorithm for the function f_i from the original signcryption c_b^* on message $\overrightarrow{m}_b^*$. Only the derived signcryption $\widehat{c}_i^*$ for all $i \in [s]$ will be given to the adversary and then, the adversary has to guess which message was signcrypted. We call it *weak* message privacy since we require that the original signcryptions on the dataset are not disclosed to the adversary.

Definition 8 (Weak Message Privacy) *An* $\mathcal{F}$*-homomorphic signcryption scheme* HSC *in a public evaluation setting is weakly message private if for all k no PPT adversary* $\mathcal{A}$ *can win the following defined experiment* $\mathsf{Expt}^{\mathsf{wMP}}_{\mathsf{HSC},\mathcal{A}}(1^\lambda)$ *with non-negligible advantage.*

- The challenger runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, k)$, $(\mathsf{pk}_S, \mathsf{sk}_S) \leftarrow \mathsf{KGen}_S(\mathsf{pp})$ and $(\mathsf{pk}_R, \mathsf{sk}_R) \leftarrow \mathsf{KGen}_R(\mathsf{pp})$, and sends $\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R$ together with sk_S to $\mathcal{A}$.
- $\mathcal{A}$ adaptively performs UnSigncryption queries as in the experiment $\mathsf{Expt}^{\mathsf{UF}}_{\mathsf{HSC},\mathcal{A}}$.
- $\mathcal{A}$ outputs $(\overrightarrow{m}_0^*, \overrightarrow{m}_1^*, f_1, \ldots, f_s)$ with $\overrightarrow{m}_0^*, \overrightarrow{m}_1^* \in \mathcal{M}^k$. The functions $f_1, \ldots, f_s$ are in $\mathcal{F}$ and satisfy $f_i(\overrightarrow{m}_0^*) = f_i(\overrightarrow{m}_1^*)$ for all $i \in [s]$.
- The challenger generates a random bit $b \in \{0, 1\}$ and a random tag $\mathsf{tag} \in \{0,1\}^\lambda$. It signcrypts the messages in $\overrightarrow{m}_b^*$ using a tag to obtain a vector $\overrightarrow{c}$ of k signcryptions, where $c_j \leftarrow \mathsf{Signcryption}(\mathsf{pp}, \mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}, m_{bj}^*, j)$ for all $j \in [k]$. Next, for each $i \in [s]$ the challenger computes a derived signcryption $\widehat{c}_i \leftarrow \mathsf{Evaluate}(\mathsf{pp}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, f_i, \overrightarrow{c})$. It sends tag and the signcryptions $(\widehat{c}_1, \ldots, \widehat{c}_s)$ to $\mathcal{A}$. Note that the functions $f_1, \ldots, f_s$ can be output adaptively after $\overrightarrow{m}_0^*, \overrightarrow{m}_1^*$ are output.
- $\mathcal{A}$ adaptively performs unsigncryption queries as before. Note that there is no need for $\mathcal{A}$ to ask its unsigncryption oracle for any query c which is the same as any one of $\widehat{c}_i$ (for $i \in [s]$) returned as the challenged signcryptions, since it already knows the answer $f_i(\overrightarrow{m}_b^*)$ which is the same for $b \in \{0, 1\}$ $(f_i(\overrightarrow{m}_0^*) = f_i(\overrightarrow{m}_1^*))$.
- $\mathcal{A}$ outputs a bit b'.

The adversary $\mathcal{A}$ wins the game if $b = b'$.

We also consider a selective variant of message privacy. The selectively weak message privacy game is equivalent to the above one with the exception that the attacker must declare the challenge messages $\overrightarrow{m}_0^*, \overrightarrow{m}_1^*$ at the very beginning before it sees the public parameters.

Definition 9 (Selectively Weak Message Privacy) *An* $\mathcal{F}$*-homomorphic signcryption scheme* HSC *is selectively weak message private for all PPT adversaries* $\mathcal{A}$*, if the advantage of* $\mathcal{A}$ *is negligible in the selective weak message privacy game.*

3.3 | Construction of an HSC in a public evaluation setting

In this section, we present an HSC scheme in a public evaluation setting from the HS scheme *without* the context-hiding property based on indistinguishability obfuscation. Our construction relies on the following building blocks:

- An $\mathcal{F}$-homomorphic signature scheme $\mathcal{HS} = (\mathsf{HS.Setup}, \mathsf{HS.Sign}, \mathsf{HS.Evaluate}, \mathsf{HS.Verify})$ with message of length $|m|$ and signature of length ℓ_{sig}.
- An IND-CPA secure public key encryption scheme $\mathcal{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.KeyGen}, \mathsf{PKE.Dec})$ for messages of length $(\ell_{\mathsf{sig}} + |m|)$, the ciphertexts of length ℓ_{PKE} and the randomness of length ℓ_{PKEr}.
- Indistinguishability obfuscation $i\mathcal{O}$.
- Statistically simulation sound non-interactive zero knowledge proofs $\mathsf{SSS\text{-}NIZKs} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ for the following NP language:

$$L = \{(\mathsf{pk}_1, \mathsf{pk}_2, e_1, e_2) | \ \exists\, \sigma \| m, r_1, r_2 \text{ such that } e_1 = \mathsf{PKE.Enc}(\mathsf{pk}_1, \sigma \| m; r_1) \wedge e_2 = \mathsf{PKE.Enc}(\mathsf{pk}_2, \sigma \| m; r_2)\}, \tag{1}$$

and has a simulator Sim.

Our HSC scheme. $\mathsf{HSC} = \mathsf{Setup}, \mathsf{KGen}_S, \mathsf{KGen}_R, \mathsf{Signcrypt}, \mathsf{Unsigncrypt}, \mathsf{Evaluate}, \mathsf{Verify}$ in a public evaluation setting with respect to $\mathcal{F}$ is built as follows.

- **HSC.Setup**$(1^\lambda, k) \rightarrow \mathsf{pp}$:
  - Run $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ and set obtain $\mathsf{pp} := \mathsf{crs}$.
- **HSC.KGen**$_S(\mathsf{pp}, k) \rightarrow (\mathsf{pk}_S, \mathsf{sk}_S)$:
  - Generate a key-pair of $\mathcal{HS}$, that is, $(\mathsf{hsk}, \mathsf{hpk}) \leftarrow \mathsf{HS.Setup}(1^\lambda)$. Set $\mathsf{sk}_S := \mathsf{hsk}$ and $\mathsf{pk}_S := \mathsf{hpk}$.
- **HSC.KGen**$_R(\mathsf{pp}, k) \rightarrow (\mathsf{pk}_R, \mathsf{sk}_R)$:
  - Generate two independent key-pairs of $\mathcal{PKE}$, that is, $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ and $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$.
  - Create two obfuscations $i\mathcal{O}(\mathsf{Prog}_\mathsf{V})$ and $i\mathcal{O}(\mathsf{Prog}_\mathsf{E})$ for the programme Prog_V defined in Figure 2a and programme Prog_E defined in Figure 2b;

- Set $\mathsf{pk}_R := (\mathsf{pk}_1, \mathsf{pk}_2, i\mathcal{O}(\mathsf{Prog_V}), i\mathcal{O}(\mathsf{Prog_E}))$ and $\mathsf{sk}_R := \mathsf{sk}_1$.

- **HSC.Signcrypt**$(\mathsf{pp}, \mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}, m, i) \to c_i$:
  - Parse $\mathsf{sk}_S = \mathsf{hsk}$ and $\mathsf{pk}_R = (\mathsf{pk}_1, \mathsf{pk}_2, i\mathcal{O}(\mathsf{Prog_V}), i\mathcal{O}(\mathsf{Prog_E}))$;
  - Use the tag $\mathsf{tag} \in \{0,1\}^\lambda$ to generate a signature for the message m as $\sigma_i \leftarrow \mathsf{HS.Sign}(\mathsf{hsk}, \mathsf{tag}, m, i)$ where $i \in [k]$ is the index of message m in the dataset.
  - Compute ciphertexts for the same plaintext $\sigma_i \| m_i$ under two different public keys pk_1 and pk_2, respectively, namely, $e_{i1} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_1, \sigma_i \| m_i; r_{i1})$ and $e_{i2} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_2, \sigma_i \| m_i; r_{i2})$ where $r_{i1}, r_{i2} \in \{0,1\}^{\ell_{\mathsf{PKEr}}}$. Then generate the proof $\pi_i \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, \nu_i, \omega_i)$ where $\nu_i = (\mathsf{pk}_1, \mathsf{pk}_2, e_{i1}, e_{i2})$ is a statement of the NP language defined in (1) and $\omega_i = (\sigma_i \| m_i, r_{i1}, r_{i2})$ is the corresponding witness.
  - Output $c_i = (e_{i1}, e_{i2}, \pi_i)$.

- **HSC.UnSigncrypt**$(\mathsf{pp}, \mathsf{sk}_R, \mathsf{pk}_S, \mathsf{tag}, c_i) \to m_i$:
  - Parse $\mathsf{pk}_S = \mathsf{hpk}$, $\mathsf{sk}_R = \mathsf{sk}_1$ and $c_i = (e_{i1}, e_{i2}, \pi_i)$;
  - Check that π_i is a valid NIZK proof using the NIZK.Verify algorithm and crs for the NP language (1). If the check fails output $\perp$; Otherwise compute $\sigma_i \| m_i = \mathsf{PKE.Dec}\ (\mathsf{sk}_1, e_{i1})$.
  - Check that $\mathsf{HS.Verify}(\mathsf{hpk}, \mathsf{tag}, m_i, \sigma_i) = 1$. If it is true, output m_i; else, output $\perp$.

- **HSC.Evaluate**$(\mathsf{pp}, \mathsf{tag}, \mathsf{pk}_R, \mathsf{pk}_S, f, \overrightarrow{c}) \to \widehat{c}$:
  - Parse $\mathsf{pp} = \mathsf{crs}$, $\mathsf{pk}_S = \mathsf{hpk}$, and $\mathsf{pk}_R = (\mathsf{pk}_1, \mathsf{pk}_2, i\mathcal{O}(\mathsf{Prog_V}), i\mathcal{O}(\mathsf{Prog_E}))$;
  - Run the obfuscated programme $i\mathcal{O}(\mathsf{Prog_E})$ on input tag, hpk, f and $\overrightarrow{c} = (c_1, \ldots, c_k)$ where $c_i = (e_{i1}, e_{i2}, \pi_i)$ for all $i \in [k]$, and obtain the output $\widehat{\sigma} \| \widehat{m}$.
  - Generate $\widehat{e}_1 \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_1, \widehat{\sigma} \| \widehat{m}; r'_1)$ and $\widehat{e}_2 \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_2, \widehat{\sigma} \| \widehat{m}; r'_2)$, and the NIZK proof $\widehat{\pi} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, \widehat{\nu}, \widehat{\omega})$ where $\widehat{\nu} = (\mathsf{pk}_1, \mathsf{pk}_2, \widehat{e}_1, \widehat{e}_2)$ is a statement of the NP language (1) and $\widehat{\omega} = (\widehat{\sigma} \| \widehat{m}, r'_1, r'_2)$ is the corresponding witness.
  - Output $\widehat{c} = (\widehat{e}_1, \widehat{e}_2, \widehat{\pi})$.

- **HSC.Verify**$(\mathsf{pp}, \mathsf{tag}, \mathsf{pk}_R, \mathsf{pk}_S, \widehat{m}, \widehat{c}, f)$:
  - Parse $\mathsf{pp} = \mathsf{crs}$, $\mathsf{pk}_R = (\mathsf{pk}_1, \mathsf{pk}_2, i\mathcal{O}(\mathsf{Prog_V}), i\mathcal{O}(\mathsf{Prog_E}))$, $\mathsf{pk}_S = \mathsf{hpk}$ and $\widehat{c} = (\widehat{e}_1, \widehat{e}_2, \widehat{\pi})$;
  - Run the obfuscated programme $i\mathcal{O}(\mathsf{Prog_V})$ on input $(\widehat{c} = (\widehat{e}_1, \widehat{e}_2, \widehat{\pi}), \widehat{m}, \mathsf{tag}, \mathsf{hpk}, f)$ and obtain the output b.
  - Output the returned bit b.

Correctness. Correctness of our HSC scheme follows immediately from the correctness of the $i\mathcal{O}$, PKE system, SSS-NIZK, HS scheme and the description of the programme template $\mathsf{Prog_V}$ and $\mathsf{Prog_E}$.

Theorem 1 *Assuming the underlying homomorphic signature scheme $\mathcal{HS}$ is existentially unforgeable against chosen message attacks as defined in Definition*

(a)

$\mathsf{Prog_V}$

Hardwired into the circuit: sk_1, crs.
Input to the circuit: $c = (e_1, e_2, \pi)$, m, tag, hpk, f.
Algorithm:

1. Check that π is a valid NIZK proof using the NIZK.Verify algorithm and crs for the statement (1).
2. If the check fails output $\perp$; otherwise compute $\sigma' \| m' = \mathsf{PKE.Dec}(\mathsf{sk}_1, e_1)$ and run $b \leftarrow \mathsf{HS.Verify}(\mathsf{hpk}, \mathsf{tag}, m, \sigma', f) = 1$;
3. Output b.

The program $\mathsf{Prog_V}$

(b)

$\mathsf{Prog_E}$

Hardwired into the circuit: sk_1, crs.
Input to the circuit: $\vec{c} = (c_1, \ldots, c_k)$, tag, hpk, f.
Algorithm:

1. For all $i \in [k]$ parse $c_i = (e_{i1}, e_{i2}, \pi_i)$ and check that π_i is a valid NIZK proof using the NIZK.Verify algorithm and the crs for the NP-statement (1);
2. If the check fails output $\perp$; otherwise for all $i \in [k]$ compute $\sigma_i \| m_i = \mathsf{PKE.\ Dec}(\mathsf{sk}_1, e_{i1})$;
3. Check that $\mathsf{HS.Verify}(\mathsf{hpk}, \mathsf{tag}, m_i, \sigma_i) = 1$. If it fails, output $\perp$; otherwise, compute $\hat{m} = f(m_1, \ldots, m_k)$ and $\hat{\sigma} \leftarrow \mathsf{HS.Evaluate}(\mathsf{hpk}, \mathsf{tag}, f, (\sigma_1, \ldots, \sigma_k))$;
4. Output $\hat{\sigma} \| \hat{m}$.

The program $\mathsf{Prog_E}$

FIGURE 2 The description of the programs $\mathsf{Prog_V}$ and $\mathsf{Prog_E}$

4, *the homomorphic signcryption scheme described above satisfies unforgeability against chosen message attacks as defined in Definition* 7.

Proof: Let us fix a PPT adversary $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HSC}}$ attacking the unforgeability security of the HSC scheme built above. We will use $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HSC}}$ to construct an adversary $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ such that, if $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HSC}}$ wins in the unforgeability game for our HSC scheme given above with non-negligible probability, then $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ breaks the underlying homomorphic signature scheme $\mathcal{HS}$, which is assumed to be existentially unforgeable against chosen message attacks.

We now describe the constructed homomorphic signature adversary, $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$. In the security game for the HS scheme, $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ is given the verification key hpk, and access to a signing oracle $\mathsf{O}_{\mathsf{HS}_{\mathsf{sig}}}$. He is considered to be successful in producing a forgery if he outputs a valid signature for a message that was not queried from $\mathsf{O}_{\mathsf{HS}_{\mathsf{sig}}}$ (type 1 forgery) or that was queried to the signing oracle, but for which m^* does not equal f^* applied to the messages queried (type 2 forgery).

$\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ interacts with $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HSC}}$, playing the role of the challenger in the unforgeability game for our HSC scheme built above. This means that $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ must simulate the signcryption oracle and the unsigncryption oracle. After receiving the challenge verification key hpk of the HS scheme, $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ first generates $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$, key-pairs $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, and creates the obfuscated programme $i\mathcal{O}(\mathsf{Prog}_\mathsf{V})$ and $i\mathcal{O}(\mathsf{Prog}_\mathsf{E})$ for the programme Prog_V depicted in Figure 2a and Prog_E in Figure 2b, respectively. $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ sets $\mathsf{pp} := \mathsf{crs}$, $\mathsf{pk}_S := \mathsf{hpk}$ and $\mathsf{pk}_R := (\mathsf{pk}_1, \mathsf{pk}_2, , i\mathcal{O}(\mathsf{Prog}_\mathsf{V}), i\mathcal{O}(\mathsf{Prog}_\mathsf{E}))$.

To answer the i-th query to the signcryption oracle, that is, a k-message vector $\overrightarrow{m}_i = \{m_{i1}, \ldots, m_{ik}\}$ issued by $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HSC}}$, $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ performs the following:

- It sends the k-message vector $\overrightarrow{m}_i = \{m_{i1}, \ldots, m_{ik}\}$ to its own signing oracle $\mathsf{O}_{\mathsf{HS}_{\mathsf{sig}}}$ to get a tag tag_i and a signature vector $\overrightarrow{\sigma}_i = (\sigma_{i1}, \ldots, \sigma_{ik})$ of k signatures.
- For each $j \in [k]$, it generates $e^1_{ij} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_1, \sigma_{ij} \| m_{ij}; r^1_{ij})$ and $e^2_{ij} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_2, \sigma_{ij} \| m_{ij}; r^2_{ij})$, and the NIZK proof π_{ij} for the statement (1).
- It sets $\left(\mathsf{tag}_i, \overrightarrow{c}_i = \{c_{i1}, \ldots, c_{ik}\}\right)$ where $c_{ij} = \left(e^1_{ij},\ e^2_{ij},\ \pi_{ij}\right)$ for each $j \in [k]$ and returns it back to $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HSC}}$.

To answer the query, the signcryption $c_\iota = (e^1_\iota, e^2_\iota, \pi_\iota)$ with corresponding tag tag_ι issued by $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HSC}}$, to the unsigncryption oracle, $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ performs as following:

- Checks that π_ι is a valid NIZK proof using the $\mathsf{NIZK.Verify}$ algorithm and the crs for the NP-statement (1). If the check fails, it outputs $\perp$; Otherwise, it computes $\sigma_i \| m_i = \mathsf{PKE.Dec}(\mathsf{sk}_1, e^1_i)$.
- Checks that $\mathsf{HS.Verify}(\mathsf{hpk}, \mathsf{tag}_i, m_i, \sigma_i) = 1$. If it is true, it returns m_i; else, it returns $\perp$.

Eventually, $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HSC}}$ outputs a tuple $(\mathsf{tag}^*, m^*, c^*, f^*)$ where $c^* = (e^*_1, e^*_2, \pi^*)$. $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ computes $\sigma^* \| m^* = \mathsf{PKE.Dec}(\mathsf{sk}_1, e^*_1)$ to obtain the output $\sigma^* \| m^*$, and then outputs $(\mathsf{tag}^*, m^*, \sigma^*, f^*)$ as its message-forgery pair in the unforgeability game for the underlying $\mathcal{HS}$ scheme.

It is easy to see that $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ exactly recreates the environment. Thus, if $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HSC}}$ produces a forgery in our HSC scheme with non-negligible probability $1/Poly(\lambda)$, then $\mathcal{A}^{\mathsf{Unf}}_{\mathsf{HS}}$ successfully forges in the underlying $\mathcal{HS}$ scheme with non-negligible probability $1/Poly(\lambda)$. But, this cannot be the case, since we have assumed that the $\mathcal{HS}$ scheme is existentially unforgeable against chosen-message attacks. We conclude that our HSC scheme as specified above satisfies the unforgeability security of Definition 7. □

Theorem 2 *Assuming iO is a secure indistinguishability obfuscator, a IND-CPA secure public key encryption $\mathcal{PKE}$, along with the statistical simulation-soundness and the zero-knowledge properties of a SSS-NIZK system, the homomorphic signcryption scheme described above is selectively weakly message private as defined in Definition* 9 *for datasets up to k.*

Proof: We now show that our HSC scheme satisfies the selective weak message privacy of Definition 9. We prove that no poly-time attacker can break the weak message privacy of our HSC scheme if our underlying assumptions hold. We organise our proof into a sequence of hybrids. In the first hybrid, the challenger signcrypts the messages in $\overrightarrow{m}^*_0$ and computes the signcryption on $f_i(\overrightarrow{m}^*_0)$ for each $i \in [s]$. We then gradually change the signcryptions in multiple hybrid steps into the signcryptions on $\overrightarrow{m}^*_1$ and $f_i(\overrightarrow{m}^*_1)$. We show that each successive hybrid experiment is indistinguishable from the former one.

Hyb_0: In this hybrid the following game is played.

- The adversary $\mathcal{A}$ outputs the challenged message $(\overrightarrow{m}^*_0, \overrightarrow{m}^*_1)$.
- The challenger generates $\mathsf{crs} \leftarrow \mathsf{NIZK.\ Setup}(1^\lambda)$, $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ and $(\mathsf{hsk}, \mathsf{hpk}) \leftarrow \mathsf{HS.Setup}(1^\lambda)$. Then, it creates the obfuscations $i\mathcal{O}(\mathsf{Prog}_V)$ and $i\mathcal{O}(\mathsf{Prog}_E)$ for the programme Prog_V of Figure 2a and Prog_E of Figure 2b. The challenger gives $\mathsf{pp} := \mathsf{crs}$, $\mathsf{pk}_S := \mathsf{hpk}$, $\mathsf{pk}_R := (\mathsf{pk}_1, \mathsf{pk}_2, i\mathcal{O}(\mathsf{Prog}_\mathsf{V}), i\mathcal{O}(\mathsf{Prog}_\mathsf{E}))$ together with $\mathsf{sk}_S := \mathsf{hsk}$ to $\mathcal{A}$.
- For the unsigncryption queries $c_i = (e^1_i, e^2_i, \pi_i)$ with corresponding tag tag_i issued by $\mathcal{A}$, the challenger checks that π_i is a valid NIZK proof using the $\mathsf{NIZK.Verify}$ algorithm and the crs for the NP-statement (1). If the check fails, it outputs $\perp$; Otherwise, it computes $\sigma_i \| m_i = \mathsf{PKE.Dec}(\mathsf{sk}_1, e^1_i)$ and verifies that $\mathsf{HS.Verify}(\mathsf{hpk}, \mathsf{tag}_i, m_i, \sigma_i) = 1$. If it is true, it returns m_i; else, it returns $\perp$.
- The adversary $\mathcal{A}$ outputs $(f_1,\ \ldots,\ f_s)$ satisfying $f_i(\overrightarrow{m}^*_0) = f_i(\overrightarrow{m}^*_1)$ for all $i \in [s]$.
- The challenger randomly samples a tag tag^* and generates $\sigma^0_j \leftarrow \mathsf{HS.Sign}\left(\mathsf{hsk}, \mathsf{tag}^*, m^*_{0j},\ j\right)$ for alright $j \in [k]$. Then, for each $j \in [k]$ it generates $e^1_j \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_1, \sigma^0_j \| m^*_{0j}; r^1_j)$ and $e^2_j \leftarrow \mathsf{PKE.Enc}\ (\mathsf{pk}_2, \sigma^0_j \| m^*_{0j}; r^2_j)$, and the NIZK proof π_j for statement (1), and sets $c^0_j = (e^1_j, e^2_j, \pi_j)$. For all $i \in [s]$

the challenger runs $(\widehat{\sigma}_i \| \widehat{m}_i) \leftarrow i\mathcal{O}(\mathsf{Prog_E})\ ((c_1^0, \dots, c_k^0), \mathsf{tag}^*, \mathsf{hpk}, f_i)$ and generates $\widehat{e}_i^1 \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_1, \widehat{\sigma}_i \| \widehat{m}_i; r'_{i1})$ and $\widehat{e}_i^2 \leftarrow \mathsf{PKE.Enc}\ (\mathsf{pk}_2, \widehat{\sigma}_i \| \widehat{m}_i; r'_{i2})$, and the NIZK proof $\widehat{\pi}_i$ for statement (1). It sets $\widehat{c}_i = (\widehat{e}_i^1, \widehat{e}_i^2, \widehat{\pi}_i)$ and sends $(\mathsf{tag}^*, \widehat{c}_1, \dots, \widehat{c}_s)$ to $\mathcal{A}$.

- The adversary $\mathcal{A}$ outputs a bit b' and wins if $b = b'$.

Hyb_1: This hybrid is identical to Hyb_0 with the exception that (crs, π) is simulated as

$$(\mathsf{crs}, \pi) \leftarrow \mathsf{Sim}\big(\exists (u, r^1, r^2),\ \text{s.t.}\quad \begin{aligned} e^1 &= \mathsf{PKE.Enc}(\mathsf{pk}_1, u; r^1) \wedge \\ e^2 &= \mathsf{PKE.Enc}(\mathsf{pk}_2, u; r^2)\big). \end{aligned} \tag{2}$$

Since the SSS-NIZK system is computationally zero knowledge, Hyb_1 is indistinguishable from Hyb_0.

Hyb_2: This hybrid is identical to Hyb_1, with the exception that the ciphertexts are generated as following. For all $j \in [k+s]$, $e_j^1 = \mathsf{PKE.Enc}(\mathsf{pk}_1, u_j^0; r_j^1)$, where for $j \in [k]$, $u_j^0 = \sigma_j^0 \| m_{0j}^*$, $\sigma_j^0 \leftarrow \mathsf{HS.Sign}\left(\mathsf{hsk}, \mathsf{tag}^*, m_{0j}^*, j\right)$ while for $j \in [k+1, k+s]$, $u_j^0 = \sigma_j^0 \| f_{j-k}(\overrightarrow{m}_0^*)$, $\sigma_j^0 \leftarrow \mathsf{HS.Evaluate}(\mathsf{hpk}, \mathsf{tag}^*, f_{j-k}, (\sigma_1^0, \dots, \sigma_k^0))$. For all $j \in [k+s]$, $e_j^2 = \mathsf{PKE.Enc}(\mathsf{pk}_2, u_j^1; r_j^2)$, where for $j \in [k]$, $u_j^1 := \sigma_j^1 \| m_{1j}^*$, $\sigma_j^1 \leftarrow \mathsf{HS.Sign}(\mathsf{hsk}, \mathsf{tag}^*, m_{1j}^*, j)$ while for $j \in [k+1, k+s]$, $u_j^1 = \sigma_j^1 \| f_{j-k}(\overrightarrow{m}_1^*)$, $\sigma_j^1 \leftarrow \mathsf{HS.Evaluate}(\mathsf{hpk}, \mathsf{tag}^*, f_{j-k}, \sigma_1^1, \dots, \sigma_k^1\big)$. (The NIZK is still simulated.) Since the PKE system is IND-CPA secure, Hyb_2 is indistinguishable from Hyb_1.

Hyb_3: This hybrid is identical to Hyb_2, with the exception that the challenger generates the obfuscation of programme $\mathsf{Prog}_\mathsf{V}^1$ which is defined in Figure 3a instead of an obfuscation for programme $\mathsf{Prog_V}$ defined in Figure 2a as well as the obfuscation of programme $\mathsf{Prog}_\mathsf{E}^1$ which is defined in Figure 3b instead of an obfuscation for programme $\mathsf{Prog_E}$ defined in Figure 2b. Since $i\mathcal{O}$ is an indistinguishability obfuscator, Hyb_3 is indistinguishable from Hyb_2.

Hyb_4: This hybrid is identical to hybrid Hyb_3 with the exception that the ciphertexts are generated as follows. For each $j \in [k+s]$, $e_j^1 = \mathsf{PKE.Enc}(\mathsf{pk}_1, u_j^1; r_j^1)$, where for each $j \in [k]$, $u_j^1 := \sigma_j^1 \| m_{1j}^*$ and $\sigma_j^1 \leftarrow \mathsf{HS.Sign}(\mathsf{hsk}, \mathsf{tag}^*, m_{1j}^*, j)$, while for each $j \in [k+1, k+s]$, $u_j^1 := \sigma_j^1 \| f_{j-k}(\overrightarrow{m}_1^*)$ and $\sigma_j^1 \leftarrow \mathsf{HS.Evaluate}(\mathsf{hpk}, \mathsf{tag}^*, f_{j-k}, (\sigma_1^1, \dots, \sigma_k^1))$. (The NIZK is still simulated and the obfuscated programs are created for the programme $\mathsf{Prog}_\mathsf{V}^1$ and $\mathsf{Prog}_\mathsf{E}^1$, respectively.) Since the PKE system is IND-CPA secure, Hyb_4 is indistinguishable from Hyb_3.

Hyb_5: The ciphertext and the crs are formed the same way as in Hyb_4 with the exception that the challenger generates the obfuscation of the programme $\mathsf{Prog_V}$ defined in 2a and the programme $\mathsf{Prog_E}$ defined in Figure 2b instead of the obfuscation for the programme $\mathsf{Prog}_\mathsf{V}^1$ defined in Figure 3a and the programme $\mathsf{Prog}_\mathsf{E}^1$ defined in Figure 3b. Since $i\mathcal{O}$ is an indistinguishability obfuscator, Hyb_5 is indistinguishable from Hyb_4.

Hyb_6: This hybrid is the same as in Hyb_5 with the exception that the crs is generated from an honest run of the $\mathsf{NIZK.Setup}$ algorithm and the NIZK proof components. This corresponds to the game when the signatures are generated on $\overrightarrow{m}_1^*$ and $f_i(\overrightarrow{m}_1^*)$. Since the SSS-NIZK system is computationally zero knowledge, Hyb_6 is indistinguishable from Hyb_5. □

4 | HOMOMORPHIC SIGNCRYPTION WITH PUBLIC PLAINTEXT-RESULT Checkability IN a PRIVATE EVALUATION SETTING: DEFINITION AND BASIC CONSTRUCTION

Definition 10 (Homomorphic Signcryption in a private evaluation setting) *A homomorphic signcryption HSC scheme in a private evaluation setting is a tuple of probabilistic, polynomial-time algorithms* KGen_S, KGen_R, $\mathsf{Signcrypt}$, $\mathsf{Unsigncrypt}$, $\mathsf{Evaluate}$, Verify *as follows:*

- $\mathsf{KGen}_S(1^\lambda, k)$: It takes as inputs the security parameter λ and a maximum size k of a dataset, whose messages can be signcrypted. It outputs a sender's key-pair $(\mathsf{pk}_S, \mathsf{sk}_S)$ and defines a message space $\mathcal{M}$, a signcryption space $\mathcal{C}$, and a set $\mathcal{F}$ of functions $f : \mathcal{M}^k \to \mathcal{M}$.
- $\mathsf{KGen}_R(1^\lambda, k)$: It takes as inputs the security parameter λ and a maximum size k of a dataset, and outputs a receiver's key-pair $(\mathsf{pk}_R, \mathsf{sk}_R)$, together with a public verification key vk and a private evaluation key ek.
- $\mathsf{Signcrypt}(\mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}, m, i)$: It takes as inputs the sender's private key sk_S, the receiver's public key pk_R, a tag $\mathsf{tag} \in \{0,1\}^\lambda$, a message $m \in \mathcal{M}$ and its corresponding index $i \in [k]$, and outputs a signcryption $c \in \mathcal{C}$.
- $\mathsf{UnSigncrypt}(\mathsf{pk}_S, \mathsf{sk}_R, c)$: It takes as inputs the sender's public key pk_S, the receiver's secret key sk_R, a signcryption $c \in \mathcal{C}$ and it outputs a message $m \in \mathcal{M}$ together with its corresponding tag tag.
- $\mathsf{Evaluate}(\mathsf{ek}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, f, \overrightarrow{c})$: It takes as inputs an evaluation key ek, sender's public key pk_S, receiver's public key pk_R, a tag $\mathsf{tag} \in \{0,1\}^\lambda$, a function $f \in \mathcal{F}$, and a tuple of signcryptions $\overrightarrow{c} \in \mathcal{C}^k$, and it outputs a derived signcryption $c' \in \mathcal{C}$.
- $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, m', c', f)$: It takes as inputs a public verification key vk, sender's public key pk_S, receiver's public key pk_R, a tag $\mathsf{tag} \in \{0,1\}^\lambda$, a message $m' \in \mathcal{M}$, a function $f \in \mathcal{F}$, and a derived signcryption $c' \in \mathcal{C}$, and it outputs either 0 (reject) or 1 (accept).

Let $\{\Phi_i : \mathcal{M}^k \to \mathcal{M}\}$ be the function $\Phi_i(m_1, \dots, m_k) = m_i$ that projects onto the i-th component and

FIGURE 3 The description of the programs $\mathsf{Prog}_{\mathsf{V}}^{1}$ and $\mathsf{Prog}_{\mathsf{E}}^{1}$

(a)

$\mathsf{Prog}_{\mathsf{V}}^{1}$

Hardwired into the circuit: sk_2, crs.
Input to the circuit: $c = (e_1, e_2, \pi)$, m, tag, hpk, f.
Algorithm:

1. Check that π is a valid NIZK proof using the NIZK.Verify algorithm and crs for the NP-statement (1);
2. If the check fails output $\perp$; otherwise compute $\sigma' \| m' = \mathsf{PKE.Dec}(\mathsf{sk}_2, e_2)$ and run $b \leftarrow \mathsf{HS.Verify}\ (\mathsf{hpk}, \mathsf{tag}, m, \sigma', f) = 1$;
3. Output b.

The program $\mathsf{Prog}_{\mathsf{V}}^{1}$

(b)

$\mathsf{Prog}_{\mathsf{E}}^{1}$

Hardwired into the circuit: sk_2, crs.
Input to the circuit: $\vec{c} = (c_1, \ldots, c_k)$, tag, hpk, f.
Algorithm:

1. For all $i \in [k]$ parse $c_i = (e_{i1}, e_{i2}, \pi_i)$ and check that π_i is valid NIZK proof using the NIZK.Verify algorithm and crs for the NP-statement (1);
2. If the check fails output $\perp$; otherwise for all $i \in [k]$ compute $\sigma_i \| m_i = \mathsf{PKE.Dec}(\mathsf{sk}_2, e_{i2})$;
3. Check that $\mathsf{HS.Verify}(\mathsf{hpk}, \mathsf{tag}, m_i, \sigma_i) = 1$. If it fails, output $\perp$; otherwise, compute $\hat{m} = f(m_1, \ldots, m_k)$ and $\hat{\sigma} \leftarrow \mathsf{HS.Evaluate}(\mathsf{hpk}, \mathsf{tag}, f, (\sigma_1, \ldots, \sigma_k))$; 4. Output $\hat{\sigma} \| \hat{m}$.

The program $\mathsf{Prog}_{\mathsf{E}}^{1}$

$\Phi_1, \ldots, \Phi_k \notin \mathcal{F}$, which implies that the verification algorithm is only allowed to verify whether the derived signcryption from the homomorphic evaluation operation is a valid signcryption or not.

Correctness. For all $(\mathsf{pk}_S, \mathsf{sk}_S) \leftarrow \mathsf{KGen}_S(1^\lambda, k)$ and. $(\mathsf{pk}_R, \mathsf{sk}_R)$
$\leftarrow \mathsf{KGen}_R(1^\lambda, k)$, we have:

1. For all tags $\mathsf{tag} \in \{0,1\}^\lambda$, $m \in \mathcal{M}$, and $i \in \{1, \ldots, k\}$, if $c \leftarrow \mathsf{Signcrypt}(\mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}, m, i)$, then with overwhelming it holds that probability $\mathsf{UnSigncrypt}(\mathsf{pk}_S, \mathsf{sk}_R, c) = m$.
2. For all tags $\mathsf{tag} \in \{0,1\}^\lambda$, all tuples $\vec{m} = (m_1, \ldots, m_k) \in \mathcal{M}^k$, and all functions $f \in \mathcal{F}$, if $c_i \leftarrow \mathsf{Signcrypt}(\mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}, m_i, i)$ for $i = 1, \ldots, k$, then with overwhelming probability it holds that

$\mathsf{Verify}(\mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, \quad \mathsf{Evaluate}(\mathsf{ek}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, f, (c_1, \ldots, c_k)), f) = 1$.

We say that a signcryption scheme as above is $\mathcal{F}$-homomorphic, or homomorphic with respect to $\mathcal{F}$.

Remark 3 *We note that the* Unsigncrypt *algorithm is allowed to perform on both the original and the derived signcryptions, while the verification algorithm can only accept the derived signcryptions as inputs . The reason for this limitation is to protect the secrecy of the message in the signcryption scheme. More precisely, if the public verification algorithm is allowed to operate on the original signcryption of a message, then it is trivial for any adversary to test the matching of the message and the challenged signcryption via the public verification algorithm.*

4.1 | Unforgeability

Definition 11 (Unforgeability) *An $\mathcal{F}$-homomorphic signcryption scheme in a private evaluation setting* $\mathsf{HSC} = (\mathsf{KGen}_S, \ \mathsf{KGen}_R, \ \mathsf{Signcrypt}, \ \mathsf{Unsigncrypt}, \ \mathsf{Evaluate}, \ \mathsf{Verify})$ *is unforgeable if for all k no PPT adversary $\mathcal{A}$ can win the following defined experiment* $\mathsf{Expt}_{\mathsf{HSC},\mathcal{A}}^{\mathsf{UF}}(1^\lambda)$ *with non-negligible probability.*

- The challenger runs $(\mathsf{pk}_S, \mathsf{sk}_S) \leftarrow \mathsf{KGen}_S(1^\lambda, k)$ and $(\mathsf{pk}_R, \mathsf{sk}_R) \leftarrow \mathsf{KGen}_R(1^\lambda, k)$, and sends $\mathsf{pk}_S, \mathsf{pk}_R$ to the adversary $\mathcal{A}$.

- $\mathcal{A}$ proceeds with adaptive queries,
  - SIGNCRYPTION QUERIES. Each query consists of:
- a dataset given a k-message vector $\vec{m}_i = \{m_{i,1}, \ldots, m_{i,k}\}$.

- For each i, the challenger sends backa randomly chosen dataset tag $\mathsf{tag}_i \in \{0,1\}^\lambda$, and a signcryption vector $\overrightarrow{c}_i = (c_{i,1}, \ldots, c_{i,k})$ where $c_{ij} \leftarrow \mathsf{Signcrypt}$ $(\mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}_i, m_{ij})$ for all $j \in [k]$.
  - UNSIGNCRYPTION QUERIES. Each query consists of:
- a signcryption $c_\iota \in \mathcal{C}$
- For each ι, challenger sends back $m_\iota \leftarrow \mathsf{UnSigncrypt}(\mathsf{pk}_S, \mathsf{sk}_R, c_\iota)$.
  - EVALUATION QUERIES. Each query consists of:
- a k-signcryption vector $\overrightarrow{c}_J = \{c_{J,1}, \ldots, c_{J,k}\} \in \mathcal{C}^k$ together with a tag tag_J and a function $f_J \in \mathcal{F}$.
- The challenger sends back $\widehat{c}_J \leftarrow \mathsf{Evaluate}(\mathsf{ek}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}_J, f_J, \overrightarrow{c}_J)$ along with tag_J.
- $\mathcal{A}$ outputs two main kinds of forged tuple that can be classified as follows:
  1. (type 1) a tuple (tag^*, c^*), where $\overrightarrow{c}^*$ is a signcryption s. t (1) $\mathsf{UnSigncrypt}(\mathsf{pk}_S, \mathsf{sk}_R, c^*) \neq \perp$ and (2) tag^* is different from all the tags associated with dataset that has been queried to the signcryption oracle and the evaluation oracle, or
  2. (type 2) a tuple $(\mathsf{tag}^*, m^*, f^*, c^*)$ s. t (1) $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}^*, m^*, c^*, f^*) = 1$ and (2) $\mathsf{tag}^* \neq \mathsf{tag}_J$ for all J where tag_J denotes the tag that has been submitted to the evaluation oracle, or
  3. (type 3) a tuple $(\mathsf{tag}^*, m^*, f^*, c^*)$ s. t (1) $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}^*, m^*, c^*, f^*) = 1$ and (2) $\mathsf{tag}^* = \mathsf{tag}_J$ for some J where tag_J denotes the tag that has been submitted to the evaluation oracle, but f^* has never been queried to the evaluation oracle, or
  4. (type 4) a tuple $(\mathsf{tag}^*, m^*, f^*, c^*)$ s. t (1) $\mathsf{Verify}(\mathsf{vk}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}^*, m^*, c^*, f^*) = 1$ and (2) $\mathsf{tag}^* = \mathsf{tag}_J$ for some J where tag_J denotes the tag that has been submitted to the evaluation oracle, and f^* has been queried to the evaluation oracle, but $m^* \neq f^*(\overrightarrow{m}_J)$ where $\overrightarrow{m}_J$ is the dataset corresponding to the tag tag_J.

4.2 | Message privacy

Contrary to the weak message privacy for HSC in a public evaluation setting in a private evaluation, the privacy notion is defined for the full message confidentiality, which captures the idea that given both the original signcryptions on the dataset and the signcryptions on a number of messages derived from one of two different datasets, the attacker cannot tell which dataset the derived signatures came from.

Definition 12 (Message Privacy) *An $\mathcal{F}$-homomorphic signcryption scheme in a private evaluation setting* HSC *is message private if for all k no PPT adversary $\mathcal{A}$ can win the following defined experiment* $\mathsf{Expt}^{\mathsf{MP}}_{\mathsf{HSC},\mathcal{A}}(1^\lambda)$ *with non-negligible advantage.*

- The challenger runs $(\mathsf{pk}_S, \mathsf{sk}_S) \leftarrow \mathsf{KGen}_S(1^\lambda, k)$ and $(\mathsf{pk}_R, \mathsf{sk}_R) \leftarrow \mathsf{KGen}_R(1^\lambda, k)$, and sends $\mathsf{pk}_S, \mathsf{pk}_R$ to $\mathcal{A}$.
- $\mathcal{A}$ adaptively proceeds with signcryption and evaluation queries as in the experiment $\mathsf{Expt}^{\mathsf{UF}}_{\mathsf{HSC},\mathcal{A}}$.
- $\mathcal{A}$ outputs $(\overrightarrow{m}^*_0, \overrightarrow{m}^*_1, f_1, \ldots, f_s)$ with $\overrightarrow{m}^*_0, \overrightarrow{m}^*_1 \in \mathcal{M}^k$. The functions $f_1, \ldots, f_s$ are in $\mathcal{F}$ and satisfy $f_i(\overrightarrow{m}^*_0) = f_i(\overrightarrow{m}^*_1)$ for all $i \in [s]$.
- The challenger generates a random bit $b \in \{0, 1\}$ and a random tag $\mathsf{tag} \in \{0,1\}^\lambda$. It signcrypts the messages in $\overrightarrow{m}^*_b$ using tag to obtain a vector $\overrightarrow{c}$ of k signcryptions, where $c_j \leftarrow \mathsf{Sign}(\mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}, m*_{bj}, j)$ for all $j \in [k]$. Next, for each $i \in [s]$ the challenger computes a derived signcryption $\widehat{c}_i \leftarrow \mathsf{Evaluate}(\mathsf{ek}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, f_i, \overrightarrow{c})$. It sends tag and the derived signcryptions $(\widehat{c}_1, \ldots, \widehat{c}_s)$ as well as original signcryption vector $\overrightarrow{c}$ to $\mathcal{A}$.
- $\mathcal{A}$ adaptively performs signcryption and evaluation queries as before.
- $\mathcal{A}$ outputs a bit b'. $\mathcal{A}$ wins the game if $b = b'$.

4.3 | Construction of an HSC in a private evaluation setting

In this section, we present an HSC scheme in a private evaluation setting from an HS scheme *without* context-hiding and a FE scheme. Our construction relies on the following building blocks:

- An $\mathcal{F}$-homomorphic signature scheme $\mathcal{HS} = (\mathsf{HS.Setup}, \mathsf{HS.Sign}, \mathsf{HS.Evaluate}, \mathsf{HS.Verify})$ with message of length $|m|$ and a signature of length ℓ_{sig}.
- A general-purpose public-key multi-input functional encryption scheme $\mathcal{MIFE} = (\mathsf{MIFE.Setup}, \mathsf{MIFE.Enc}, \mathsf{MIFE.KeyGen}, \mathsf{MIFE.Dec})$.

Our HSC scheme $\mathsf{HSC} = (\mathsf{KGen}_S, \mathsf{KGen}_R, \mathsf{Signcrypt}, \mathsf{Unsigncrypt}, \mathsf{Evaluate}, \mathsf{Verify})$ with respect to $\mathcal{F}$ is built as follows.

- **KGen**$_S(1^\lambda, k) \rightarrow (\mathsf{pk}_S, \mathsf{sk}_S)$:
  - Sample $(\mathsf{hsk}, \mathsf{hpk}) \leftarrow \mathsf{HS.Setup}(1^\lambda)$. Set $\mathsf{pk}_S := \mathsf{hpk}$ and $\mathsf{sk}_S := \mathsf{hsk}$.
- **KGen**$_R(1^\lambda, k) \rightarrow (\mathsf{pk}_R, \mathsf{sk}_R, \mathsf{vk}, \mathsf{ek})$:
  - Sample $(\mathsf{msk}^0, \mathsf{mpk}^0) \leftarrow \mathsf{MIFE.Setup}(1^\lambda)$, $(\mathsf{msk}^1, \mathsf{mpk}^1) \leftarrow \mathsf{MIF\ E.Setup}(1^\lambda)$;
  - Compute $\mathsf{sk}^U_{\mathsf{FE}} \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{msk}^0, U)$ where the function U is described in Figure 4a;
  - Compute $\mathsf{sk}^G_{\mathsf{FE}} \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{msk}^1, G)$ where the function G is described in Figure 4b;
  - Set $\mathsf{pk}_R := (\mathsf{mpk}^0, \mathsf{mpk}^1)$, $\mathsf{sk}_R := (\mathsf{msk}^0, \mathsf{msk}^1)$, the evaluation key $\mathsf{ek} := \mathsf{sk}^U_{\mathsf{FE}}$ and verification key $\mathsf{vk} := \mathsf{sk}^G_{\mathsf{FE}}$.
- **Signcryption**$(\mathsf{sk}_S, \mathsf{pk}_R, \mathsf{tag}, \overrightarrow{x}, i) \rightarrow c_i$:
  - Parse $\mathsf{sk}_S = \mathsf{hsk}$, $\mathsf{pk}_R = (\mathsf{mpk}^0, \mathsf{mpk}^1)$ and $\overrightarrow{x} = (x_1, \ldots, x_k)$.

- Using the tag $\mathsf{tag} \in \{0,1\}^\lambda$ generate a k-signature vector $\vec{\sigma} = (\sigma_1, \ldots, \sigma_k)$ for message $\vec{x}$, where $\sigma_i \leftarrow \mathsf{HS.Sign}(\mathsf{hsk}, \mathsf{tag}, x_i, i)$.
- Compute $\mathsf{ct}_i \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^0, \mathsf{tag}\|\sigma_i\|x_i\|\Phi_i)$, $\mathsf{ct}_{\mathsf{tag}} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \mathsf{tag})$, $\mathsf{ct}_{\sigma_i} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \sigma_i)$, $\mathsf{ct}_{x_i} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, x_i)$, and $\mathsf{ct}_{\Phi_i} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \Phi_i)$;
- Output $c_i = (0, \mathsf{ct}_i, \mathsf{ct}_{\mathsf{tag}}, \mathsf{ct}_{\sigma_i}, \mathsf{ct}_{x_i}, \mathsf{ct}_{\Phi_i})$.

- **UnSigncryption**$(\mathsf{pk}_S, \mathsf{sk}_R, c_i) \to (\mathsf{tag}, x_i)$:
  - Parse $\mathsf{pk}_S = \mathsf{hpk}$, $\mathsf{sk}_R = (\mathsf{msk}^0, \mathsf{msk}^1)$, and $c_i = (\beta, \mathsf{ct}_i, \mathsf{ct}_{\mathsf{tag}}, \mathsf{ct}_{\sigma_i}, \mathsf{ct}_{x_i}, \mathsf{ct}_f)$.
  - Compute $\mathsf{tag}\|\sigma_i\|x_i\|f \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^0, \mathsf{ct}_i)$.
  - Run $\mathsf{tag}' \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^1, \mathsf{ct}_{\mathsf{tag}})$, $\sigma_i' \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^1, \mathsf{ct}_{\sigma_i})$, $x_i' \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^1, \mathsf{ct}_{x_i})$, and $f' \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^1, \mathsf{ct}_f)$.
  - Check if $\mathsf{tag} = \mathsf{tag}'$, $\sigma_i = \sigma_i'$, $x_i = x_i'$ and $f = f'$. If they hold, then check if $\mathsf{HS.Verify}(\mathsf{hpk}, \mathsf{tag}, x_i, \sigma_i, f) = 1$; If it is true, then output (tag, x_i). Otherwise, abort.
- **Evaluate**$(\mathsf{pk}_S, \mathsf{pk}_R, \mathsf{ek}, \mathsf{tag}, \{c_i\}_{i\in[k]}, f) \to \hat{c}$:
  - Parse $\mathsf{ek} = \mathsf{sk}_{\mathsf{FE}}^U$, $\mathsf{pk}_R = (\mathsf{mpk}^0, \mathsf{mpk}^1)$ and $c_i = (\beta, \mathsf{ct}_i, \mathsf{ct}_{\mathsf{tag}}, \mathsf{ct}_{\sigma_i}, \mathsf{ct}_{x_i}, \mathsf{ct}_{\Phi_i})$. If $\beta = 1$, abort; otherwise, proceeds as following.
  - Encrypt the function f under the public key mpk^0 of MIFE, namely, $\mathsf{ct}_f \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^0, f)$. Decrypt the ciphertexts with the function key of FE and obtain $(\mathsf{tag}', y, \hat{\sigma}) \leftarrow \mathsf{MIFE.Dec}(\mathsf{sk}_{\mathsf{FE}}^U, \mathsf{ct}_1, \ldots, \mathsf{ct}_k, \mathsf{ct}_f)$.
  - If $\mathsf{tag} = \mathsf{tag}'$, then compute $\widehat{\mathsf{ct}} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^0, \mathsf{tag}\|\hat{\sigma}\|y\|f)$, $\widehat{\mathsf{ct}}_{\mathsf{tag}} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \mathsf{tag})$, $\widehat{\mathsf{ct}}_{\hat{\sigma}} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \hat{\sigma})$, $\widehat{\mathsf{ct}}_y \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, y)$, and $\widehat{\mathsf{ct}}_f \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, f)$;
  - Output $\hat{c} = (1, \widehat{\mathsf{ct}}, \widehat{\mathsf{ct}}_{\mathsf{tag}}, \widehat{\mathsf{ct}}_{\hat{\sigma}}, \widehat{\mathsf{ct}}_y, \widehat{\mathsf{ct}}_f)$.
- **Verify**$(\mathsf{vk}, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{tag}, y, c, f) \to \{0,1\}$:
  - Parse $\mathsf{pk}_S = \mathsf{hpk}$, $\mathsf{pk}_R = (\mathsf{mpk}^0, \mathsf{mpk}^1)$, $\mathsf{vk} = \mathsf{sk}_{\mathsf{FE}}^G$ and $c = (\beta, \mathsf{ct}, \mathsf{ct}_{\mathsf{tag}}, \mathsf{ct}_\sigma, \mathsf{ct}_y, \mathsf{ct}_f)$;
  - Compute $\mathsf{ct}'_{\mathsf{tag}} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \mathsf{tag})$, $\mathsf{ct}'_y \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, y)$, and $\mathsf{ct}'_f \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, f)$. Then run $b \leftarrow \mathsf{MIFE.Dec}(\mathsf{sk}_{\mathsf{FE}}^G, \mathsf{ct}'_{\mathsf{tag}}, \mathsf{ct}'_y, \mathsf{ct}_\sigma, \mathsf{ct}'_f)$ and output b.

Correctness. The correctness of HSC scheme described above follows immediately from the correctness of MIFE and HS scheme.

Theorem 3 *Assuming the underlying homomorphic signature scheme $\mathcal{HS}$ is existentially unforgeable against chosen message attacks as defined in Definition 4, the homomorphic signcryption scheme described above satisfies unforgeability against chosen message attacks as defined in Definition 11.*

Proof: Let us fix a PPT adversary $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$ attacking our HSC scheme constructed above, we will use $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$ to construct an adversary $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ such that, if $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$ wins in the unforgeability game for our HSC scheme given above with non-negligible probability, then $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ breaks the underlying existential unforgeability of homomorphic signature scheme $\mathcal{HS}$.

We now describe the constructed HS adversary, $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$. After receiving the challenge verification key hpk of the HS scheme, $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ first generates $(\mathsf{msk}^0, \mathsf{mpk}^0) \leftarrow \mathsf{MIFE.Setup}(1^\lambda)$, $(\mathsf{msk}^1, \mathsf{mpk}^1) \leftarrow \mathsf{MIFE.Setup}(1^\lambda)$. Then it computes $\mathsf{sk}_{\mathsf{FE}}^U \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{msk}^0, U)$ for the function U described in Figure 4a and $\mathsf{sk}_{\mathsf{FE}}^G \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{msk}^1, G)$ for the function G described in Figure 4b. $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ sets $\mathsf{pk}_S := \mathsf{hpk}$, $\mathsf{pk}_R := (\mathsf{mpk}^0, \mathsf{mpk}^1)$, $\mathsf{sk}_R := (\mathsf{msk}^0, \mathsf{msk}^1)$, $\mathsf{vk} := \mathsf{sk}_{\mathsf{FE}}^G$, and $\mathsf{ek} := \mathsf{sk}_{\mathsf{FE}}^U$. $\mathsf{pk}_S, \mathsf{pk}_R, \mathsf{vk}$ are given to $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$.

To answer the i-th query submitted to the signcryption oracle, that is, a k-message vector $\vec{m}_i = \{m_{i1}, \ldots, m_{ik}\}$ issued by $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$, $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ performs the following:

- Sends $\vec{m}_i = \{m_{i1}, \ldots, m_{ik}\}$ to its own signing oracle $\mathsf{O}_{\mathsf{HS}_{\mathsf{sig}}}$ to get a tag tag_i and a k-signature vector $\vec{\sigma}_i = (\sigma_{i1}, \ldots, \sigma_{ik})$.

(a)

$U_{\mathsf{hpk}}(\mathsf{tag}_1\|\sigma_1\|x_1\|f_1, \ldots, \mathsf{tag}_k\|\sigma_k\|x_k\|f_k, f)$

1. For all $i \in [k]$ check if $\mathsf{HS.Verify}(\mathsf{hpk}, \mathsf{tag}_i, x_i, \sigma_i, f_i) = 1$ and tag_i are the same; If it is false, output $\perp$.
2. Else, denote the same tag_i as tag and compute $\hat{\sigma} \leftarrow \mathsf{HS.Evaluate}(\mathsf{hpk}, \mathsf{tag}, f, (\sigma_1, \ldots, \sigma_k))$ and $y = f(x_1, \ldots, x_k)$.
3. Output tag, y and $\hat{\sigma}$.

The function U_{hpk}

(b)

$G_{\mathsf{hpk}}(\mathsf{tag}, \sigma, y, f)$

1. Run $\mathsf{HS.Verify}(\mathsf{hpk}, \mathsf{tag}, y, \sigma, f) \to b$ and output b.

The function G_{hpk}

FIGURE 4 Description of the function U_{hpk} and G_{hpk}

- For each $j \in [k]$, run $\mathsf{ct}_{ij} \leftarrow \mathsf{MIFE.Enc}\,(\mathsf{mpk}^0, \mathsf{tag}_i \| \sigma_{ij} \| m_{ij} \| \Phi_i)$, $\mathsf{ct}_{\mathsf{tag}_i} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \mathsf{tag}_i)$, $\mathsf{ct}_{\sigma_{ij}} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \sigma_{ij})$, $\mathsf{ct}_{m_{ij}} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, m_{ij})$, and $\mathsf{ct}_{\Phi_i} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \Phi_i)$. Then set $c_{ij} = (0, \mathsf{ct}_{ij}, \mathsf{ct}_{\mathsf{tag}_i}, \mathsf{ct}_{\sigma_{ij}}, \mathsf{ct}_{m_{ij}}, \mathsf{ct}_{\Phi_i})$.
- Return $\left(\mathsf{tag}_i, \overrightarrow{c}_i = \{c_{i1}, \ldots, c_{ik}\}\right)$ back to $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$.

To answer the query submitted to the unsigncryption oracle, that is, a signcryption $c_\iota = (\beta, \mathsf{ct}_\iota, \mathsf{ct}_{\mathsf{tag}_\iota}, \mathsf{ct}_{\sigma_\iota}, \mathsf{ct}_{m_\iota}, \mathsf{ct}_f)$ with corresponding tag tag_ι issued by $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$, $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ performs the following:

- Computes $\mathsf{tag}_\iota \| \sigma_\iota \| m_\iota \| f \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^0, \mathsf{ct}_\iota)$;
- Run $\mathsf{tag}'_\iota \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^1, \mathsf{ct}_{\mathsf{tag}_\iota})$, $\sigma'_\iota \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^1, \mathsf{ct}_{\sigma_\iota})$, $m'_\iota \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^1, \mathsf{ct}_{m_\iota})$, and $f' \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^1, \mathsf{ct}_f)$.
- Check if $\mathsf{tag}_\iota = \mathsf{tag}'_\iota$, $\sigma_\iota = \sigma'_\iota$, $m_\iota = m'_\iota$ and $f = f'$. If they hold, then check if $\mathsf{HS.Verify}(\mathsf{hpk}, \mathsf{tag}_\iota, m_\iota, \sigma_\iota, f) = 1$; If it is true, then output m_ι. Otherwise, abort.

To answer the query to the evaluation oracle, that is, k-signcryption vector $\overrightarrow{c}_J = \{c_{J1}, \ldots, c_{Jk}\}$ with corresponding tag tag_J and function f_J issued by $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$, $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ performs the following:

- Compute $\mathsf{ct}_{f_J} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^0, f)$ and $(\mathsf{tag}'_J, y_J, \widehat{\sigma}_J) \leftarrow \mathsf{MIFE.\ Dec}(\mathsf{sk}_{\mathsf{FE}}^U, \mathsf{ct}_{J1}, \ldots, \mathsf{ct}_{Jk}, \mathsf{ct}_{f_J})$.
- If $\mathsf{tag}_J = \mathsf{tag}'_J$, then run $\widehat{\mathsf{ct}}_J \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^0, \mathsf{tag}_J \| \widehat{\sigma}_J \| y_J \| f_J)$, $\widehat{\mathsf{ct}}_{\mathsf{tag}_J} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \mathsf{tag}_J)$, $\widehat{\mathsf{ct}}_{\widehat{\sigma}_J} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \widehat{\sigma}_J)$, $\widehat{\mathsf{ct}}_{y_J} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, y_J)$, and $\widehat{\mathsf{ct}}_{f_J} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, f_J)$;
- Return $\widehat{c}_J = (1, \widehat{\mathsf{ct}}_J, \widehat{\mathsf{ct}}_{\mathsf{tag}_J}, \widehat{\mathsf{ct}}_{\widehat{\sigma}_J}, \widehat{\mathsf{ct}}_{y_J}, \widehat{\mathsf{ct}}_{f_J})$.

Eventually, $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$ outputs a k-signcryption vector $\overrightarrow{c}^* = (c_1^*, \ldots, c_k^*)$ together with a tag tag^* or a tuple $(\mathsf{tag}^*, m^*, c^*, f^*)$. We then analyse the following two cases.

Case 1: When $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$ outputs a k-signcryption vector $\overrightarrow{c}^* = (c_1^*, \ldots, c_k^*)$ along with a tag tag^*. Since $\mathsf{UnSigncrypt}(\mathsf{usk}, c_i^*) \neq \perp$ for all $i \in [k]$, $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ decrypts c_i^* using the master secret key of FE to obtain $\mathsf{tag}^* \| \sigma'_i \| m'_i \| f_i^* \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^0, c_i^*)$ for all $i \in [k]$, thus resulting in k-message/signature pairs $\{(m'_i, \sigma'_i)\}_{i \in [k]}$. As it is required that tag^* is different from all the tags associated with the dataset that has been queried to the signcryption oracle and the evaluation oracle, $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ outputs $(\overrightarrow{m}', \overrightarrow{\sigma}')$ together with the tag tag^* as a forgery, which is the forgery of HS of type 1 defined in Definition 4.

Case 2: When $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$ outputs a tuple $(\mathsf{tag}^*, m^*, c^*, f^*)$ such that $\mathsf{HSC.Verify}(\mathsf{pk}, \mathsf{tag}^*, m^*, c^*, f^*) = 1$, where $c^* = \left(\beta, \mathsf{ct}^*, \mathsf{ct}^*_{\mathsf{tag}^*}, \mathsf{ct}^*_\sigma, \mathsf{ct}^*_m, \mathsf{ct}^*_f\right)$. $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ computes $\mathsf{ct}'_{\mathsf{tag}^*} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \mathsf{tag}^*)$, $\mathsf{ct}'_{m^*} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, m^*)$ and $\mathsf{ct}'_{f^*} \leftarrow \mathsf{MIFE.Enc}\,(\mathsf{mpk}^1, f^*)$. Then, from $\mathsf{HSC.Verify}(\mathsf{pk}, \mathsf{tag}^*, m^*, c^*, f^*) = 1$, we have. $\mathsf{MIFE.Dec}(\mathsf{sk}_{\mathsf{FE}}^G, \mathsf{ct}'_{\mathsf{tag}^*}, \mathsf{ct}'_{y^*}, \mathsf{ct}^*_\sigma, \mathsf{ct}'_{f^*}) = 1$, which implies. $\mathsf{HS.Verify}$ $\mathsf{hpk}, \mathsf{tag}^*, m^*, \sigma, f^*) = 1$, where $\sigma \leftarrow \mathsf{MIFE.Dec}(\mathsf{msk}^1, \mathsf{ct}^*_\sigma)$. $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ outputs $(\mathsf{tag}^*, m^*, \sigma, f^*)$ as its message-forgery pair in the unforgeability game for the underlying $\mathcal{HS}$ scheme, which means that the adversary $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ successfully outputs a forgery of $\mathcal{HS}$ scheme of type 2 defined in Definition 4.

Thus, if $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$ produces a forgery in our HSC scheme with non-negligible probability $1/Poly(\lambda)$, then $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ successfully forges in the underlying $\mathcal{HS}$ scheme with non-negligible probability $1/Poly(\lambda)$. But, this cannot be the case, since we have assumed that the $\mathcal{HS}$ scheme is existentially unforgeable against chosen-message attacks. We conclude that our HSC scheme as specified above satisfies the unforgeability security of Definition 11. □

Theorem 4 *Assuming the underlying public key functional encryption scheme $\mathcal{FE}$ is secure, the homomorphic signcryption scheme described above is message private as defined in Definition* 12 *for datasets up to k.*

Proof: Let $\mathcal{A}_{\mathsf{MP}}$ be an adversary attacking our HSC scheme constructed above in the sense of message privacy. We construct an adversary $\mathcal{A}_{\mathsf{FE}}$ attacking the security of the MIFE, and then upper bound the advantage of $\mathcal{A}$ in terms of the advantages of these adversaries.

We now describe the constructed public-key multi-input functional encryption adversary, $\mathcal{A}_{\mathsf{FE}}$. $\mathcal{A}_{\mathsf{FE}}$ interacts with $\mathcal{A}_{\mathsf{MP}}$, playing the role of the challenger in the game of message privacy for HSC. This means that $\mathcal{A}_{\mathsf{FE}}$ must simulate the signcryption oracle and the evaluation oracle.

After receiving the master public key $\mathsf{mpk}^0, \mathsf{mpk}^1$ of the MIFE scheme, $\mathcal{A}_{\mathsf{FE}}$ samples a key pair of HS, $(\mathsf{hsk}, \mathsf{hpk}) \leftarrow \mathsf{HS.Setup}(1^\lambda, k)$. $\mathcal{A}_{\mathsf{FE}}$ sends the function U defined in Figure 4a and the function G defined in Figure 4b to the MIFE scheme's challenger and obtains the secret keys $\mathsf{sk}_{\mathsf{FE}}^U$ and $\mathsf{sk}_{\mathsf{FE}}^G$ as responses, respectively. $\mathcal{A}_{\mathsf{HS}}^{\mathsf{Unf}}$ sets $\mathsf{pk}_S := \mathsf{hpk}$, $\mathsf{pk}_R := (\mathsf{mpk}^0, \mathsf{mpk}^1)$, $\mathsf{sk}_R := (\mathsf{msk}^0, \mathsf{msk}^1)$, $\mathsf{vk} := \mathsf{sk}_{\mathsf{FE}}^G$, and $\mathsf{ek} := \mathsf{sk}_{\mathsf{FE}}^U$. $\mathsf{pk}_S, \mathsf{pk}_R, \mathsf{vk}$ are given to $\mathcal{A}_{\mathsf{HSC}}^{\mathsf{Unf}}$. To simulate the signcryption oracle, for the i-th query, a k-message vector $\overrightarrow{m}_i = \{m_{i1}, \ldots, m_{ik}\}$ issued by $\mathcal{A}_{\mathsf{MP}}$, $\mathcal{A}_{\mathsf{FE}}$ performs as following:

- Generate a random $\mathsf{tag}_i \in \{0,1\}^\lambda$ and sign the message in $\overrightarrow{m}_i$ using the tag tag_i to obtain a k-signature vector $\overrightarrow{\sigma}_i = (\sigma_{i1}, \ldots, \sigma_{ik})$, where $\sigma_{ij} \leftarrow \mathsf{HS.Sign}(\mathsf{hsk}, \mathsf{tag}_i, \overrightarrow{m}_i, j)$ for alright $j \in [k]$.
- For each $j \in [k]$, run $\mathsf{ct}_{ij} \leftarrow \mathsf{MIFE.Enc}\,(\mathsf{mpk}^0, \mathsf{tag}_i \| \sigma_{ij} \| m_{ij} \| \Phi_i)$, $\mathsf{ct}_{\mathsf{tag}_i} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \mathsf{tag}_i)$, $\mathsf{ct}_{\sigma_{ij}} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \sigma_{ij})$, $\mathsf{ct}_{m_{ij}} \leftarrow \mathsf{MIFE.Enc}\,(\mathsf{mpk}^1, m_{ij})$, and $\mathsf{ct}_{\Phi_i} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \Phi_i)$. Then set $c_{ij} = (0, \mathsf{ct}_{ij}, \mathsf{ct}_{\mathsf{tag}_i}, \mathsf{ct}_{\sigma_{ij}}, \mathsf{ct}_{m_{ij}}, \mathsf{ct}_{\Phi_i})$.
- Return $\left(\mathsf{tag}_i, \overrightarrow{c}_i = \{c_{i1}, \ldots, c_{ik}\}\right)$ back to $\mathcal{A}_{\mathsf{MP}}$.

To answer the query to evaluation oracle, that is, k-signcryption vector $\overrightarrow{c}_J = \{c_{J1}, \ldots, c_{Jk}\}$ with corresponding tag tag_J and function f_J issued by $\mathcal{A}_{\mathsf{MP}}$, $\mathcal{A}_{\mathsf{FE}}$ performs as following:

- Compute $\mathsf{ct}_{f_J} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^0, f)$ and $(\mathsf{tag}'_J, y_J, \widehat{\sigma}_J) \leftarrow \mathsf{MIFE.\ Dec}(\mathsf{sk}_{\mathsf{FE}}^U, \mathsf{ct}_{J1}, \ldots, \mathsf{ct}_{Jk}, \mathsf{ct}_{f_J})$.

- If $\mathsf{tag}_j = \mathsf{tag}'_j$, then run $\widehat{\mathsf{ct}}_j \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^0, \mathsf{tag}_j \| \widehat{\sigma}_j \| y_j \| f_j)$, $\widehat{\mathsf{ct}}_{\mathsf{tag}_j} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \mathsf{tag}_j)$, $\widehat{\mathsf{ct}}_{\widehat{\sigma}_j} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \widehat{\sigma}_j)$, $\widehat{\mathsf{ct}}_{y_j} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, y_j)$, and $\widehat{\mathsf{ct}}_{f_j} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, f_j)$;
- Return $\widehat{c}_j = (1, \widehat{\mathsf{ct}}_j, \widehat{\mathsf{ct}}_{\mathsf{tag}_j}, \widehat{\mathsf{ct}}_{\widehat{\sigma}_j}, \widehat{\mathsf{ct}}_{y_j}, \widehat{\mathsf{ct}}_{f_j})$.

After receiving a tuple $(\overrightarrow{m}_0^*, \overrightarrow{m}_1^*, f_1, \ldots, f_s)$ with $\overrightarrow{m}_0^*, \overrightarrow{m}_1^* \in \mathcal{M}^k$ from $\mathcal{A}_{\mathsf{MP}}$, where functions $f_1, \ldots, f_s$ belong to $\mathcal{F}$ and satisfy $f_i(\overrightarrow{m}_0^*) = f_i(\overrightarrow{m}_1^*)$ for all $i \in [s]$, $\mathcal{A}_{\mathsf{FE}}$ randomly samples two tags tag_0^* and tag_1^* and signs two message vectors $\overrightarrow{m}_b^*$ using the tag tag_b^* to obtain $\sigma_{bj} \leftarrow \mathsf{HS.Sign}(\mathsf{hsk}, \mathsf{tag}_b^*, \overrightarrow{m}_b^*, j)$ for all $j \in [k]$ and all $b \in \{0, 1\}$. Then, $\mathcal{A}_{\mathsf{FE}}$ sets $x_{b,j} := \mathsf{tag}_b^* \| \sigma_{bj} \| m_{bj}^* \| \Phi_j$ for all $j \in [k]$ and all $b \in \{0, 1\}$. It sends $(x_{0,1}, \ldots, x_{0,k})$ and $(x_{1,1}, \ldots, x_{1,k})$ to its own MIFE challenger with respect to mpk^0 and receives the challenged ciphertexts $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_k^*$. For all $j \in [k]$, $\mathcal{A}_{\mathsf{FE}}$ computes $\mathsf{ct}_{\Phi_j}^* \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \Phi_j)$, and sends $(\mathsf{tag}_0^*, \mathsf{tag}_1^*)$, $(\sigma_{0j}, \sigma_{1j})$, (m_{0j}^*, m_{1j}^*) to its own MIFE challenger with respect to mpk^1 and receives the challenged ciphertexts $\mathsf{ct}_{\mathsf{tag}}^*, \mathsf{ct}_{\sigma_j}^*, \mathsf{ct}_{m_j}^*$, respectively. $\mathcal{A}_{\mathsf{FE}}$ sets $c_j^* = (0, \mathsf{ct}_j^*, \mathsf{ct}_{\mathsf{tag}}^*, \mathsf{ct}_{\sigma_j}^*, \mathsf{ct}_{m_j}^*, \mathsf{ct}_{\Phi_j}^*)$. $\mathcal{A}_{\mathsf{FE}}$ computes $\mathsf{ct}_{f_i} \leftarrow \mathsf{MIFE.Enc}\ (\mathsf{mpk}^0, f_i)$ for all $i \in [s]$ and decrypts ciphertexts with function key $\mathsf{sk}_{\mathsf{FE}}^U$ to obtain $(\mathsf{tag}'_i, y_i, \widehat{\sigma}_i) \leftarrow \mathsf{MIFE.Dec}(\mathsf{sk}_{\mathsf{FE}}^U, \mathsf{ct}_1^*, \ldots, \mathsf{ct}_k^*, \mathsf{ct}_{f_i})$. Since tag'_i should be the same, we denote it as tag'. For all $i \in [s]$, $\mathcal{A}_{\mathsf{FE}}$ computes $\widehat{\mathsf{ct}}_i \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^0, \mathsf{tag}' \| \widehat{\sigma}_i \| y_i \| f_i)$, $\widehat{\mathsf{ct}}_{\mathsf{tag}'} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \mathsf{tag}')$, $\widehat{\mathsf{ct}}_{\widehat{\sigma}_i} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, \widehat{\sigma}_i)$, $\widehat{\mathsf{ct}}_{y_i} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, y_i)$, and $\widehat{\mathsf{ct}}_{f_i} \leftarrow \mathsf{MIFE.Enc}(\mathsf{mpk}^1, f_i)$. Then $\mathcal{A}_{\mathsf{FE}}$ sets $\widehat{c}_i = (1, \widehat{\mathsf{ct}}_i, \widehat{\mathsf{ct}}_{\mathsf{tag}'}, \widehat{\mathsf{ct}}_{\widehat{\sigma}_i}, \widehat{\mathsf{ct}}_{y_i}, \widehat{\mathsf{ct}}_{f_i})$. $\mathcal{A}_{\mathsf{FE}}$ sends the tag tag' and the signcryptions $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_k^*, \widehat{c}_1, \ldots, \widehat{c}_s$ to $\mathcal{A}_{\mathsf{MP}}$. Finally, if $\mathcal{A}_{\mathsf{MP}}$ outputs b^* to indicate that the challenged signcryptions are the encoded value of the message $\overrightarrow{m}_{b^*}^*$, then $\mathcal{A}_{\mathsf{FE}}$ returns b^* to indicate that $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_k^*$ are the encryptions of the messages $(x_{b^*,1}, \ldots, x_{b^*,k})$.

Thus, if $\mathcal{A}_{\mathsf{MP}}$ correctly guesses which message the challenger encodes in the game of message privacy of the HSC scheme with non-negligible probability $\epsilon(\lambda)$, then $\mathcal{A}_{\mathsf{FE}}$ correctly guesses which message the challenger encrypts in the underlying IND-CPA game for the MIFE scheme with non-negligible probability $\epsilon(\lambda)$. □

5 | CONCLUSION

In this article, we investigate the question of how to homomorphically perform arbitrary computations on signcrypted data, going beyond the existing additive homomorphic operation. We augment the concept of homomorphic signcryption on two aspects, one of which is to provide public plaintext-result checkability such that anyone is able to publicly check whether a given ciphertext is the signcryption of the message under the key, thus no longer bound to the recipient. Another property that our homomospric signcryption schemes achieve is message privacy. The latter guarantees that the derived signcryption will not reveal any information about the underlying dataset, beyond what is revealed by the outcome of evaluation on the underlying dataset. We also propose constructions of a homomorphic signcryption scheme with public plaintext-result checkability both in a public evaluation setting and a private evaluation setting. We believe that homomorphic signcryptions that achieve both plaintext-result checkability and confidentiality are very useful in a wide variety of settings involving data processing by untrusted entities.

ACKNOWLEDGEMENTS

This work is done when Bei Liang works at the Chalmers University of Technology, Sweden. The work of Bei Liang and Aikaterini Mitrokotsa was partially supported by the VR PRECIS grant. This work of Bei Liang was partially supported by the National Natural Science Foundation of China [grant number 61972124]. This work of Shimin Li and Rui Xue was supported by National Natural Science Foundation of China [grant number 61472414, 61772514, 61602061], and National Key R&D Programme of China (2017YFB1400700).

REFERENCES

1. Zheng, Y.: Digital signcryption or how to achieve cost (signature & encryption)¡¡ cost (signature) + cost (encryption). In: Proceedings on Advances in Cryptology - CRYPTO '97, pp. 165–179. Santa Barbara (1997)
2. Gentry, C: Fully homomorphic encryption using ideal lattices. In: Proceedings on 41st Annual ACM Symposium on Theory of Computing - STOC'09, pp. 169–178. Bethesda (2009)
3. Dijk, Van., et al.: Fully homomorphic encryption over the integers. In: Proceedings on Advances in Cryptology - EUROCRYPT 2010, pp. 24–43. French Riviera (2010)
4. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM J. Comput. 43(2), 831–871 (2014)
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (levelled) Fully homomorphic encryption without bootstrapping. ACM Trans. Computation Theory (TOCT). 6(3), 13 (2014)
6. Boneh, D., Freeman, D.M.: Homomorphic signatures for polynomial functions. In: Proceedings on Advances in Cryptology - EUROCRYPT 2011, pp. 149–168. Tallinn, Estonia (2011)
7. Gorbunov, S., Vaikuntanathan, V., Wichs, D.: Levelled fully homomorphic signatures from standard lattices. In: Proc. Forty-Seventh Annual ACM Symposium on Theory of Computing - STOC'15, pp. 469–477. Portland, Oregon (2015)
8. Catalano, D., Fiore, D., Warinschi, B.: Homomorphic signatures with efficient verification for polynomial functions. In: Proceedings on Advances in Cryptology - CRYPTO 2014, pp. 371–389. Santa Barbara (2014)
9. Boyen, X., Fan, X., Shi, E.: Adaptively Secure Fully Homomorphic Signatures Based on Lattices, pp. 916. IACR Cryptology ePrint Archive, eprint (2014)
10. Rezaeibagha, F., et al.: Provably secure homomorphic signcryption. In: Proc. 11th International Conference on Provable Security - ProvSec 2017, pp. 349–360. Xi'an, China (2017)
11. Apon, D., et al.: Implementing Cryptographic Programme Obfuscation, pp. 779. IACR Cryptology ePrint Archive, eprint (2014)
12. Ananth, P., et al.: 'Optimising obfuscation: avoiding barrington's theorem'. In: Proceedings on 2014 ACM SIGSAC Conference on

Computer and Communications Security - CCS'14, pp. 646–658. Scottsdale (2014)
13. Lewi, K., et al.: 5gen: a framework for prototyping applications using multilinear maps and matrix branching programs. In: Proceedings on 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16, pp. 981–992. Vienna (2016)
14. Halevi, S., et al.: Implementing bp-obfuscation using graph-induced encoding. In: Proceedings on 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS'17, pp. 783–798. Dallas (2017)
15. Nayak, K., et al.: Hop: hardware makes obfuscation practical. In: Proceedings on 24th Annual Network and Distributed System Security Symposium - NDSS 2017. San Diego (2017)
16. Cousins, D.B., et al.: Implementing conjunction obfuscation under entropic ring lwe. In: Proceedings on 2018 IEEE Symposium on Security and Privacy, pp. 354–371. San Francisco (2018)
17. Boneh, D., et al.: Semantically secure order-revealing encryption: multi-input functional encryption without obfuscation. In: Proceedings on Advances in Cryptology - EUROCRYPT 2015, pp. 563–594. Sofia (2015)
18. Rezaeibagha, F., et al.: 'Provably secure (broadcast) homomorphic signcryption', Int. J. Found. Comput. Sci. 30(4), 511–529.(2019). https://doi.org/10.1142/S0129054119400100
19. Catalano, D., Marcedone, A., Puglisi, O.: Authenticating computation on groups: new homomorphic primitives and applications. In: Proceedings on Advances in Cryptology - ASIACRYPT 2014, pp. 193–212. Kaoshiung (2014)
20. Struck, P., et al.: Linearly homomorphic authenticated encryption with provable correctness and public verifiability, . In: Proceedings on Second International Conference on Codes, Cryptology and Information Security - C2SI 2017, pp. 142–160 Rabat, Morocco (2017)
21. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Proceedings on Advances in Cryptology - CRYPTO 2010, pp. 465–482. Santa Barbara, CA, USA (2010)
22. Chung, K.M., Kalai, Y.T., Vadhan, S.P.: Improved delegation of computation using fully homomorphic encryption. In: Proceedings on Advances in Cryptology - CRYPTO 2010, pp. 483–501. Santa Barbara (2010)
23. Tang, C., Chen, Y.: Efficient Non-Interactive Verifiable Outsourced Computation for Arbitrary Functions, pp. 439. IACR Cryptology ePrint Archive, 2014, eprint (2014)
24. Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: Proceedings on 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS'14, pp. 844–855.Scottsdale, Arizona, USA (2014)
25. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: Proceedings on 2013 ACM SIGSAC Conference on Computer and Communications Security - CCS'13, pp. 863–874. Berlin (2013)
26. Parno, B., Raykova, M., Vaikuntanathan, V.: How to delegate and verify in public: verifiable computation from attribute-based encryption. In: Proceedings on 9th Theory of Cryptography Conference - TCC 2012, pp. 422–439.Taormina, Sicily, Italy (2012)
27. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Proceedings on 17th International Conference on Practice and Theory in Public-Key Cryptography Public-Key Cryptography - PKC 2014, pp. 501–519. Buenos Aires (2014)
28. Boneh, D., Sahai, A., Waters, B.: Functional encryption: definitions and challenges. In: Proceedings on 8th Theory of Cryptography onference - TCC 2011, pp. 253–273. Providence (2011)
29. O'Neill, A.: Definitional Issues in Functional Encryption, pp. 556. IACR Cryptology ePrint Archive, eprint (2010)
30. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Functional encryption with bounded collusions via multi-party computation. In: Proceedings on Advances in Cryptology - CRYPTO 2012, pp. 162–179. Santa Barbara (2012)
31. Waters, B.: A punctured programing approach to adaptively secure functional encryption. In: Proceedings on Advances in Cryptology - CRYPTO 2015, pp. 678–697. Santa Barbara (2015)
32. Ahn, J.H., et al.: Computing on authenticated data. J. Cryptol. 28(2), 351–395 (2015)
33. Barak, B., et al.: 'On the (im) possibility of obfuscating programs'. J. ACM. 59(2), 6:1–6:48 (2012)
34. Garg, S., et al.: Candidate indistinguishability obfuscation and functional encryption for alright circuits. SIAM J. Comput. 45(3), 882–929 (2016)

How to cite this article: Li S, Liang B, Mitrokotsa A, Xue R. Homomorphic signcryption with public plaintext-result checkability. *IET Inf. Secur*. 2021;1–18. https://doi.org/10.1049/ise2.12026