Adaptively Secure Fully Homomorphic Signatures Based on Lattices

Xavier Boyen
Queensland University of Technology[*]

Xiong Fan   Elaine Shi
University of Maryland[†]

Abstract

In a homomorphic signature scheme, given the public key and a vector of signatures $\boldsymbol{\sigma} := (\sigma_1, \ldots, \sigma_l)$ over l messages $\boldsymbol{\mu} := (\mu_1, \ldots, \mu_l)$, there exists an efficient algorithm to produce a signature σ' for $\mu = f(\boldsymbol{\mu})$. Given the tuple (σ', μ, f), anyone can then publicly verify the validity of the signature σ'.

Inspired by the recent (selectively secure) key-homomorphic functional encryption for circuits, recent works propose fully homomorphic signature schemes in the selective security model. However, in order to gain adaptive security, one must rely on generic complexity leveraging, which is not only very inefficient but also leads to reductions that are "unfalsifiable".

In this paper, we construct the first *adaptively secure* homomorphic signature scheme that can evaluate any circuit over signed data. For *poly-logarithmic depth* circuits, our scheme achieves adaptive security under the standard *Small Integer Solution* (SIS) assumption. For *polynomial depth* circuits, the security of our scheme relies on sub-exponential SIS — but unlike complexity leveraging, the security loss in our reduction depends only on circuit depth and on neither message length nor dataset size.

1 Introduction

Motivated by the advances in cloud computing, there has been an increasing demand to outsource data and computations to servers in the cloud. Recently, a number of cryptographic schemes have been proposed to address the security and authenticity of computation outsourcing. The ground-breaking development of *fully homomorphic encryption* by Gentry [29] allows us to outsource computation on encrypted data. Two recent works [32, 39] propose *fully homomorphic signatures* that allow us to ascertain the authenticity of computation results on outsourced data. Under the hood, these homomorphic signatures are rooted, not so much in Gentry's homomorphic encryption, but from *key-homomorphic functional encryption for circuits* proposed by Boneh et al. at the recent Eurocrypt [17]; specifically, they can be seen as the signature algorithm naturally contained in the private key generation of the functional encryption.

However, both recently proposed fully homomorphic signature schemes [32, 39], such as the key-homomorphic functional encryption scheme from which they borrow, are only secure in the selective security model, where the adversary must announce ahead of time the message whose signature it intends to forge. Although selective security can be converted into adaptive security using the generic complexity leveraging technique [14], this approach is highly inefficient in this application: with complexity leveraging, the security reduction would incur a loss that is *exponential* in *both* the length of each message and the number of messages allowable in each dataset. Although the loss can be compensated by an increase in

[*] Email: xb@cs.stanford.edu. Research funded under ARC Discovery Project grant number DP-140103885 with the generous support of the Australian Research Council.

[†] Email: {xfan,elaine}@cs.umd.edu. Research funded in part by an NSF grant CNS-1314857, a Sloan Research Fellowship, and Google Faculty Research Awards. The views and conclusions contained herein are those of the authors and should not be interpreted as representing funding agencies.

the security parameters, the resulting security reduction becomes "unfalsifiable" in the sense that even with a perfect probability-one attack against the scheme, the security reduction will *fail* to solve the underlying hard problem in sub-exponential time (it will merely decrease its complexity from a large exponential down to a smaller exponential function of the security parameter [14]).

In this work, we are interested in building an adaptively secure, fully homomorphic signature scheme. Before presenting our results, we first briefly review the definition of homomorphic signatures. Informally, the problem of homomorphic signatures is as follows. Alice has a number of datasets, where each dataset has an identifying tag, and consists of l data entries $\mu_1, \ldots, \mu_l$. For example, a dataset stores the electricity consumption of l appliances in a household, and the tag of the dataset identifies the household. For each dataset, Alice computes l "root" signatures $\sigma_1, \ldots, \sigma_l$, and outsource both the dataset and the signatures to a cloud provider. Later, Alice specifies a dataset with its tag, and asks the server to compute some function f on the specified dataset. At this point, the server can not only compute the function f, but also produce a derived signature σ' for the computation result, such that Alice can verify that the result is correctly computed by evaluating the function f on the specified dataset.

By contrast to group-homomorphic signatures, which only provide homomorphism over a group operation such as $+$ or $\times$ but not both, in a fully homomorphic signature, the homomorphic operation(s)—such as $(+, \times)$ in $\mathbb{F}_p$, or NAND over $\{0,1\}^*$—are complete for the evaluation of arbitrary boolean functions. Homomorphic signatures can even be "multi-hop"—as is the case of our construction—, by allowing further homomorphic evaluations on derived signatures.

1.1 Results and Contributions

We show the following main results:

- For circuits *poly-logarithmic in depth*, we show how to construct an adaptively secure fully homomorphic signature scheme from standard assumptions, specifically, the Shortest Integer Solutions (SIS) assumption.

- For circuits that are polynomial in depth, we can also achieve adaptive security, but from *sub-exponential* SIS. Even though sub-exponential assumption is necessary, we note that the security loss of our reduction is $n^{O(d_{max})}$, i.e., depends only on the depth of the circuit d_{max}, but not on the size of the dataset. In comparison, with standard complexity leveraging techniques, the security loss is exponential in the size of each dataset.

Technical highlight. Our construction builds on the recent result by Boneh et al. [17] on key-homomorphic function encryption for circuits. Two very recent papers [39, 32] concurrently showed how to construct a (selectively secure) fully homomorphic based on the ideas in [17]. Our construction build upon the same ideas in order to achieve fully homomorphism.

In order to achieve full security against adaptively chosen-message queries, one idea is to rely on the "lattice mixing" trick by Boyen [18], who showed how to transform the selectively secure (H)IBE of Agrawal et al. [2] into a fully secure signature using a lattice-based partitioning technique. Indeed, the arithmetic structure of the key-homomorphic encryption scheme of [17] is eerily reminiscent of that of [2], which suggests that Boyen's trick could apply here as well.

To this end, we modify the fully homomorphic signature scheme by [32] to incorporate a mixing of public-key lattices in a message-dependent manner, in order to divide the message space into "known" and "unknown" parts in suitable proportion. In the "unknown" part, the simulator is unable to sign, but he can exploit a forgery in order to solve a hard problem, such as SIS. The technical contribution in [18] was to embed trapdoors in message-dependent lattices in a certain way, so that the trapdoor cancels unpredictably but with non-negligible probability.

However, simply using the lattice mixing trick is not sufficient, because the possibly large number of messages allowed in a dataset makes the partitioning technique nearly impossible to succeed with sufficient probability without additional help. We provide this help in the form of additional degrees of freedom that come into play precisely when the “trapdoors vanish”. Those additional degrees of freedom can be used to program the simulator so that it is still able to generate signatures on all possible messages even when the partitioning trapdoor has vanished, but only *one* such signature for each possible message.

Since the signatures are exponentially many and the adversary does not know which one the simulator knows in that case, any forgery will leave us with two distinct signatures, from which a solution to the SIS problem is easily derived.

Technical results. Performance-wise, our construction is very efficient and features polynomial reduction efficiency for circuits of polynomially many inputs and polylogarithmic depth, *in the adaptive security model*.

Compared with previous workds, our construction inherits the same limitation as [17] and its successors [39, 32] regarding the exponential loss of reduction efficiency with the *depth* of the circuits being homomorphically evaluated. For circuits of polynomial depth, it thus relies on sub-exponential SIS assumption [17]. Our construction however provides a direct and efficient (and falsifiable) adaptive security reduction. In particular, it eschews the need to rely on complexity leveraging, which in the case of the aforementioned papers would result in a reduction “looseness”—the degree by which an attack against the scheme can be exploited as an attack against the assumption—that is *exponential* in both the individual message length and the allowable dataset size.

1.2 Related Work

Homomorphic authenticators such as signatures and message authentication codes have for several years been a very active area of research. We briefly recapitulate some of the salient results in this search.

Restricted Homomorphic Signatures. Homomorphic signatures were studied by Johnson et al. [34] who proposed redactable signatures and set-homomorphic signatures. Redactable signatures were also studied later in various works [20, 8]. These schemes have the property that given a signature on a message, anyone can derive signatures on subsets of the message.

Several studied signatures schemes homomorphic with respect to linear functions [16, 7, 23, 25]. Such signature schemes have interesting applications in network coding [4] and proofs of retrievability. Boneh and Freeman [15] were the first to propose a scheme that can compute constant-degree polynomials on signed messages. Ahn et al. [5] propose a homomorphic signature scheme for a class of predicates such as quoting, subsets, weighted sum, etc.

(Leveled) Homomorphic Signatures from Lattices. We already mentioned the two recent and concurrent works [39, 32] which address the problem of evaluating arbitrary circuits over signed data. Both schemes are inspired from the key-homomorphic functional encryption scheme for circuits of [17]: indeed key generation in functional encryption can generically be used as a signature that will inherit the security of the original scheme. Unfortunately, in this case security only holds in the selective security model (assuming the hardness of *small integer solution* problem).

SNARKs. Another approach to address the homomorphic signature problem is to exploit CS-Proofs [35], or more generally, succinct non-interactive arguments of knowledge (SNARKs)[11, 12, 13, 27, 38, 10]. SNARKs allow a prove to produce a succinct proof for an NP statement, such that the verifier can verify the proof in succinct time. Unfortunately, due to a well-known lower bound by Gentry and Wichs [31], SNARKs – even without a "knowledge" requirement – cannot be constructed from standard assumptions. In this paper, we are interested in constructing fully homomorphic signature schemes from standard assumptions.

Verifiable Computation. Verifiable computation schemes [26, 38] delegate the computation of a function f over some input x to a server. Furthermore, the server can produce a succinct proof such that the client can verify the correctness of the computation result without performing the computation itself. In verifiable computation, the verifier knows the input x, but does not wish to perform the computation. By contrast, in our setting, the verifier may not know x – in fact x may be a huge dataset and stored on the server.

Symmetric-Key Homomorphic MACs. There has also been activity in constructing homomorphic message authentication (MACs) for various classes of homomorphisms. Unlike signatures, MACs allow only private verification, based on a symmetric key. The work initiated by Agrawal and Boneh [1] focuses on network coding applications. The recent paper by Gennaro and Wichs [28] defines and achieves fully homomorphic MACs without verification queries using fully homomorphic encryption. More recent works [21, 9, 22] show how to get homomorphic MACs that remain secure in the presence of verification queries, but only for restrictively homomorphic functions.

2 Preliminaries

Notations. Let PPT denote probabilistic polynomial time. We use bold uppercase letters to denote matrices, and bold lowercase letters to denote vectors. We let λ be the security parameter, $[n]$ denote the set $\{1, ..., n\}$, and $|\boldsymbol{t}|$ denote the number of bits in a string or vector $\boldsymbol{t}$. We denote the i-th bit value of a string $\boldsymbol{s}$ by $\boldsymbol{s}[i]$. We use $[\cdot|\cdot]$ to denote the concatenation of vectors or matrices, and $||\cdot||$ to denote the norm of vectors or matrices respectively.

2.1 Homomorphic Signatures

We adapt the notions [15, 32] to describe the syntax and security definition of homomorphic signatures. We denote the message space by $\mathcal{M}$, and let $C : \mathcal{M}^l \to \mathcal{M}$ denote a circuit that takes l messages and output a result message in $\mathcal{M}$. A homomorphic signature scheme for the circuit family $\mathcal{C}$ is a tuple of polynomial time algorithms $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Verify})$ specified as follows:

$\mathsf{Setup}(1^\lambda, 1^l)$: On input the security parameter λ and the maximum size l of a dataset whose messages can be signed, it outputs a public key pk and a secret key sk.

$\mathsf{Sign}(\mathsf{sk}, \boldsymbol{\tau}, i, \mu)$: On input a secret key sk, a tag $\boldsymbol{\tau} \in \{0,1\}^\lambda$, a message $\mu \in \{0,1\}$ and its corresponding index $i \in [l]$, it outputs a signature σ.

$\mathsf{Eval}(\mathsf{pk}, \boldsymbol{\tau}, \boldsymbol{\mu} = (\mu_1, ..., \mu_l), \boldsymbol{\sigma} = (\sigma_1, ..., \sigma_l), C)$: On input a public key pk, a tag $\boldsymbol{\tau} \in \{0,1\}^\lambda$, a sequence of messages $\boldsymbol{\mu}$ and a corresponding sequence of signatures $\boldsymbol{\sigma}$, and a circuit $C \in \mathcal{C}$, it outputs a derived signature σ' which corresponds to the evaluated message $C(\boldsymbol{\mu})$.

$\mathsf{Verify}(\mathsf{pk}, \boldsymbol{\tau}, \mu, \sigma, C)$: On input a public key pk, a tag $\boldsymbol{\tau} \in \{0,1\}^\lambda$, a message μ, its signature σ, and a circuit $C \in \mathcal{C}$, it outputs 0 (reject) or 1 (accept).

The tags serve to distinguish between different datasets, with the intent being that only signatures with matching tags be combinable homomorphically. From the user's viewpoint, the tag is a bit-string of length λ selected uniformly at random.

Correctness. We say that the $\mathcal{C}$-homomorphic signature $\mathcal{HS}$ is correct, if for any tag $\tau \in \{0,1\}^\lambda$, any circuit $C \in \mathcal{C}$, any set of messages $\boldsymbol{\mu} \in \mathcal{M}^l$, and any index $i \in [l]$, we have

$$\mathbf{Pr}[\mathsf{Verify}(\mathsf{pk}, \tau, \mu', \sigma', C) = 1] = 1$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^l)$, $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, \boldsymbol{\tau}, i, \mu_i)$, for $i \in [l]$, $\sigma' \leftarrow \mathsf{Eval}(\mathsf{pk}, \boldsymbol{\tau}, \boldsymbol{\mu}, \boldsymbol{\sigma}, C)$ and $\mu' = C(\boldsymbol{\mu})$. We remark the circuit C can also be a projection circuit P_i, i.e., $P_i(\mu_1, ..., \mu_l) = \mu_i$, which means that the correctness also must hold for single-message signatures.

2.2 Security Definitions

Adaptive security for homomorphic signatures can be defined in different ways, depending on whether the adversary is only allowed to make adaptive queries one full message dataset at a time, or can make adaptive queries on individual messages within a given dataset, possibly even interleaving those queries across several datasets. We propose the following three security models.

Fully adaptive queries at the message level. The strongest model allows the adversary to make signature queries on messages specified adaptively in any order, regardless of the dataset to which they belong.

Let $\mathcal{A}$ denotes any PPT adversary. Let $\mathcal{HS}$ be a homomorphic signature scheme for circuit family $\mathcal{C}$. We define the notion of *Existential Unforgeability of Homomorphic Signatures under Full Chosen-Message Attacks* using the following experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{EUF\text{-}FH\text{-}CMA}}(1^\lambda)$.

- The challenger runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^l)$ and sends pk to the adversary $\mathcal{A}$.
- Proceeding adaptively, the adversary $\mathcal{A}$ specifies a sequence of signature queries. Each query consists of:
  - a dataset index $i \in 2^\lambda$;
  - a message index $j \in [l]$, such that the pair $(\boldsymbol{\tau}_i, j)$ has not been queried before;
  - A message $\mu_{ij} \in \mathcal{M}$.
- The challenger assigns a random tag $\boldsymbol{\tau}_i \in \{0,1\}^\lambda$ to the dataset i, if one had not previously been assigned. (The challenger either maintains a private list of pairs $(i, \boldsymbol{\tau}_i)$ or statelessly maps i to $\boldsymbol{\tau}_i$ using a private PRF under a random indistinguishability PRF assumption.
- The challenger sends back:
  - the dataset tag $\boldsymbol{\tau}_i$;
  - the signature $\sigma_{ij} \leftarrow \mathsf{Sign}(\mathsf{sk}, \boldsymbol{\tau}_i, j, \mu_{ij})$;
- The adversary $\mathcal{A}$ outputs a tuple $(\tau^*, \mu^*, \sigma^*, C^*)$.

The adversary wins if $\mathsf{Verify}(\mathsf{pk}, \tau^*, \mu^*, \sigma^*, C^*) = 1$, and either

1. (Type-A Forgery) $\tau^* \neq \tau_i$ for all i, or
2. (Type-B Forgery) $\tau^* = \tau_i$ for some i, but $\mu^* \neq C^*(\boldsymbol{\mu}_i)$, where $\boldsymbol{\mu}_i = (\mu_{i1}, ..., \mu_{il})$ is the vector of messages queried under a common tag τ_i but differing indices $j \in [l]$.

Definition 2.1 (Existential unforgeability against fully adaptive chosen-message attacks)**.** *We say a homomorphic signature scheme $\mathcal{HS}$ is fully unforgeable against adaptive message queries with respect to a circuit family $\mathcal{C}$ if no* PPT *adversary $\mathcal{A}$ can win the experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{EUF\text{-}FH\text{-}CMA}}(1^\lambda)$ with non-negligible probability.*

Sequentially adaptive queries at the message level. We can relax the above model by requiring the adversary to make all its message queries within a dataset before moving to the next dataset. Messages are still chosen adaptively within a dataset, and so are the datasets themselves. We omit the formal definition since we do not use this model in our construction.

Adaptive queries at the dataset level. The weaker form of adaptive security we consider requires the adversary to query all the messages in a dataset at once. Queries now consist of (ordered) message vectors $\boldsymbol{\mu}_i = \{\mu_{i1}, ...\mu_{il}\}$ representing a dataset. The choice of datasets themselves remains adaptive from one query to the next.

Formally, we define the notion of *Existential Unforgeability of Homomorphic Signatures under Chosen-Dataset Attacks* using the following experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{EUF\text{-}FH\text{-}CDA}}(1^\lambda)$:

- The challenger runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^l)$ and sends pk to the adversary $\mathcal{A}$.
- Proceeding adaptively, the adversary $\mathcal{A}$ specifies a sequence of signature queries. Each query consists of:
  - a dataset given as an l-message vector $\boldsymbol{\mu}_i = \{\mu_{i1}, ...\mu_{il}\}$;
- The challenger sends back:
  - a randomly chosen dataset tag $\boldsymbol{\tau}_i$;
  - a signature vector $\boldsymbol{\sigma}_i = \{\sigma_{ij}\}_{j \in [l]}$ where $\sigma_{ij} \leftarrow \mathsf{Sign}(\mathsf{sk}, \boldsymbol{\tau}_i, j, \mu_{ij})$.

Definition 2.2 (Existential unforgeability against adaptive chosen-dataset attack)**.** *We say a homomorphic signature scheme $\mathcal{HS}$ is fully unforgeable against adaptive dataset queries with respect to a circuit family $\mathcal{C}$ if no* PPT *adversary $\mathcal{A}$ can win the experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{EUF\text{-}FH\text{-}CDA}}(1^\lambda)$ with non-negligible probability.*

For simplicity of presentation, we shall first focus on constructing and proving the relaxed EUF-FH-CDA notion of adaptive security. In the Extensions section we show the modifications to apply in order to construct and prove the strict EUF-FH-CMA notion of adaptive security.

Toward fully secure key-homomorphic functional encryption. One common limitation of the above models is that they require random tags $\boldsymbol{\tau}_i$, rather than tags chosen by the adversary, or tags that would be otherwise predictable. For signature purposes, this is a benign limitation, as the signer can safely choose the tag in an unpredictable way without impairing functionality. However, the random-tag restriction is the critical technical limitation that prevents us from applying the techniques in this paper, to the (selectively secure) key-homomorphic functional encryption of Boneh et al.[17], in order to make it adaptively secure. The problem of constructing an adaptively secure key-homomorphic functional encryption scheme for circuits remains open.

2.3 Lattices

A full-rank m-dimensional integer lattice $\Lambda \subset \mathbb{Z}^m$ is a discrete additive subgroup whose linear span is $\mathbb{R}^m$. The basis of Λ is a linearly independent set of vectors whose linear combinations are exactly Λ. Every integer lattice is generated as the $\mathbb{Z}$-linear combination of linearly independent vectors $\mathbf{B} = \{\boldsymbol{b}_1, ..., \boldsymbol{b}_m\} \subset \mathbf{Z}^m$. For a matrix $\mathbf{A} \in \mathbf{Z}_q^{n \times m}$, we define the "$q$-ary" integer lattices:

$$\Lambda_q^{\perp} = \{\boldsymbol{e} \in \mathbf{Z}^m | \mathbf{A}\boldsymbol{e} = 0 \mod q\},\ \Lambda_q^{\mathbf{u}} = \{\boldsymbol{e} \in \mathbf{Z}^m | \mathbf{A}\boldsymbol{e} = \boldsymbol{u} \mod q\}$$

It is obvious that $\Lambda_q^{\boldsymbol{u}}$ is a coset of $\Lambda_q^{\perp}$.

Let Λ be a discrete subset of $\mathbf{Z}^m$. For any vector $\boldsymbol{c} \in \mathbb{R}^m$, and any positive parameter $\sigma \in \mathbb{R}$, let $\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x}) = \exp(-\pi||\boldsymbol{x} - \boldsymbol{c}||^2/\sigma^2)$ be the Gaussian function on $\mathbb{R}^m$ with center $\boldsymbol{c}$ and parameter σ. Next, we set $\rho_{\sigma,\boldsymbol{c}}(\Lambda) = \sum_{\boldsymbol{x}\in\Lambda} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x})$ be the discrete integral of $\rho_{\sigma,\boldsymbol{x}}$ over Λ and $D_{\Lambda,\sigma,\boldsymbol{c}}(\boldsymbol{y}) := \frac{\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y})}{\rho_{\sigma,\boldsymbol{c}}(\Lambda)}$. We abbreviate σ in the $\rho_{\sigma,\boldsymbol{c}}$ and $D_{\sigma,\boldsymbol{c}}$ when $\sigma = 0$.

Let S^m denote the set of vectors in $\mathbb{R}^{m+1}$ whose length is 1. Then the norm of a matrix $\mathbf{R} \in \mathbb{R}^{m\times m}$ is defined to be $\mathsf{sup}_{\boldsymbol{x}\in S^m}||\mathbf{R}\boldsymbol{x}||$. Then we have the following lemma, which bounds the norm for some specified distributions.

Lemma 2.1 ([2]). *Regarding the norm defined above, we have the following bounds:*

- *Let* $\mathbf{R} \in \{-1, 1\}^{m\times m}$ *be chosen at random, then we have* $\mathbf{Pr}[||\mathbf{R}|| > 12\sqrt{2m}] < e^{-2m}$.
- *Let* $\mathbf{R}$ *be sampled from* $D_{\mathbb{Z}^{m\times m},\sigma}$, *then we have* $\mathbf{Pr}[||\mathbf{R}|| > \sigma\sqrt{m}] < e^{-2m}$.

Below, we recall some classical technique and result that can reduce the norm of matrix, but still preserves the result of multiplications. For simplicity, we define the algorithms regarding vectors as input, but they can also be applied to matrix by taking the column vectors of matrix as input.

- $\mathsf{BitDecomp}(\boldsymbol{r} \in \mathbb{Z}^m, q)$ decomposes $\boldsymbol{r}$ into its bit representation, namely $\boldsymbol{r} = \sum_{i=0}^{\lfloor\log q\rfloor} 2^i \cdot \boldsymbol{u}_i$, where $\boldsymbol{u}_i \in \{0, 1\}^m$, and outputs $(\boldsymbol{u}_0, ..., \boldsymbol{u}_{\lfloor\log q\rfloor})$.
- $\mathsf{Powersof2}(\boldsymbol{r} \in \mathbb{Z}^m, q)$ outputs the vector $(\boldsymbol{r}, 2 \cdot \boldsymbol{r}, ..., 2^{\lfloor\log q\rfloor} \cdot \boldsymbol{r})$.

Lemma 2.2 ([19]). *For vectors* $\boldsymbol{c}, \boldsymbol{s}$ *of equal length, we have*

$$\langle\mathsf{BitDecomp}(\boldsymbol{c}, q), \mathsf{Powersof2}(\boldsymbol{s}, q)\rangle = \langle\boldsymbol{c}, \boldsymbol{s}\rangle \pmod q$$

where $\langle\cdot, \cdot\rangle$ *is the inner product of vectors.*

Randomness extraction. We will use the following lemma to argue the indistinghishability of two different distributions, which is a generalization of the leftover hash lemma proposed by Dodis et al. [24].

Lemma 2.3 ([3]). *Suppose that* $m > (n + 1)\log q + w(\log n)$. *Let* $\mathbf{R} \in \{-1, 1\}^{m\times k}$ *be chosen uniformly at random for some polynomial* $k = k(n)$. *Let* $\mathbf{A}, \mathbf{B}$ *be matrix chosen randomly from* $\mathbb{Z}_q^{n\times m}, \mathbb{Z}_q^{n\times k}$ *respectively. Then, for all vectors* $\boldsymbol{w} \in \mathbb{Z}^m$, *the two following distributions are statistically close:*

$$(\mathbf{A}, \mathbf{AR}, \mathbf{R}^T\boldsymbol{w}) \approx (\mathbf{A}, \mathbf{B}, \mathbf{R}^T\boldsymbol{w})$$

Small Integer Solution. The SIS problem was first suggested to be hard on average by Ajtai [6] and then formalized by Micciancio and Regev [37].

Definition 2.3 (SIS). *For any* $n \in \mathbb{Z}$, *and any functions* $m = m(n), q = q(n), \beta = \beta(n)$, *the average-case Small Integer Solution problem* ($\mathsf{SIS}_{q,n,m,\beta}$) *is: Given an integer q, a matrix* $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ *chosen uniformly at random and a real* $\beta \in \mathbb{R}$, *find a non-zero integer vector* $\boldsymbol{z} \in \mathbb{Z}^m \setminus \{\mathbf{0}\}$, *such that* $\mathbf{A}\boldsymbol{z} = 0 \mod q$ *and* $||\boldsymbol{z}|| \leq \beta$.

Micciancio and Regev [37] showed that solving the average-case $\mathsf{SIS}_{q,n,m,\beta}$ problem for certain parameters is as hard as approximating the Shortest Independent Vector Problem in the worst case to within certain $\gamma = \beta \cdot \tilde{O}(\sqrt{n})$ factors.

Trapdoors and sampling algorithms. We will use the following algorithms to sample short vectors from specified lattices.

Lemma 2.4 ([30])**.** *Let q, n, m be positive integers with $q \geq 2$ and $m \geq 6n \log q$. There exists a* PPT *algorithm* $\mathsf{TrapGen}(q, n, m)$ *that with overwhelming probability outputs a pair* $(\mathbf{A} \in \mathbb{Z}_q^{n \times m}, \mathbf{T_A} \in \mathbb{Z}^{m \times m})$ *such that* $\mathbf{A}$ *is statistically close to uniform in* $\mathbb{Z}_q^{n \times m}$ *and* $\mathbf{T_A}$ *is a basis for* $\Lambda_q^{\perp}(\mathbf{A})$ *satisfying* $||\mathbf{T_A}|| \leq O(n \log q)$.

Lemma 2.5 ([30, 36])**.** *There is an efficient algorithm* $\mathsf{SampleD}$ *that takes as input a matrix* $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ *together with its trapdoor* $\mathbf{T_A}$*, and a matrix* $\mathbf{U} \in \mathbb{Z}_q^{n \times m}$ *and outputs a matrix* $\mathbf{R} \in \mathbb{Z}^{m \times m}$ *such that:*

$$\mathbf{AR} = \mathbf{U} \mod q, \textit{and } ||\mathbf{R}|| \leq ||T|| \times O(m)$$

Lemma 2.6 ([2])**.** *Let $q > 2, m > n$ and $s > ||\mathbf{T_A}|| \cdot w(\sqrt{\log m + m_1})$. There exists a* PPT *algorithm* $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \boldsymbol{u}, s)$*, where given* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$*, a short basis* $\mathbf{T_A}$ *for lattice* $\Lambda_q^{\perp}(\mathbf{A})$*, a matrix* $\mathbf{B} \in \mathbb{Z}_q^{n \times m_1}$*, a vector* $\boldsymbol{u} \in \mathbb{Z}_q^n$ *and the Gaussian parameter s, outputs a vector* $\mathbf{e} \in \mathbb{Z}_q^{m+m_1}$ *such that* $\boldsymbol{e} \in \Lambda_q^{\boldsymbol{u}}(\mathbf{F})$ *where* $\mathbf{F} := (\mathbf{A}|\mathbf{B})$*, and is statistical close to* $D_{\Lambda_q^{\boldsymbol{u}}(\mathbf{F}),s}$.

Lemma 2.7 ([2])**.** *Let $q > 2, m > n$ and $s > ||\mathbf{T_B}|| \cdot w(\sqrt{\log m}) s_{\mathbf{R}}$, where $s_{\mathbf{R}} = ||\mathbf{R}||$. There exists a* PPT *algorithm* $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \boldsymbol{u}, s)$*, taking a matrix* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$*, a short matrix* $\mathbf{R} \in \mathbb{Z}_q^{m \times n}$*, a matrix* $\mathbf{B} \in \mathbb{Z}_q^{n \times n}$*, a short basis* $\mathbf{T_B}$ *for lattice* $\Lambda_q^{\perp}(\mathbf{B})$*, a vector* $\boldsymbol{u} \in \mathbb{Z}_q^n$ *and the Gaussian parameter s, outputs a vector* $\boldsymbol{e} \in \mathbb{Z}_q^{m+n}$ *such that* $\boldsymbol{e} \in \Lambda_q^{\mathbf{u}}(\mathbf{F})$ *where* $\mathbf{F} := (\mathbf{A}|\mathbf{AR} + \mathbf{B})$*, and is statistical close to* $D_{\Lambda_q^{\boldsymbol{u}}(\mathbf{F}),s}$.

3 Construction

Recall that to each dataset must be associated a unique random tag $\boldsymbol{\tau}$, used for signing and verification. In our construction, the tag $\boldsymbol{\tau}$ shall consist of two parts: $\boldsymbol{t} \in \{0,1\}^{\lambda}$ and $\boldsymbol{b} \in \{0,1\}^{l}$, where for convenience we fix the first bit $\boldsymbol{t}[0] = 0$.

The first component $\boldsymbol{t}$ is used with the ”lattice mixing” technique of [18] to provide adaptive security over the choice of datasets. The second component $\boldsymbol{b}$ is needed to prove adaptive security. More specifically, we leverage vector $\boldsymbol{b}$ to provide enough degrees of freedom such that the reduction can pre-sample signatures for all possible adaptive chosen messages. (We refer to these latter forgeries as Type III forgeries in the proof of security; they are a special case of Type I forgeries as defined above.)

Our $\mathcal{C}$-homomorphic signature schemes $\mathcal{HS}$ can be described as follows:

- $\mathsf{Setup}(1^{\lambda}, 1^{l}, 1^{|\boldsymbol{\tau}|}, 1^{d_{max}})$: The setup algorithm takes in the security parameter λ, the maximum number l of inputs for the circuit family $\mathcal{C}$, the number of bits $|\boldsymbol{\tau}|$ for the tag (where $|\boldsymbol{\tau}| = |\boldsymbol{t}| + |\boldsymbol{b}|$ normally $= \lambda + l$), and the maximum depth d_{max} of the circuit family $\mathcal{C}$.

  1. Set the lattices parameters $n = n(\lambda, d_{max})$, $q = q(n, d_{max})$, $m = m(n, d_{max})$. Let the Gaussian parameters be $s_1 = s_1(n)$ and $s_2 = s_2(n)$.
  2. Sample $2l + 1$ random matrices $\mathbf{A}$ and $\{\mathbf{D}_{i,b}\}_{i \in [l], b \in \{0,1\}}$ in $\mathbb{Z}_q^{n \times m}$, and $|\boldsymbol{t}|$ random matrices $\{\mathbf{T}_i\}_{i \in [|\boldsymbol{t}|]}$ from $\mathbb{Z}_q^{n \times m}$.
  3. Sample two matrices with their associated trapdoors:

     $$(\mathbf{A}^*, \mathbf{T}_{\mathbf{A}^*}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, 1^q), \qquad (\mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, 1^q)$$

  4. Output the public key $\mathsf{pk} = (\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \{\mathbf{D}_{i,b}\}_{i \in [l], b \in \{0,1\}}, \{\mathbf{T}_i\}_{i \in [|\boldsymbol{t}|]}, \mathbf{T_B})$ and the secret key $\mathsf{sk} = \mathbf{T}_{\mathbf{A}^*}$.

- $\mathsf{Sign}(\mathsf{sk}, \boldsymbol{\tau}, \mu, i)$: The signing algorithm takes in a secret key sk, a tag $\boldsymbol{\tau}$, a message m, and a message index i (respective to the dataset specified by the tag $\boldsymbol{\tau}$).

  1. Parse $\boldsymbol{\tau}$ as $\boldsymbol{\tau} = [\boldsymbol{t}|\boldsymbol{b}] \in \{0,1\}^\lambda \times \{0,1\}^l$. Ensure that the tag's first bit $\boldsymbol{\tau}[0] = 0$.
  2. Define the $n \times m$ matrix $\mathbf{T} := \sum_{i=0}^{|\boldsymbol{t}|} (-1)^{\boldsymbol{t}[i]} \mathbf{T}_i$.
  3. Let $\mathbf{A}_{\boldsymbol{t}} := [\mathbf{A}^*|\mathbf{A} + \mathbf{T}]$ denote the dataset matrix.
  4. Sample a matrix $[\mathbf{R}_2|\mathbf{R}_1] \in \mathbb{Z}^{2m \times m}$ using,

     $$[\mathbf{R}_2|\mathbf{R}_1] \leftarrow \mathsf{SampleLeft}(\mathbf{A}^*, \mathbf{A}_{\boldsymbol{t}}, \mathbf{T}_{\mathbf{A}^*}, \mathbf{D}_{i,\boldsymbol{b}[i]} + \mu\mathbf{B}, s_2)$$

     Therefore, it holds

     $$[\mathbf{A}^*|\mathbf{A} + \mathbf{T}] \begin{bmatrix} \mathbf{R}_2 \\ \mathbf{R}_1 \end{bmatrix} = \mathbf{D}_{i,\boldsymbol{b}[i]} + \mu\mathbf{B}$$

  5. Output the signature $\sigma_i = [\mathbf{R}_2|\mathbf{R}_1]$.

  Recall that matrices $\mathbf{D}_C$ serve to "bind" any circuit C to a signature. Here, the matrix $\mathbf{D}_{i,\boldsymbol{b}[i]}$ binds the signature to the (trivial) projection circuit that selects the i-th message μ_i from the dataset $\boldsymbol{\mu}$. Two observations are in order: (1) Sign is only input the message m_i and its index i, not the whole dataset $\boldsymbol{\mu}$ which need not be know at this stage; (2) the circuit-encoding matrix $\mathbf{D}_{i,\boldsymbol{b}[i]}$ is one of two possible choices, selected by the tag bit $\boldsymbol{b}[i]$ which is supplied as input to Sign and shall likewise be available for verification.

- $\mathsf{Verify}(\mathsf{pk}, \boldsymbol{\tau}, \mu, \sigma, C)$: The verification algorithm takes in a public key pk, a tag $\boldsymbol{\tau}$, a message/signature pair (μ, σ), and a circuit C. It outputs 1 only if the following two requirements are satisfied:

  1. Parse the tag $\boldsymbol{\tau} = (\boldsymbol{t}|\boldsymbol{b}) \in \{0,1\}^{\lambda + l}$ and signature $\sigma = [\mathbf{R}_1|\mathbf{R}_2] \in \mathbb{Z}^{2m \times m}$ and verify that $\boldsymbol{t}[0] = 0$. Verify that $||\mathbf{R}|| \leq B$.
  2. Let $\mathbf{A}_{\boldsymbol{t}} := [\mathbf{A}^*|\mathbf{A} + \mathbf{T}]$, where $\mathbf{T} := \sum_{i=0}^{|\boldsymbol{t}|} (-1)^{\boldsymbol{t}[i]} \mathbf{T}_i$, and $\mathbf{D}_C$ denote the public matrix associated with the circuit C as computed by the evaluation algorithm defined below. Check that

     $$\mathbf{A}_{\boldsymbol{t}}\mathbf{R} = \mathbf{D}_C + \mu\mathbf{B} \pmod q$$

  Here, generally μ will be the message resulting from the circuit evaluation $\mu = C(\boldsymbol{\mu})$, and σ the corresponding signature homomorphically evaluated from the dataset signature vector $\boldsymbol{\sigma} = \{\sigma_i\}_{i \in [l]}$.

- $\mathsf{Eval}(\mathsf{pk}, \boldsymbol{\tau}, \boldsymbol{\mu}, \boldsymbol{\sigma}, C)$: The evaluation algorithm takes in a public key pk, a tag $\boldsymbol{\tau} = (\boldsymbol{t}|\boldsymbol{b})$, a pair of message/signature vectors $(\boldsymbol{\mu}, \boldsymbol{\sigma})$, and a circuit C.

  1. Suppose the gate is $g = (u, v, w)$ is a NAND gate. For each wire in the gate, let $\mathbf{D}_i$ be the public matrix associated with that wire. Also construct the dataset matrix $\mathbf{A}_{\boldsymbol{t}} := [\mathbf{A}^*|\mathbf{A} + \mathbf{T}]$, where $\mathbf{T} := \sum_{i=0}^{|\boldsymbol{t}|} (-1)^{\boldsymbol{t}[i]} \mathbf{T}_i$.
  2. By induction on the ($\boldsymbol{b}$-tag-dependent) public matrix associated with each wire, we have $(\mathbf{R}_u, \mathbf{R}_v)$ such that:

     $$\mathbf{A}_{\boldsymbol{t}}\mathbf{R}_u = \mathbf{D}_u + x\mathbf{B},\ \mathbf{A}_{\boldsymbol{t}}\mathbf{R}_v = \mathbf{D}_v + y\mathbf{B}$$

     where (x, y) are the values carried by wires (u, v).
  3. Define the public matrix associated with output wire w as $\mathbf{D}_w := \mathbf{D}_v\widetilde{\mathbf{D}}_u - \mathbf{B}$, where $\widetilde{\mathbf{D}}_u \leftarrow \mathsf{SampleD}(\mathbf{B}, \mathbf{T}_\mathbf{B}, \mathbf{D}_u, s_1) \in \mathbb{Z}^{m \times m}$, so that $\mathbf{B}\widetilde{\mathbf{D}}_u = \mathbf{D}_u$.
  4. Output the homomorphic signature:

     $$\mathbf{R}_w := \mathbf{R}_v\widetilde{\mathbf{D}}_u - y\mathbf{R}_u$$

3.1 Correctness

We show that (with lattice parameters as specified later on in Section 4.3) our construction satisfies the correctness condition defined above, which means that the verification algorithm accepts an honestly computed homomorphic signature. We adapt the induction method presented in [32] to state our claim.

Lemma 3.1. *Let $\mathbf{A}_{\boldsymbol{t}}$ be the dataset matrix, $\mathbf{D}_C$ be the public key derived with respect to the circuit C, and $\sigma = \mathbf{R} \in \mathbb{Z}^{2m \times m}$ denote the homomorphically computed signature. Then, as we require in the verification algorithm:*

$$\mathbf{A}_{\boldsymbol{t}} \cdot \mathbf{R} = \mathbf{D}_C + C(\boldsymbol{\mu})\mathbf{B}$$

Proof. Without loss of generality, we consider NAND gates $g = g(u, v, w)$ with input wires u, v carrying input values x, y respectively, and with output wire w.

Base case. Let $\sigma = [\mathbf{R}_2|\mathbf{R}_1]$ be the signature for message m under tag $\tau = (\boldsymbol{t}|\boldsymbol{b})$, then we have, by construction of the algorithm Sign:

$$[\mathbf{A}^*|\mathbf{A} + \mathbf{T}] \begin{bmatrix} \mathbf{R}_2 \\ \mathbf{R}_1 \end{bmatrix} = \mathbf{D}_{i,\boldsymbol{b}[i]} + \mu\mathbf{B}$$

where $(\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \{\mathbf{D}_{i,b}\}, \{\mathbf{T}_i\}, \mathbf{T}_\mathbf{B})$ are generated in the Setup algorithm, and $\mathbf{T} := \sum_{i=0}^{|\boldsymbol{t}|}(-1)^{t[i]}\mathbf{T}_i$. By definition the matrix $\mathbf{D}_{i,\boldsymbol{b}[i]}$ encodes the projection circuit for the i-th input parameterized by tag $\boldsymbol{b}$.

Inductive step. Let matrix $\mathbf{R}_u, \mathbf{R}_v$ be signatures for messages x, y respectively, under public keys $\mathbf{D}_u, \mathbf{D}_v$, such that

$$\mathbf{A}_{\boldsymbol{t}}\mathbf{R}_u = \mathbf{D}_u + x\mathbf{B}, \qquad \mathbf{A}_{\boldsymbol{t}}\mathbf{R}_v = \mathbf{D}_v + y\mathbf{B}$$

Let

$$\mathbf{R}_w = \mathbf{R}_v\widetilde{\mathbf{D}}_u - y\mathbf{R}_u, \qquad \mathbf{D}_w := \mathbf{D}_v\widetilde{\mathbf{D}}_u - \mathbf{B}$$

then we must show that

$$\mathbf{A}_{\boldsymbol{t}}\mathbf{R}_w = \mathbf{D}_w + (x \;\mathsf{NAND}\; y)\mathbf{B}$$

Indeed, we have

$$\begin{aligned} \mathbf{A}_{\boldsymbol{t}} \cdot \mathbf{R}_w &= \mathbf{A}_{\boldsymbol{t}}(\mathbf{R}_v\widetilde{\mathbf{D}}_u - y\mathbf{R}_u) \\ &= (\mathbf{D}_v + y\mathbf{B})\widetilde{\mathbf{D}}_u - y(\mathbf{D}_u + x\mathbf{B}) \\ &= \mathbf{D}_v\widetilde{\mathbf{D}}_u - xy\mathbf{B} \\ &= \mathbf{D}_w + (1 - xy)\mathbf{B} \\ &= \mathbf{D}_w + (x \;\mathsf{NAND}\; y)\mathbf{B} \end{aligned}$$

This establishes the lemma, and thus correctness of the signature scheme. □

4 Security Proof

In this section, we prove the unforgeability of our scheme against *chosen dataset* attacks. Later in Section 5, we describe how to modify our construction to achieve the stronger notion of unforgeability against fully adaptive chosen message attacks.

4.1 Intuition

We now explain the intuition behind our proof. Assume there exists a PPT adversary $\mathcal{A}$ who wins the unforgeability security game defined above, we construct an reduction $\mathcal{B}$ which can leverage the adversary $\mathcal{A}$ to break the $\mathsf{SIS}_{q,n,m,\beta}$ assumption for lattice defined by the basis $\mathbf{A}^*$. The careful reader may have noted that the key difference between our construction and that of counterpart in [32] is the following: 1) the introduction of matrix $\mathbf{T}_i$'s in the public key; and 2) the introduction of the vector $\boldsymbol{b}$ as part of the tag. Both are essential in the adaptive security proof. First, the matrix $\mathbf{T}_i$ in the public key allow us to rely on a lattice mixing and vanishing trick [18, 2]. Second, the introduction of vector $\boldsymbol{b}$ tag creates enough degrees of freedom such that the reduction can "pre-sample" signatures in the simulation without knowing what message the adversary will query ahead of time.

As noted earlier in Definition 2.1, there are two types of forgeries, Type A and Type B forgeries. In the security proof, we will further divide the forgeries into three different types:

- Type-I forgery: adversary $\mathcal{A}$ submits a forgery tuple $(\boldsymbol{\tau}^* = (\boldsymbol{t}^*|\boldsymbol{b}^*), \mu^*, \sigma^*, C^*)$, where the tag $\boldsymbol{t}^*$ has not appeared in an answer to a previous query.
- Type-II forgery: adversary $\mathcal{A}$ submits a forgery $(\boldsymbol{\tau}^*, \mu^*, \sigma^*, C^*)$, where $\boldsymbol{\tau}^* = (\boldsymbol{t}^*|\boldsymbol{b}^*)$ comes from the answer to a previous query, but $\mu^* \neq C^*(\boldsymbol{\mu}^*)$ where $\boldsymbol{\mu}^*$ is the message corresponding to tag $\boldsymbol{\tau}^*$.
- Type-III forgery: adversary $\mathcal{A}$ submits a forgery $((\boldsymbol{t}^*|\boldsymbol{b}^*), m^*, \sigma^*, C^*)$, where $\boldsymbol{t}^*$ was queried before, but $\boldsymbol{b}^*$ was not ever used.

In particular, the union of Type I and Type III correspond to Type A in Definition 2.1. Type II corresponds to the Type B in Definition 2.1.

Type I forgery is handled using the standard lattice mixing trick [18] for proving adaptive security. Basically, the reduction chooses the public parameters $\mathbf{T}_i$'s as $\mathbf{A}^*\mathbf{S}_i + h_i\mathbf{B}$ For each signing query made by the adversary, the reduction will randomly select a $\boldsymbol{t}$ tag such that if the following good event happens: $\mathbf{A}_{\boldsymbol{t}} = \mathbf{A}^*\mathbf{U} + h\mathbf{B}$ for a non-zero scalar h, then the reduction will be able to use knowledge of the matrix $\mathbf{B}$'s trapdoor $\mathbf{T}_{\mathbf{B}}$ to answer the signing queries. Finally, when the adversary forges a signature for an unseen $\boldsymbol{t}$ tag, if the good event $\mathbf{A}_{\boldsymbol{t}} = \mathbf{A}^*\mathbf{U} + h\mathbf{B}$ for a non-zero scalar h happens – then the reduction can easily leverage the forgery to break SIS. The reduction aborts if a bad event happens during the simulation. Using a standard argument [18] we show that the the probability that the reduction completes the simulation without aborting is non-negligible.

For Type II forgery, the reduction will randomly guess the query c^* whose tag $\boldsymbol{t}^*$ the adversary will inherit in the forgery. For the c^*-th query, the reduction will embed a tag $\boldsymbol{t}^*$ such that $\mathbf{A}_{\boldsymbol{t}^*} = \mathbf{A}^*\mathbf{U}$ (or abort). In this case, the reduction cannot compute signatures using the honest algorithm since it does not have a trapdoor $\mathbf{T}_{\mathbf{A}^*}$ for matrix $\mathbf{A}^*$. Here, the reduction will crucially rely on the existence of the $\boldsymbol{b}$ tag to pre-select signatures for any possible message submitted in the c^*-th query. Namely, for each coordinate $(i, b), i \in [l], b \in \{0, 1\}$, the simulator pre-selects a random mapping $\pi_i : \{0, 1\} \to \{0, 1\}$, such that $\mathbf{D}_{i,\pi_i(b)}$ will be used if $m_i = b$ in the c^*-th query. During the setup, the challenger randomly generates the signatures first for the c^*-th query, and then computes the matrix $\mathbf{D}_{i,b}, i \in [l], b \in \{0, 1\}$ values based on the pre-selected signatures. In this case, if the adversary makes a forgery, we show that it can help the reduction break SIS.

For Type III forgery, the reduction performs the simulated setup in the same way as Type II forgery, by pre-selecting the signatures for the c^*-th query, where c^* is guessed by the reduction at the beginning of the simulation. In the end, the adversary will forge a signature for the tag $\boldsymbol{t}^*$, but with a different $\boldsymbol{b}$ tag. In this case, even though the reduction can answer queries for any message submitted in the c^*-th query, we can still leverage the adversary to break SIS. This is due to the fact that the adversary (computationally) has no information about the reduction's pre-selected signatures that have not been revealed to the adversary.

Therefore, except with negligible probability the forgery made by adversary differs from the pre-selected, unseen signatures or their derivatives.

4.2 Unforgeability Proof

Theorem 4.1 (Unforgeability)**.** *Assuming the hardness of* $\mathsf{SIS}_{q,n,m,\beta}$*, the homomorphic signature scheme described satisfies adaptive unforgeability against chosen dataset attacks as defined in Definition 2.2.*

Proof. The description of reduction $\mathcal{B}$ is as follows:

- **Invocation.** Reduction $\mathcal{B}$ is invoked on a random instance of the $\mathsf{SIS}_{q,n,m,\beta}$ assumption, i.e., $\mathbf{A}^* \in \mathbb{Z}_q^{n\times m}$, and is asked to return an solution $\boldsymbol{e} \in \mathbb{Z}^m$, such that $\mathbf{A}^*\boldsymbol{e} = 0 \mod q$, and $0 \neq ||\boldsymbol{e}|| \leq \beta$.
- **Setup.** Reduction $\mathcal{B}$ runs the following algorithm $\mathsf{Setup}'(1^\lambda, 1^l, 1^{|\boldsymbol{\tau}|}, 1^{d_{max}})$ using the matrix $\mathbf{A}^*$ from the SIS challenge:
  1. Set the parameters q, n, m, B according to the $\mathsf{SIS}_{q,n,m,\beta}$. Let $s_1 = s_1(n), s_2 = s_2(n)$ denote the Gaussian parameters.
  2. Sample a matrix with associated trapdoor:
  $$(\mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{TrapGen}(1^q, 1^n, 1^m)$$
  3. Guess the type of forgery submitted by adversary $\mathcal{A}$. Based on the guess, the reduction $\mathcal{B}$ continues with one of the following three subroutines:
     - **Type I forgery:** Randomly select $|\boldsymbol{t}|$ small matrix $\{\mathbf{S}_i\} \in \mathbb{Z}^{m\times m}$, which can be done by sampling column by column from $D_{\mathbb{Z}^n,s}$. Pick $|\boldsymbol{t}|$ uniformly random scalars $h_0, ..., h_{|\boldsymbol{t}|} \in \mathbb{Z}_q$ with the restriction that $h_0 = 1$, and randomly select $2l+1$ matrices $\mathbf{U}_{i,b}, \mathbf{U} \in \{-1,1\}^{m\times m}$, and set $\mathbf{A} = \mathbf{A}^*\mathbf{U}$. Then set the public key:
     $$\mathsf{pk} = (\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \{\mathbf{D}_{i,b} = \mathbf{A}^*\mathbf{U}_{i,b}\}_{i\in[l],b\in\{0,1\}}, \{\mathbf{T}_i = \mathbf{A}^*\mathbf{S}_i + h_i\mathbf{B}\}_{i\in[|\boldsymbol{t}|]})$$
     - **Type II and Type III forgery**: Randomly select $|\boldsymbol{t}|$ small matrices $\{\mathbf{S}_i\} \in \mathbb{Z}^{m\times m}$, a binary matrix $\mathbf{U} \in \{-1,1\}^{m\times m}$, and $|\boldsymbol{t}|$ uniformly random scalars $h_0, ..., h_{|\boldsymbol{t}|} \in \mathbb{Z}_q$ with the restriction that $h_0 = 1$. Set $\mathbf{T}_i = \mathbf{A}^*\mathbf{S}_i + h_i\mathbf{B}$, for $i \in |\boldsymbol{t}|$. Then randomly select a tag $\boldsymbol{t}^*$. Set $\mathbf{A} = \mathbf{A}^*\mathbf{U} - \sum_i(-1)^{\boldsymbol{t}^*[i]}\mathbf{T}_i$. Sample $[\mathbf{R}_{i,2,b}|\mathbf{R}_{i,1,b}]^T \leftarrow (D_{\mathbb{Z}^m,s_2})^{2m}$ for $i \in [l], b \in \{0,1\}$. Pick a random binary permutation $\pi_i : \{0,1\} \to \{0,1\}$, for $i \in [l]$, and set:
     $$\mathbf{D}_{i,\pi_i(b)} = [\mathbf{A}^*|\mathbf{A}^*\mathbf{U}]\begin{bmatrix}\mathbf{R}_{i,2,b}\\ \mathbf{R}_{i,1,b}\end{bmatrix} - b\mathbf{B} \qquad \forall i \in [l], \forall b \in \{0,1\}$$
     Finally, randomly select a counter value c^*, and initialize a counter $c = 0$. The public key is output as:
     $$\mathsf{pk} = (\mathbf{A}^*, \mathbf{A}, \mathbf{B}, \{\mathbf{D}_{i,b}\}_{i\in[l],b\in\{0,1\}}, \{\mathbf{T}_i = \mathbf{A}^*\mathbf{S}_i + h_i\mathbf{B}\}_{i\in|[\boldsymbol{t}]|}, \mathbf{T_B})$$
- **Queries.** Reduction $\mathcal{B}$ answers adaptive message queries from $\mathcal{A}$ on any message $\boldsymbol{m}$ as follows. The answers are constructed differently depending on the setup-time guess by reduction $\mathcal{B}$ as to the type of forgery it will be given.
  - **Type I forgery:** Reduction $\mathcal{B}$ answers queries $\boldsymbol{\mu}$ as follows:
    1. Choose a random tag $\boldsymbol{\tau} = (\boldsymbol{t}|\boldsymbol{b}) \in \{0,1\}^{\lambda+l}$ with the restriction that $\boldsymbol{t}[0] = 0$.

2. Compute the matrix $\mathbf{T}_{\boldsymbol{t}} = \sum_i (-1)^{\boldsymbol{t}[i]} \mathbf{T}_i$. Also compute the scalar $h_{\boldsymbol{t}} = \sum_i (-1)^{\boldsymbol{t}[i]} h_i$, and if $h_{\boldsymbol{t}} = 0$ then abort this simulation.
3. Compute the dataset matrix
$$\mathbf{A}_{\boldsymbol{t}} = [\mathbf{A}^* | \mathbf{A} + \mathbf{T}_{\boldsymbol{t}}]$$
4. For all $i \in [l]$, compute the signature $\sigma_i = [\mathbf{R}_2 | \mathbf{R}_1]$ as follows:
$$[\mathbf{R}_2^T | \mathbf{R}_1^T]^T \leftarrow \mathsf{SampleRight}(\mathbf{A}^*, \mathbf{B}, \mathbf{U} + \sum_i (-1)^{t[i]} \mathbf{S}_i, \mathbf{T}_{\mathbf{B}}, \mathbf{D}_{i,\boldsymbol{b}[i]} + \mu_i \mathbf{B}, s_2)$$
5. Output the tag $\boldsymbol{\tau} = (\boldsymbol{t}|\boldsymbol{b})$ and the signature vector $\boldsymbol{\sigma} = \{\sigma_i\}_{i \in [l]}$.
6. Increment the counter c.

- **Type II and Type III forgery:** Reduction $\mathcal{B}$ answers queries $\boldsymbol{\mu}$ as follows:
If the counter $c \neq c^*$, pick a random tag $\boldsymbol{t}$ and check if $h_{\boldsymbol{t}} := \sum_i (-1)^{\boldsymbol{t}[i]} h_i - (-1)^{\boldsymbol{t}^*[i]} h_i \mod q$ is equal to 0. If so, abort. Otherwise, continue to answer the queries using the trapdoor $\mathbf{T}_{\mathbf{B}}$ for $\mathbf{B}$, in a similar fashion as in answering Type 1 queries.
Otherwise, if $c = c^*$, proceed as follows:

  1. Set $\boldsymbol{b}[i] = \pi_i(\mu_i)$ for $i \in [l]$, and set the tag $\boldsymbol{\tau} = (\boldsymbol{t}^*|\boldsymbol{b})$ where $\boldsymbol{t}^*$ is the special tag selected at setup time for Type II forgeries. Output the resulting tag $\boldsymbol{\tau}$.
  2. Output the signature vector $\boldsymbol{\sigma} = \{\sigma_i = [\mathbf{R}_{i,2,\mu_i}^T | \mathbf{R}_{i,1,\mu_i}^T]^T\}_{i \in [l]}$ taken from the signatures pregenerated in the Type II setup phase.

  Regardless of which case, increment the counter c.

- **Forgery.** Reduction $\mathcal{B}$ receives from $\mathcal{A}$ a forgery tuple $(\boldsymbol{\tau}^* = (\boldsymbol{t}^*|\boldsymbol{b}^*), \mu^*, \sigma^*, C^*)$.

1. If the type of forgery submitted by adversary $\mathcal{A}$ is different than the type that reduction $\mathcal{B}$ initially guessed, then abort the simulation.
2. Otherwise, construct a solution to the $\mathsf{SIS}_{q,n,m,\beta}$ challenge as follows:
   - **Type I forgery:** In this case, the adversary $\mathcal{A}$ never asked to sign any messages according to challenge tag $\boldsymbol{t}^*$. Compute the tag matrix and scalar:
   $$\mathbf{T}^* = \sum_i (-1)^{\boldsymbol{t}[i]} \mathbf{T}_i, \qquad h^* = \sum_i (-1)^{\boldsymbol{t}[i]} h_i$$
   If $h^* \neq 0 \mod q$, abort this simulation (we could have aborted during setup, when the tag $\boldsymbol{t}^*$ was selected). We show that the completion probability with respect to this is non-negligible in Lemma 4.3. Otherwise, we have
   $$[\mathbf{A}^* | \mathbf{A}^*(\mathbf{U} + \sum_i (-1)^{\boldsymbol{t}^*[i]} \mathbf{S}_i)] \begin{bmatrix} \mathbf{R}_2^* \\ \mathbf{R}_1^* \end{bmatrix} = \mathbf{D}_C + \mu^* \mathbf{B}$$
   Rearranging and applying Lemma 4.4, which shows $\mathbf{D}_C = \mathbf{A}^* \mathbf{U}_C + k\mathbf{B}$, we can obtain
   $$\mathbf{A}^*(\mathbf{R}_2^* + (\mathbf{U} + \sum_i (-1)^{\boldsymbol{t}^*[i]} \mathbf{S}_i) \mathbf{R}_1^* - \mathbf{U}_C) = (k + \mu^*)\mathbf{B}$$
   If $k + \mu^* = 0$, then output $(\mathbf{R}_2^* + (\mathbf{U} + \sum_i (-1)^{\boldsymbol{t}^*[i]} \mathbf{S}_i)])\mathbf{R}_1^* - \mathbf{U}_C)$ as the solution for SIS-instance $\mathbf{A}^*$, otherwise, output $(\mathbf{R}_2^* + (\mathbf{U} + \sum_i (-1)^{\boldsymbol{t}^*[i]} \mathbf{S}_i))\mathbf{R}_1^* - \mathbf{U}_C)\mathbf{T}_{\mathbf{B}}$ as the solution.

- **Type II forgery:** In this case, the adversary $\mathcal{A}$ submits the forgery tuple $(\boldsymbol{\tau}^*, \sigma^*, \mu^*, C^*)$, where the tag $\boldsymbol{\tau}^* = (\boldsymbol{t}^*|\boldsymbol{b}^*)$ has been used for generating signatures for messages $\boldsymbol{\mu}^*$ before, and $C(\boldsymbol{\mu}^*) \neq \mu^*$. If the challenge tag is not $\boldsymbol{t}^*$ which is chosen at the Setup phase, then abort this simulation. Then, reduction $\mathcal{B}$ uses the sequence of signature/message pairs presampled in the setup phase, which has been queried under tag $\boldsymbol{t}^*$ to honestly compute σ' by
$$\sigma' = [\mathbf{R}_2'|\mathbf{R}_1']^T \leftarrow \mathsf{Eval}(\mathsf{pk}, \boldsymbol{t}^*, \boldsymbol{\sigma}^*, \boldsymbol{\mu}^*, C^*)$$
Per correctness, we have
$$\mathbf{A}_{\boldsymbol{t}} \begin{bmatrix} \mathbf{R}_2' \\ \mathbf{R}_1' \end{bmatrix} = \mathbf{D}_{C^*} + C^*(\boldsymbol{\mu}^*)\mathbf{B}$$
$$\mathbf{A}_{\boldsymbol{t}} \begin{bmatrix} \mathbf{R}_2^* \\ \mathbf{R}_1^* \end{bmatrix} = \mathbf{D}_{C^*} + \mu^*\mathbf{B}$$
Since σ^* is a forged signature on a different message $\mu^* \neq C^*(\boldsymbol{\mu}^*)$, it follows that $\sigma' - \sigma^*$ is necessarily non-zero. Thus, we have
$$\mathbf{A}_{\boldsymbol{t}} \begin{bmatrix} \mathbf{R}_2' - \mathbf{R}_2^* \\ \mathbf{R}_1' - \mathbf{R}_1^* \end{bmatrix} = (C^*(\boldsymbol{\mu}^*) - \mu^*)\mathbf{B}$$
Then, by expanding the matrix $\mathbf{A}_{\boldsymbol{t}}$ and rearranging, we can get
$$\mathbf{A}^*((\mathbf{R}_2' - \mathbf{R}_2^*) + \mathbf{U}(\mathbf{R}_1' - \mathbf{R}_1^*)) = (C^*(\boldsymbol{\mu}^*) - \mu^*)\mathbf{B}$$
Output $((\mathbf{R}_2' - \mathbf{R}_2^*) + \mathbf{U}(\mathbf{R}_1' - \mathbf{R}_1^*))\mathbf{T}_{\mathbf{B}}$ as the SIS solution for matrix $\mathbf{A}^*$.
- **Type III forgery:** In this case, the adversary submits the forgery tuple $(\boldsymbol{\tau}^* := (\boldsymbol{t}^*|\boldsymbol{b}^*), \sigma^*, \mu^*, C^*)$, where the tag $\boldsymbol{t}^*$ has been used for generating signatures for messages $\boldsymbol{\mu}^*$ before, but $\boldsymbol{b}^*$ is not equal to the $\boldsymbol{b}$ used in the corresponding query. If the challenge tag is not $\boldsymbol{t}^*$ which is chosen at the Setup phase, then abort this simulation. Otherwise, we have
$$[\mathbf{A}^*|\mathbf{A}^*\mathbf{U}] \begin{bmatrix} \mathbf{R}_2^* \\ \mathbf{R}_1^* \end{bmatrix} = \mathbf{D}_{C^*} + \mu^*\mathbf{B}$$
where $\mathbf{D}_{C^*}$ is computed by using $\{\mathbf{D}_{i,\boldsymbol{b}^*[i]}\}_{i\in[l]}$ in the evaluation algorithms, and by Lemma 4.4 and rearranging, we have
$$\mathbf{A}^*(\mathbf{R}_2^* + \mathbf{U}\mathbf{R}_1^* - \mathbf{U}_C) = (k + \mu^*)\mathbf{B}$$
However, our argument here relies on the requirement that $b^* \neq b$. We show that
$$\mathbf{Pr}[\mathbf{R}_2^* + \mathbf{U}\mathbf{R}_1^* - \mathbf{U}_C \neq 0 \mid \boldsymbol{b} \neq \boldsymbol{b}^*] = \mathsf{negl}(\lambda)$$
for two reasons. First, by Lemma 4.2 proved below, matrix $\{\mathbf{D}_{i,\boldsymbol{b}^*[i]}\}_{i\in[l]}$ is indistinguishable from their counterpart in the real scheme, thus it reveals negligible information about the pre-selected signatures. Secondly, if adversary $\mathcal{A}$ selects a different $\boldsymbol{b}$, then by computation in Lemma 4.4, $\mathbf{U}_C$ has negligible possibility to be $\mathbf{R}_2^* + \mathbf{U}\mathbf{R}_1^*$. Therefore, if $k + \mu^* = 0$, then output $(\mathbf{R}_2^* + \mathbf{U}\mathbf{R}_1^* - \mathbf{U}_C)$ as the solution for SIS-instance $\mathbf{A}^*$, otherwise, output $(\mathbf{R}_2^* + \mathbf{U}\mathbf{R}_1^* - \mathbf{U}_C)\mathbf{T}_{\mathbf{B}}$ as the solution.

□

Lemma 4.2. *Let* $(\mathsf{pk}, \{\boldsymbol{\sigma}_i\})$ *be the output in the real execution, and* $(\mathsf{pk}^*, \{\boldsymbol{\sigma'}_i\})$ *be the output in the simulated execution described above, We show that the two distributions are statistically indistinguishable.*

Proof. Although we describe three different types of forgeries above, we may notice that the public key pk and signing algorithm for Type II and Type III forgery are identical. Therefore, we only analyze the first two types of forgery regarding $(\mathsf{pk}^*, \{\boldsymbol{\sigma'}_i\})$.

Type I forgery. The differences between real execution and simulation can be summarized as follows:

- In real Setup, matrix $\mathbf{A}^*$ is sampled from algorithm $\mathsf{TrapGen}$ together with its trapdoor $\mathbf{T}_{\mathbf{A}^*}$. However, in the simulated Setup algorithm, matrix $\mathbf{A}^*$ is chosen by the SIS generator without its trapdoor $\mathbf{T}_{\mathbf{A}^*}$.

- In real Setup, matrix $(\mathbf{A}, \{\mathbf{D}_{i,b}\}_{i\in[l],b\in\{0,1\}}, \{\mathbf{T}_i\}_{i\in[|\boldsymbol{t}|]})$ are chosen uniformly at random. However, in the simulated Setup algorithm, matrix $\mathbf{A} = \mathbf{A}^*\mathbf{U}$ for a uniformly random matrix $\mathbf{U} \in \{-1, 1\}^{m\times m}$. For $i \in [l], b \in \{0, 1\}$, $\mathbf{D}_{i,b} = \mathbf{A}^*\mathbf{U}_{i,b}$ for uniformly random matrix $\mathbf{U}_{i,b} \in \{-1, 1\}^{m\times m}$. Also for $i \in [|\boldsymbol{t}|]$, $\mathbf{T}_i = \mathbf{A}^*\mathbf{S}_i + h_i\mathbf{B}$ for $\mathbf{S}_i \in D_{\mathbb{Z}^{m\times m},s}$ and random scalars $h_i \in \mathbb{Z}_q$ with the restriction $h_0 = 1$.

- In the real Sign algorithm, every vector of signatures $\boldsymbol{\sigma}_i$ is obtained by using $\mathsf{SampleLeft}$ algorithm and a trapdoor for matrix $\mathbf{A}^*$. However, in the simulated Sign algorithm, the signatures $\boldsymbol{\sigma}_i$ is obtained by using $\mathsf{SampleRight}$ algorithm and a trapdoor for $\mathbf{B}$.

We now argue that the distribution $(\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \{\mathbf{D}_{i,b}\}_{i\in[l],b\in\{0,1\}}, \{\mathbf{T}_i\}_{i\in[|\boldsymbol{t}|]}, \mathbf{P})$ is statistically indistinguishable in real and simulated execution. By the property shown algorithm $\mathsf{TrapGen}$ in Lemma 2.4, $\mathbf{A}^*$ is distributed statistically close to uniform. Let $\mathbf{A}' = [\mathbf{A}|\mathbf{T}_0|\cdots|\mathbf{T}_{|\boldsymbol{t}|}|\mathbf{D}_{1,0}|\mathbf{D}_{1,1}|\cdots|\mathbf{D}_{l,1}]$. Then, by Lemma 2.3, it holds that

$$(\mathbf{A}^*, \mathbf{A}') \approx (\mathbf{A}^*, [\mathbf{A}^*\mathbf{U}|\mathbf{A}^*\mathbf{S}_0 + \mathbf{B}|\mathbf{A}^*\mathbf{S}_1 + h_1\mathbf{B}|\cdots|\mathbf{A}^*\mathbf{U}_{1,0}|\cdots|\mathbf{A}^*\mathbf{U}_{l,1}])$$

Tuple $(\mathbf{B}, \mathbf{P})$ is generated identically in both executions. In conclusion, pk in real execution is statistically indistinguishable from pk' in the simulated execution.

For the signing algorithm in both executions, consider a signing query on messages $\boldsymbol{\mu}$. Note that the simulation aborts only if $h_{\boldsymbol{t}} = 0$, which happens with negligible probability. Let $\mathbf{A}_{\boldsymbol{t}} = [\mathbf{A}^*|\mathbf{A} + \mathbf{T}_{\boldsymbol{t}}]$, where $\mathbf{T}_{\boldsymbol{t}} = \sum_i(-1)^{\boldsymbol{t}[i]}\mathbf{T}_i$ for a randomly chosen $\boldsymbol{t}$. Let $\mathbf{C} = \mathbf{D}_{i,\boldsymbol{b}[i]} + \mu_i\mathbf{B}$ be the coset defined by message μ_i for a randomly chosen $\boldsymbol{b}$. By Lemma 2.6, 2.7, for sufficiently large Gaussian parameter s_2, the outputs of algorithms $\mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ are distributed statistically close to $D_{\Lambda_{\mathbf{A}_{\boldsymbol{t}}+\mathbf{C}},s_2}$. Therefore, we prove the claim for Type I forgery.

Type II and Type III forgery. We start the analysis for Type II (and Type III) by summarizing the differences between real and simulated executions:

- In the real Setup, matrix $\mathbf{A}^*$ is sampled from algorithm $\mathsf{TrapGen}$ together with its trapdoor $\mathbf{T}_{\mathbf{A}^*}$. However, in the simulated Setup algorithm, matrix $\mathbf{A}^*$ is chosen by the SIS generator without its trapdoor $\mathbf{T}_{\mathbf{A}^*}$.

- In the real Setup, matrix $(\mathbf{A}, \{\mathbf{D}_{i,b}\}_{i\in[l],b\in\{0,1\}}, \{\mathbf{T}_i\}_{i\in[|\boldsymbol{t}|]})$ are chosen uniformly at random. However, in the simulated Setup algorithm, for $i \in [|\boldsymbol{t}|]$, $\mathbf{T}_i = \mathbf{A}^*\mathbf{S}_i + h_i\mathbf{B}$ for $\mathbf{S}_i \in D_{\mathbb{Z}^{m\times m},s}$ and random scalars $h_i \in \mathbb{Z}_q$ with the restriction $h_0 = 1$, matrix $\mathbf{A} = \mathbf{A}^*\mathbf{U} - \sum_i(-1)^{\boldsymbol{t}^*[i]}\mathbf{T}_i$ for a uniformly random matrix $\mathbf{U} \in \{-1, 1\}^{m\times m}$. For $i \in [l], b \in \{0, 1\}$, matrix $\mathbf{D}_{i,\pi_i(b)} = [\mathbf{A}^*|\mathbf{A}^*\mathbf{U}] \begin{bmatrix}\mathbf{R}_{i,2,b}\\ \mathbf{R}_{i,1,b}\end{bmatrix} - b\mathbf{B}$ for a random binary permutation $\pi : \{0, 1\} \to \{0, 1\}$, and $[\mathbf{R}_{i,2,b}|\mathbf{R}_{i,1,b}]^T \leftarrow (D_{\mathbb{Z}^m,s_2})^{2m}$ for $i \in [l], b \in \{0, 1\}$.

- The simulated signing algorithm in Type II is the same as in Type I forgery for $c \neq c^*$, which means we only need to show that the signatures distributions in both executions are statistically indistinguishable for c^*.

We now argue that the distribution $(\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \{\mathbf{D}_{i,b}\}_{i\in[l],b\in\{0,1\}}, \{\mathbf{T}_i\}_{i\in[|t|]}, \mathbf{P})$ is statistically indistinguishable in real and simulated execution. By the property shown algorithm TrapGen in Lemma 2.4, $\mathbf{A}^*$ is distributed statistically close to uniform. Let $\mathbf{A}' = [\mathbf{A}|\mathbf{T}_0|\cdots|\mathbf{T}_{|t|}|\mathbf{D}_{1,0}|\mathbf{D}_{1,1}|\cdots|\mathbf{D}_{l,1}]$. Then by Lemma 2.3, it holds that

$$(\mathbf{A}^*, \mathbf{A}') \approx (\mathbf{A}^*, [\mathbf{A}^*\mathbf{U}|\mathbf{A}^*\mathbf{S}_0 + \mathbf{B}|\mathbf{A}^*\mathbf{S}_1 + h_1\mathbf{B}|...|\mathbf{A}^*(\mathbf{R}_{1,2,0} + \mathbf{U}\mathbf{R}_{1,1,0}) - b\mathbf{B}|\cdots]$$

Tuple $(\mathbf{B}, \mathbf{P})$ is generated identically in both executions. In conclusion, pk in real execution is statistically indistinguishable from pk' in the simulated execution.

For the signing algorithms in both executions, as we analyzed above, we only need to consider the case where $c = c^*$, which is the challenge messages. However, without the public key, the signatures on challenge messages are statistically close to real execution by Lemma 2.6. Together with these two cases, we prove the claim for Type II (and Type III) forgery.

□

Lemma 4.3. *For a prime modulus $q = q(\lambda)$ and a number of queries $Q > 0$, the simulation completes both the Queries and Forgery phases without aborting, with probability*

$$\frac{1}{q}(1 - \frac{Q}{q}) \leq \mathbf{Pr}[\mathsf{Completion}] \leq \frac{1}{q}$$

where Q is the number of queries made by adversary. In particular, when $Q \leq \frac{q}{2}$, this probability is $\mathbf{Pr}[\mathsf{Completion}] \in [\frac{1}{2q}, \frac{1}{q}]$ regardless of the adversary's strategy.

Proof. Although the way we apply the scalar is different from the scenario in [18], but the proof stays exactly the same. Therefore, we omit the proof here. □

We adapt the similar lemma in [32], and state a more general case below:

Lemma 4.4. *Let C be an arbitrary circuit with at most l bit input and one bit output. Let $\mathbf{A}^*, \mathbf{B} \in \mathbb{Z}_q^{n\times m}$. For all three type forgeries above, fix the public key for all input wires as $D_{i,b} = \mathbf{A}^*\mathbf{U}_{i,b} + k\mathbf{B}$, for some low norm matrix $\mathbf{U}_{i,b}$, where $i \in [l], b \in \{0,1\}$. For each gate $g = (u, v, w)$, assume input public keys $\mathbf{D}_u, \mathbf{D}_v$ are fixed and let*

$$\mathbf{D}_w = \mathbf{D}_v\mathbf{P}\mathsf{BitDecomp}(\mathbf{D}_u) - \mathbf{B}$$

where $\mathbf{P}$ is of low norm, satisfying $\mathbf{B}\mathbf{P} = \mathsf{Powersof2}(\mathbf{I}_n)$. Then, the public key associated with the output wire of the circuit C is of the form $\mathbf{A}^*\mathbf{U}_C + k\mathbf{B}$, where $\mathbf{U}_C$ is of low norm and $k \in \mathbb{Z}$.

Proof. Instead of sampling one array of matrix $\mathbf{D}_i$ in [32], in the security proof described above, we sample two sequences of matrix $\mathbf{D}_{i,b}, b \in [l]$ in all three types of forgeries. However, since the main proof of this lemma stays almost the same as in [32], thus we only sketch the proof as follows:

Base case. We divide the base case into two sub-cases which corresponds to our unforgeability proof described above:

- **Type I forgery:** The public key associated with each input wires is $\mathbf{D}_{i,b} = \mathbf{A}^*\mathbf{U}_{i,b}$, where $\mathbf{U}_{i,b} \in \{-1,1\}^{m\times m}$. Thus, it certainly satisfied the form $\mathbf{A}^*\mathbf{U}_C + k\mathbf{B}$ with $k = 0$.
- **Type II and III forgery:** The public key associated with each input wires is

$$\mathbf{D}_{i,\pi_i(b)} = [\mathbf{A}^*|\mathbf{A}^*\mathbf{U}]\begin{bmatrix}\mathbf{R}_{i,2,b}\\ \mathbf{R}_{i,1,b}\end{bmatrix} - b\mathbf{B} \qquad \forall i \in [l], \forall b \in \{0,1\}$$

  where $[\mathbf{R}_{i,2,b}|\mathbf{R}_{i,1,b}]^T \leftarrow (D_{\mathbb{Z}^m,s_2})^{2m}$, and $\mathbf{U} \in \{-1,1\}^{m\times m}$. Thus, by re-arranging, we can say that it still satisfies the form $\mathbf{A}^*\mathbf{U}_C + k\mathbf{B}$ with $k = 0$ or 1.

Induction step. Now consider a gate $g = (u, v, w)$, where the public keys $\mathbf{D}_u, \mathbf{D}_v$ satisfies $\mathbf{D}_u = \mathbf{A}^*\mathbf{U}_u + i_u\mathbf{B}, \mathbf{D}_u = \mathbf{A}^*\mathbf{U}_u + i_u\mathbf{B}$ for some integers i_u, i_v. Then we have

$$\begin{aligned}\mathbf{D}_w &= \mathbf{D}_v\mathbf{P}\mathsf{BitDecomp}(\mathbf{D}_u) - \mathbf{B}\\ &= (\mathbf{A}^*\mathbf{U}_v + i_v\mathbf{B})\mathbf{P}\mathsf{BitDecomp}(\mathbf{D}_u) - \mathbf{B}\\ &= \mathbf{A}^*\mathbf{U}_v\mathbf{P}\mathsf{BitDecomp}(\mathbf{D}_u) + i_v\mathsf{Powersof2}(\mathbf{I}_n)\mathsf{BitDecomp}(\mathbf{D}_u) - \mathbf{B}\\ &= \mathbf{A}^*(\mathbf{U}_v\mathbf{P}\mathsf{BitDecomp}(\mathbf{D}_u) + \mathbf{U}_u) + i_w\mathbf{B}\end{aligned}$$

By analyzing the norm of $\mathbf{U}_w = \mathbf{U}_v\mathbf{P}\mathsf{BitDecomp}(\mathbf{D}_u) + \mathbf{U}_u$, we have $||\mathbf{U}_w|| = O(s_1 m^3)$. Therefore, by induction, the norm of $\mathbf{U}_C$ should be $O(s_1 m^{O(d_{max})})$. □

4.3 Parameter Selection

Let λ denote the security parameter. First, we set the lattice parameter $n = \mathsf{poly}(\lambda)$. For the size of the signature, we set its bound to be $B = \omega(2^{d_{\mathsf{max}}})$, where d_{max} is the maximum depth of circuit. Then we set the modulus $q = q(n) = n^{O(d_{\mathsf{max}})} > B$, and $m = O(n \log q)$.

The Gaussian parameter in SampleD is set to be $s_1 = O(\sqrt{n \log q})$, and in order to achieve indistinguishability of algorithms SampleLeft and SampleRight, their common parameter should be set to $s_2 = \omega(m \log q \sqrt{\log m})$.

A signature after homomorphic evaluation is $\mathbf{R}_w := \mathbf{R}_v\tilde{\mathbf{D}}_u - y\mathbf{R}_u$, where $\tilde{\mathbf{D}_u}$ is output by algorithm SampleD. Setting the parameters for s_1, s_2 as above, we find the size of the evaluated signature to be $O(s_1 s_2 m^3)$.

5 Achieving Unforgeability Against Fully Adaptive Chosen Message Attack

We propose several adaptive security models for homomorphic signatures. The strongest, EUF-FH-CMA, differs from the weakest, EUF-FH-CDA, in that the strict model (EUF-FH-CMA) allows fully adaptive queries at the message level, while the relaxed model (EUF-FH-CMA) only allows adaptive queries at the dataset level. We have so far worked in the weaker model, which make the proofs simpler. In order to prove the stronger notion, we must modify the scheme: specifically, how the dataset tag $\boldsymbol{\tau}$ is assembled and used.

The Problem. Recall that in our scheme, $\boldsymbol{\tau} = (\boldsymbol{t}|\boldsymbol{b})$. Handling $\boldsymbol{t}$ in the security proof for the fully adaptive chosen message model presents no difficulty: we do exactly the same as in the chosen dataset model. Specifically, we pre-select a random $\boldsymbol{t}'$ which we use when answering the adversary's query on one dataset at a random point in the simulation. We catch him if he makes a forgery (of Type II or Type III) that uses the same $\boldsymbol{t}'$. Since the random choice of $\boldsymbol{t}'$ does not depend on the messages, it does not matter in which order they are queried.

With the sub-tag $\boldsymbol{b}$, the situation is more delicate. In a Type-II simulation, we need to fix the value of $\boldsymbol{b}'$ at the last minute, in order to answer an adaptive query on the dataset associated with $\boldsymbol{t}'$ (the special tag pre-selected as above). To do this, the simulator will have prepared $2l$ signatures, one for each of the two possible values of each the l binary messages in the dataset. To answer a query on the j-th message, the simulator needs to "fix" the j-th bit of $\boldsymbol{b}'$ to make one of the two pre-sampled signatures for the j-th message, verify correctly. This fix must occur *on the fly*.

This poses fewer problems in the relaxed model: since *all* the messages in a dataset are queried at once, *all* the bits of $\boldsymbol{b}'$ can be determined before $\boldsymbol{b}'$ is revealed. In the strict model, however, seeing the j-th message query (of the $\boldsymbol{t}'$-dataset) allows the simulator to fix the j-th bit of $\boldsymbol{b}'$, but no other. The simulator would need to predict the full dataset in order to fix the full vector $\boldsymbol{b}'$, which he cannot do in the strict model.

The Solution. Our solution is to detach the (full) vector $\boldsymbol{b} \in \{0,1\}^l$ from the dataset tag $\boldsymbol{\tau}$, and instead attach a (partial) vector $\boldsymbol{b} \in \{0,1,\perp\}^l$ to every signature σ on any message computed from the dataset.

Essentially, while the handling of "$\boldsymbol{t}$-tags" remains unchanged, we are now "$\boldsymbol{b}$-tagging" the signatures bit-by-bit in the following sense: to a signature σ on a message $\mu = C(\boldsymbol{m})$ computed using circuit C, we attach a partial $\boldsymbol{b}$-tag, where some of the bits are set and some of the bits are left unset to $\perp$. The bits that are set to 0 or 1 correspond to the message indices in the dataset that are used by C. The bits that are left unset to $\perp$ correspond to the indices of messages in the dataset that are will not be used by C. In particular, for the initial messages μ_j, the circuit $C = Proj_j(.)$ is a projection circuit with the j-message as the only input, so only the j-th bit of the $\boldsymbol{b}$-tag attached to $\sigma(\mu_j)$ will be set. It will be set at random by Sign, as it would have been in the prior version.

The rest of the scheme does not change. In particular, signatures can be verified combined homomorphically using partial $\boldsymbol{b}$-tags as long as they are bit-wise compatible (where we say that two matching bits are compatible if they are the same, or if at least one of them is $\perp$). The operations Verify and Eval will work as before, since by this rule they will only ever need to use the bits of a (valid) $\boldsymbol{b}$-tags that have already been set, and they can only (validly) be set in compatible ways since the Sign algorithm is not supposed to be called on opposite messages in the same position in the same dataset per the security definition.

This corresponds exactly to the current Eval and Verify operations, where the $\boldsymbol{t}$-tags are used as before to enforce dataset dependence, and the bits of the $\boldsymbol{b}$-tag are merely used to select the matrices $\mathbf{D}_{i,b}$: if an input wire is not used in a given circuit, then the corresponding bit is already ignored in the current versions of Eval and Verify. It is therefore irrelevant whether those ignored bits are set all at once as part of the tag returned in a signature query on a whole dataset, or whether those bits are left unset per this mechanism.

We note that—in the real scheme—the selection and assignment of the bits in $\boldsymbol{b}$ does not depend on either the messages or the dataset tag $\boldsymbol{t}$. The position of the bits of $\boldsymbol{b}$ that are moved from "unset" ($\perp$) to "set" (0 or 1) is solely determined by the connection structure within the circuit, while the actual value of each bit being set (0 versus 1) results from a random choice made by the Sign algorithm as modified.

There exist some minor differences in the unforgeability proof of this new solution comparing to before. We define a notion named *compatible* regarding the vector $\boldsymbol{b}$ for Type II and III forgeries. We require that in Type II forgery, the challenge tag $\boldsymbol{t}^*$ is equal to one queried tag $\boldsymbol{t}$ as before, but $\boldsymbol{b}^*$ is *compatible* with all signature queries. While in Type III forgery, $\boldsymbol{b}^*$ is required not to be *compatible* with all signature queries. We recall that $\boldsymbol{b} \in \{0,1,\perp\}^l$, for all signature queries, we say the vector $\boldsymbol{b}^*$ in the challenge signature is *compatible* with queries if for $i \in [l]$, $\boldsymbol{b}^*[i] = \boldsymbol{b}_j[i]$ for some j in queries or $\boldsymbol{b}^*[i] = \perp$. Although we add a new notion $\perp$ to vector $\boldsymbol{b}$, but we can still leverage the same argument to say that the event related to Type III forgery $\mathbf{Pr}[\mathbf{R}_2^* + \mathbf{U}\mathbf{R}_1^* - \mathbf{U}_C \neq 0 \mid \boldsymbol{b} \neq \boldsymbol{b}^*] = \mathsf{negl}(\lambda)$ still holds. The other parts of unforgeability proof are the same as bofore, thus we omit the detail here.

6 Conclusion

We propose the first fully secure construction of fully homomorphic signatures, in the standard model. The scheme is efficient, and is efficiently reducible to the standard SIS assumption in the case of circuits of poly-logarithmic depth. In the case of polynomial-depth circuits, the reduction is to the sub-exponential average-case SIS assumption.

This paper is subsequent to the recent works by Gorbunov et al. [32] and Wichs [39], but independent and concurrent to the more recent work by Gorbunov et al. [33]. In particular, Gorbunov et al. propose a technique reminiscent of the chameleon hash trick to achieve adaptive security. In contrast, we propose a different approach to this same problem.

Our construction combines the key-homomorphic techniques of [17] with the lattice mixing approach of [18]. Our construction further makes use of a new technique for "boosting" the efficiency and applicability of partitioning-type proofs, thanks to which we are able to ensure that the simulator is *always* able to issue signatures on *every* message query, but in some cases he can only issue *one*.

Although not yet sufficient to prove adaptive security of the key-homomorphic encryption scheme of [17], which remains an open problem, our technique lets us resolve this question for the case of fully homomorphic signatures. We expect this powerful technique to find further applications.

References

[1] Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-based integrity for network coding. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09: 7th International Conference on Applied Cryptography and Network Security*, volume 5536 of *Lecture Notes in Computer Science*, pages 292–305. Springer, June 2009.

[2] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572. Springer, May 2010.

[3] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 98–115. Springer, August 2010.

[4] Shweta Agrawal, Dan Boneh, Xavier Boyen, and David Mandell Freeman. Preventing pollution attacks in multi-source network coding. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 161–176. Springer, May 2010.

[5] Jae Hyun Ahn, Susan Hohenberger, Dan Boneh, and Brent Waters. Computing on authenticated data. In *In Theory of Cryptography TCC 2012, Springer LNCS 7194*, pages 1–20, 2012.

[6] Miklós Ajtai. Determinism versus non-determinism for linear time RAMs (extended abstract). In *31st Annual ACM Symposium on Theory of Computing*, pages 632–641. ACM Press, May 1999.

[7] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 17–34. Springer, March 2011.

[8] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. Computing on authenticated data: New privacy definitions and constructions. In *Advances in Cryptology – ASIACRYPT 2012*, Lecture Notes in Computer Science, pages 367–385. Springer, December 2012.

[9] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 863–874. ACM Press, 2013.

[10] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. Lecture Notes in Computer Science, pages 90–108. Springer, August 2013.

[11] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. pages 326–349, 2012.

[12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *45th Annual ACM Symposium on Theory of Computing*, pages 111–120. ACM Press, 2013.

[13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC 2013: 10th Theory of Cryptography Conference*, Lecture Notes in Computer Science, pages 315–333. Springer, 2013.

[14] Dan Boneh and Xavier Boyen. Efficient selective identity-based encryption without random oracles. *Journal of Cryptology*, 24(4):659–693, October 2011.

[15] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 149–168. Springer, May 2011.

[16] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 1–16. Springer, March 2011.

[17] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *Advances in Cryptology – EUROCRYPT 2014*, Lecture Notes in Computer Science, pages 533–556. Springer, 2014.

[18] Xavier Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 499–517. Springer, May 2010.

[19] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. pages 309–325, 2012.

[20] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 317–336. Springer, March 2009.

[21] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In *Advances in Cryptology – EUROCRYPT 2013*, Lecture Notes in Computer Science, pages 336–352. Springer, 2013.

[22] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic MACs for arithmetic circuits. In *PKC 2014: 17th International Workshop on Theory and Practice in Public Key Cryptography*, Lecture Notes in Computer Science, pages 538–555. Springer, 2014.

[23] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signatures in the standard model. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 680–696. Springer, May 2012.

[24] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer, May 2004.

[25] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 697–714. Springer, May 2012.

[26] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, August 2010.

[27] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology – EUROCRYPT 2013*, Lecture Notes in Computer Science, pages 626–645. Springer, 2013.

[28] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. Lecture Notes in Computer Science, pages 301–320. Springer, December 2013.

[29] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178. ACM Press, May / June 2009.

[30] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206. ACM Press, May 2008.

[31] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 99–108. ACM Press, June 2011.

[32] Sergey Gorbunov and Vinod Vaikuntanathan. (leveled) fully homomorphic signatures from lattices. Cryptology ePrint Archive, Report 2014/463, 2014. `http://eprint.iacr.org/2014/463`.

[33] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. Cryptology ePrint Archive, Report 2014/897, 2014. `http://eprint.iacr.org/`.

[34] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *Topics in Cryptology – CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262. Springer, February 2002.

[35] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science*, pages 436–453. IEEE Computer Society Press, November 1994.

[36] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, April 2012.

[37] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th Annual Symposium on Foundations of Computer Science*, pages 372–381. IEEE Computer Society Press, October 2004.

[38] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, 2013.

[39] Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. Cryptology ePrint Archive, Report 2014/451, 2014. `http://eprint.iacr.org/2014/451`.