(Leveled) Fully Homomorphic Signatures from Lattices

Sergey Gorbunov* MIT

Vinod Vaikuntanathan† MIT

July 6, 2014

Abstract

In a homomorphic signature scheme, given a vector of signatures $\vec{\sigma}$ corresponding to a dataset of messages $\vec{\mu}$, there is a *public* algorithm that allows to derive a signature σ' for message $\mu' = f(\vec{\mu})$ for any function f. Given the tuple (σ', μ', f) anyone can *publicly* verify the result of the computation of function f. Along with the standard notion of unforgeability for signatures, the security of homomorphic signatures guarantees that no adversary is able to make a forgery σ^* for $\mu^* \neq f(\vec{\mu})$.

We construct the first homomorphic signature scheme for evaluating arbitrary functions. In our scheme, the public parameters and the size of the resulting signature grows polynomially with the depth of the circuit representation of f. Our scheme is secure in the standard model assuming hardness of finding *Small Integer Solutions* in hard lattices. Furthermore, our construction has asymptotically fast verification which immediately leads to a new solution for verifiable outsourcing with pre-processing phase. Previous state of the art constructions were limited to evaluating polynomials of constant degree, secure in random oracle model without asymptotically fast verification.

*Email: sergeyg@mit.edu. Supported by Alexander Graham Bell Canada Graduate Scholarship (NSERC-CGSD3).

†Email: vinodv@mit.edu.

1 Introduction

With advances in cloud computing, an increasing amount of sensitive data is stored and computations on them are performed remotely, raising questions of privacy of the data and correctness of computations. Recently, a number of cryptographic schemes have been developed to address these concerns. For example, fully homomorphic encryption [Gen09, BV11, BGV12] enables us to compute on encrypted data, paving the road to achieving privacy in outsourcing. Many flavors of verifiable outsourcing schemes have been developed to deal with the question of correctness of computations (cf. [Mic00, Kil92, GKR08, GGP10, CKV10, AIK10, CKLR11, KRR13b, KRR13a] and many others). A particularly natural way to verifiably outsource computation is through the notion of homomorphic signatures [CJL09, BFKW09, GKKR10, BF11b, BF11a].

A homomorphic signature scheme is one where anyone can homomorphically compute on the signatures $\vec{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_\ell)$ corresponding to a dataset $\vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_\ell)$ and produce a signature σ' for a circuit C and the result $\mu' = C(\vec{\mu})$ of applying C to the dataset $\vec{\mu}$. Given only the public key pk and the signature σ' on the circuit C and a message μ', anyone can verify that σ' is indeed the result of applying C to some set of signed messages $\vec{\mu}$. In order to tie the signature to a particular dataset, we "tag" each dataset of messages, and give the tag to the verification algorithm as well. A key feature is that this verification can be done without knowing the original dataset $\vec{\mu}$.

The signature σ' "proves" that the computation was done correctly, in the sense that computing a signature σ' for any pair (C, μ') where $\mu' \neq C(\vec{\mu})$ is hard for any PPT adversary. Moreover, the resulting signature is compact, namely, its size and the time to verify it depends neither on the size of the original data or the size of the circuit that was computed on it. This gives us a very natural, publicly verifiable scheme to outsource computation (in an amortized setting).

However, constructions of homomorphic signatures have been few and far between. In particular:

- The initial schemes [CJL09, BFKW09, GKKR10, BF11b] handled only linear functions. The state of the art is a scheme of Boneh and Freeman [BF11a] that can compute constant degree polynomial functions on signed messages.
- The schemes are shown secure in the random oracle model.
- Finally, the polynomially homomorphic schemes rely on the short integer solutions (SIS) problem on *ideal lattices*. In contrast, in the case of fully homomorphic *encryption*, we know several solutions by now that rely on the SIS problem on *arbitrary lattices* with no ideal structure.

Our goal in this paper is to overcome these limitations and construct a *fully homomorphic signature* for all circuits, *without random oracles* and based on the SIS problem on *arbitrary lattices*. We describe our results and techniques in more detail below.

1.1 Our Results and Techniques

We show how to build a fully homomorphic signature scheme in steps, our starting point being the "hash-and-sign" paradigm [DH76, RSA78] and the signature scheme of Bellare and Rogaway from trapdoor permutations [BR93]. The signature algorithm generates a trapdoor permutation pair of functions $f_{\mathsf{pk}}, f_{\mathsf{pk}}^{-1}$ and a hash function $H(\cdot)$. The description of f_{pk} and the hash function $H(\cdot)$ form the public key, and f_{pk}^{-1} is kept secret. The signature of message m is obtained by first hashing and then applying the trapdoor inverse function on the result, namely, $\sigma = f_{\mathsf{pk}}^{-1}(H(m))$. To verify the signature, the user checks that

$f_{\mathsf{pk}}(\sigma) = H(m)$. It is not necessary to have a trapdoor *permutation* to make this paradigm work, rather, it is enough to have a *pre-image sampleable* family of surjective trapdoor functions, as defined in [GPV08]. It is natural to ask:

Can we make this signature scheme homomorphic?

Our first idea is to construct and use a special type of hash function $H(\cdot)$. That is, $H(m_1) + H(m_2) = H(m_1 + m_2)$ and $H(m_1) \times H(m_2) = H(m_1 \times m_2)$.[1] Now, suppose the trapdoor function also satisfies these homomorphic properties, then we can add (and multiply) signatures of messages m_1, m_2 to obtain a signature on the result $m_1 + m_2$ (and $m_1 \times m_2$, respectively):

$$\sigma' = f_{\mathsf{pk}}^{-1}\big(H(m_1)\big) + f_{\mathsf{pk}}^{-1}\big(H(m_2)\big) = f_{\mathsf{pk}}^{-1}\big(H(m_1) + H(m_2)\big) = f_{\mathsf{pk}}^{-1}\big(H(m_1 + m_2)\big)$$

To verify, the user checks that $f_{\mathsf{pk}}(\sigma') = H(m_1 + m_2)$, and similarly for multiplication. However, there is a problem: there is no mechanism so far to authenticate the computation performed to arrive at the σ'! That is, a user can apply any function f to a collection of signatures to obtain σ', but then go on to claim that σ' was obtained by applying a different function f'. In other words, wherein lies the mechanism to authenticate computations?

Our key idea is to sign messages using different trapdoor functions. Our key tool is a notion of fully *key-homomorphic* trapdoor functions, a notion recently introduced in the work of Boneh et al. [BGG+14] in the context of attribute-based encryption. Suppose m_1 and m_2 are signed using functions $f_{\mathsf{pk}_1}^{-1}$ and $f_{\mathsf{pk}_2}^{-1}$ respectively, then, assuming the family is also key-homomorphic, anyone can obtain a signature:

$$\sigma' = f_{\mathsf{pk}_1}^{-1}\big(H(m_1)\big) + f_{\mathsf{pk}_2}^{-1}\big(H(m_2)\big) = f_{\mathsf{pk}_1+\mathsf{pk}_2}^{-1}\big(H(m_1 + m_2)\big)$$

The verification algorithm checks that $f_{\mathsf{pk}_1+\mathsf{pk}_2}(\sigma') = H(m_1 + m_2)$ (and respectively, for multiplication).

How can we implement this outline, and why is this secure? To see this, let us look at a toy version of our instantiation of the fully homomorphic hash functions, and the fully key-homomorphic trapdoor functions (these are already good enough to sign a single dataset and compute on the signatures). The recent work of Boneh et al. [BGG+14] constructs a fully key-homomorphic *injective* trapdoor function, which is unsuitable for our purposes. What we need is a fully key-homomorphic *pre-image sampleable* trapdoor function, which we construct.

- Our hash function H is parameterized by $\ell + 1$ uniformly random matrices $\mathbf{B}, \mathbf{D}_1, \ldots, \mathbf{D}_\ell \in \mathbb{Z}_q^{m \times n}$ where n is the lattice dimension, q the modulus and $m = \Omega(n \log q)$. The description of the hash function also contains a "trapdoor" for the matrix $\mathbf{B}$.

  For an ℓ-bit input $\vec{\mu} = (\mu_1, \ldots, \mu_\ell)$, we define

  $$H(\vec{\mu}) = \big(\mathbf{D}_1 + \mu_1\mathbf{B}, \mathbf{D}_2 + \mu_2\mathbf{B}, \ldots, \mathbf{D}_\ell + \mu_\ell\mathbf{B}\big)$$

- The trapdoor function family is parameterized by a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$. The domain of the function is ℓ-tuples of $m \times m$ matrices $\mathbf{R} \in \mathbb{Z}^{2m \times m}$ with "small" entries, and the output is computed as

  $$f_{\mathbf{A}}(\mathbf{R}_1, \ldots, \mathbf{R}_\ell) = \big(\mathbf{A}_1\mathbf{R}_1, \ldots, \mathbf{A}_\ell\mathbf{R}_\ell\big)$$

[1] For the sake of simplicity, we abuse notation by denoting both the operation on the messages and the operation on the hash values by the same operator, $+$ or $\times$. The messages and their hashes live in different worlds and therefore, are operated on differently.

The one-wayness of this function follows from the hardness of the short integer solutions (SIS) problem which asks for a "short" solution to the linear system of equations $\mathbf{AR} = 0$, given a uniformly random $\mathbf{A}$. The parameters are set in a way that there are many solutions to this equation (even many short solutions), but it is hard to find them. The famous worst-case to average-case reductions of Ajtai [Ajt96], Micciancio and Regev [MR07] show that the average-case hardness of SIS follows from the worst-case hardness of several standard lattice problems.

This is also a pre-image sampleable trapdoor function, as shown by Gentry, Peikert and Vaikuntanathan [GPV08]. That is, using a short basis of the lattice $\Lambda^{\perp}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{Ax} = 0 \pmod q\}$, one can sample a random inverse (according to a discrete Gaussian distribution) of any given element in the range of this function.

We now show that the hash function is fully homomorphic and the trapdoor function is fully key-homomorphic. That is one can compute any circuit C (say, composed of NAND gates) on $H(\vec{\mu})$ to get a hash of the output $H(C(\vec{\mu}))$, and given $f^{-1}_{\vec{\mathsf{pk}}}(H(\vec{\mu}))$, one can compute $f^{-1}_{C(\vec{\mathsf{pk}})}(H(C(\vec{\mu}))$. Let us see how to compute a NAND gate on the hash values for the first two bits, namely, $\mathbf{D}_1 + \mu_1\mathbf{B}$ and $\mathbf{D}_2 + \mu_2\mathbf{B}$. We first compute a matrix $\widetilde{\mathbf{D}}_2$ such that $\mathbf{B}\widetilde{\mathbf{D}}_2 = \mathbf{D}_2$. This can be done by invoking the deterministic "nearest plane" algorithm of Babai [Bab86] using the trapdoor for $\mathbf{B}$. We then compute

$$\begin{aligned}(\mathbf{D}_1 + \mu_1\mathbf{B}) \cdot \widetilde{\mathbf{D}}_2 - \mu_1 \cdot (\mathbf{D}_2 + \mu_2\mathbf{B}) = \mathbf{D}_1\widetilde{\mathbf{D}}_2 - \mu_1\mu_2 \cdot \mathbf{B} &= (\mathbf{D}_1\widetilde{\mathbf{D}}_2 - \mathbf{B}) + (\mu_1 \text{ NAND } \mu_2) \cdot \mathbf{B} \\ &= \mathbf{D}_{\mathsf{out}} + (\mu_1 \text{ NAND } \mu_2) \cdot \mathbf{B}\end{aligned}$$

Here, we think of $\mathbf{D}_1\widetilde{\mathbf{D}}_2 - \mathbf{B}$ as the description of the hash function for the output wire of the NAND gate. We proceed iteratively, in a gate-by-gate fashion, to compute the hash of the entire circuit evaluated on the message.

Now, suppose that one has the inverses of $\mathbf{D}_1 + \mu_1\mathbf{B}$ and $\mathbf{D}_2 + \mu_2\mathbf{B}$, namely matrices "short" $\mathbf{R}_1$ and $\mathbf{R}_2$ such that

$$\mathbf{AR}_1 = \mathbf{D}_1 + \mu_1\mathbf{B} \quad \text{and} \quad \mathbf{AR}_2 = \mathbf{D}_2 + \mu_2\mathbf{B}$$

Homomorphic NAND of the signatures follows by observing that

$$\mathbf{A}(\mathbf{R}_1\widetilde{\mathbf{D}}_2 - \mu_1\mathbf{R}_2) = (\mathbf{D}_1 + \mu_1\mathbf{B})\widetilde{\mathbf{D}}_2 - \mu_1(\mathbf{D}_2 + \mu_2\mathbf{B}) = \mathbf{D}_{\mathsf{out}} + (\mu_1 \text{ NAND } \mu_2) \cdot \mathbf{B}$$

Thus, $\mathbf{R}_1\widetilde{\mathbf{D}}_2 - \mu_1\mathbf{R}_2$ is a signature of the NAND of the first two bits which can be computed publicly, by anyone who knows the signatures on the input. Proceeding this way gate-by-gate, one computes the signature of the output of the circuit.

The size of the signatures grow by a $\mathsf{poly}(m)$ factor for each level of the circuit, and thus, to size $m^{O(d)}$ where d is the depth of the evaluated circuit. To ensure correctness and security, we must set the modulus q to be larger than this bound. In other words, to set the system parameter q, one has to fix a maximum depth d_{max} of circuits that we evaluate with the scheme. Hence, we get a leveled fully homomorphic signature scheme.

To deal with multiple datasets, we need a "tagged" version of these fully key-homomorphic trapdoor functions, which we will construct in Section 4, borrowing on ideas from [ABB10a, BGG+14]. Security in the sense of Boneh and Freeman [BF11a] follows from the hardness of the SIS problem. For more details, see Section 4.

2 Homomorphic Signatures: Definitions

We follow the notation introduced by Boneh and Freeman [BF11a]. We use λ to denote the security parameter throughout this paper. Let $\mathcal{M}$ be a message space and let $\mathcal{C}$ be a collection of circuits $C : \mathcal{M}^\ell \to \mathcal{M}$ that take ℓ inputs over the message space $\mathcal{M}$ and produce an output in $\mathcal{M}$. A homomorphic signature scheme for the class of circuits $\mathcal{C}$ (called a $\mathcal{C}$-homomorphic signature scheme) is a tuple of polynomial-time algorithms $\mathcal{HS}$ =(Setup, Sign, Eval, Verify) which work as follows:

- $\mathsf{Setup}(1^\lambda, 1^\ell)$: The setup algorithm takes the security parameter λ, and the maximum size of the dataset ℓ that it signs at any one time. It outputs a public key pk and a secret key sk.
- $\mathsf{Sign}(\mathsf{sk}, t, i, \mu)$: The signing algorithm takes the secret key sk, a tag $t \in \{0,1\}^\lambda$, a message $\mu \in \mathcal{M}$ and an index $i \in [\ell]$, and outputs a signature σ. We remark that the tags are used to differentiate the message sets that one produces signatures for.
- $\mathsf{Eval}(\mathsf{pk}, t, \vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_\ell), \vec{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_\ell), C)$: The evaluation algorithm takes the public key pk, the tag $t \in \{0,1\}^\lambda$, a vector of messages $\vec{\mu}$ and their signatures $\vec{\sigma}$, and a circuit $C \in \mathcal{C}$, and outputs a derived signature σ'.
- $\mathsf{Verify}(\mathsf{pk}, t, \mu', \sigma', C)$: The verification algorithm takes the public key pk, the tag $t \in \{0,1\}^\lambda$, a message/signature pair μ, σ and a circuit $C \in \mathcal{C}$. It outputs reject (0) or accept (1).

We remark that the verification algorithm always takes the result of a homomorphic evaluation as input. This is without loss of generality since in this work we are interested in arbitrary circuit collections, which include all "projection circuits" that output their i'th argument, namely all circuits $P_i(\mu_1, \mu_2, \ldots, \mu_\ell) = \mu_i$. In particular, this will allow for verification of individual message signatures.

Definition 2.1 (Correctness)**.** *We say that a $\mathcal{C}$-homomorphic signature scheme $\mathcal{HS}$ is correct if:*

1. *For all tags $t \in \{0,1\}^\lambda$, all $\mu \in \mathcal{M}$, and all $i \in [\ell]$:*

$$\Pr\left[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell); \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, t, i, \mu) : \mathsf{Verify}(\mathsf{pk}, t, \mu, \sigma, P_i) = 1\right] = 1$$

   where the probability is over the coins of the algorithms in $\mathcal{HS}$.

2. *For all tags $t \in \{0,1\}^\lambda$, all messages $\vec{\mu} = (\mu_1, \ldots, \mu_\ell) \in \mathcal{M}^\ell$, and all circuits $C \in \mathcal{C}$:*

$$\Pr\Big[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell);\ \sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, t, i, \mu_i);$$
$$\mu' = C(\mu_1, \mu_2, \ldots, \mu_\ell);\ \sigma' \leftarrow \mathsf{Eval}(\mathsf{pk}, t, \vec{\mu}, \vec{\sigma}, C) :\ \mathsf{Verify}(\mathsf{pk}, t, C(\vec{\mu}), \sigma', C) = 1\Big] = 1$$

   where the probability is over the coins of the algorithms in $\mathcal{HS}$.

We say that a signature scheme is *fully homomorphic* if it is homomorphic with respect to a family of all poly-size circuits $\mathcal{C}$. In this paper, we construct *leveled fully homomorphic* signature schemes that are homomorphic for the class of all circuits whose depth is bounded by a polynomial function $d = d(\ell)$.

Length Efficiency. An important criterion that makes homomorphic signature schemes non-trivial is the notion of length efficiency. We say that a fully homomorphic signature scheme $\mathcal{HS}$ is length-efficient if for a fixed security parameter λ, the length of the derived signature depends only on λ, and not on either the size of the dataset ℓ or the size of the circuit C. In the case of a leveled fully homomorphic signature scheme, we relax this requirement and allow the length of the signature to depend polynomially on the depth of the evaluated circuit (but not on its size).

2.1 Security

We now define the notion of unforgeability of homomorphic signature schemes.

Definition 2.2 (Unforgeability)**.** *A homomorphic signature scheme $\mathcal{HS}$ is unforgeable with respect to a class of circuits $\mathcal{C}$ taking ℓ inputs if for all security parameters $\lambda \in \mathbb{N}$, and all probabilistic poly-time adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ winning the following game is negligible in λ. (We will let $\mathcal{O}_{\mathsf{sk}}(\cdot)$ denote an oracle that takes a vector of messages $\vec{\mu} \in \mathcal{M}^{\ell}$ as input, chooses a random tag $t \in \{0,1\}^{\lambda}$ and outputs a set of signatures $\vec{\sigma}$ and the tag t, where $\sigma_i = \mathsf{Sign}(\mathsf{sk}, t, i, \mu_i)$).*

$\underline{\mathsf{Game}_{\mathcal{HS},\mathcal{A}}(1^{\lambda})}$**:**

1: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\ell})$
2: $(t^*, \mu^*, \sigma^*, C^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})$
   // Let q be the number of oracle calls to $\mathcal{O}_{\mathsf{sk}}$
   // Let $\{\vec{\mu}_j\}_{j\in[q]}$ be the datasets queried to $\mathcal{O}_{\mathsf{sk}}$ and
   // let $\{\vec{\sigma}_j, t_j\}_{j\in[q]}$ be the associated replies
3: Output 1 iff $\mathsf{Verify}(\mathsf{pk}, t^*, \mu^*, \sigma^*, C^*) = 1$ and
   (a) $t^* \neq t_j$ for all $j \in [q]$ **or**
   (b) $t^* = t_j$ for some $j \in [q]$ but $\mu^* \neq C^*(\vec{\mu}_j)$.

Remarks. A forgery of type (a) corresponds to the standard notion of unforgeability for signature schemes. That is, the adversary is able to produce a valid signature for a fresh tag which was never issued in the honest execution. The forgery of type (b) corresponds to the security requirement for homomorphic signatures. Namely, here, the adversary should not be able to falsely claim the output of a circuit of her choice on a signed dataset.

Definition 2.3 (Selective-unforgeability)**.** *Define the game $\mathsf{Game}_{\mathcal{HS},\mathcal{A}}(1^{\lambda})^{\mathsf{sel}}$ to be the same as $\mathsf{Game}_{\mathcal{HS},\mathcal{A}}(1^{\lambda})$ except that the adversary outputs the dataset of messages $\vec{\mu}$ and the challenge tag $t^* \in \{0,1\}^{\lambda}$ before the challenger runs the Setup algorithm ($\vec{\mu}$ can be $\perp$ in the case the adversary will make a type (a) forgery).*

We say a homomorphic signature scheme $\mathcal{HS}$ is selectively-unforgeable with respect to a circuit class $\mathcal{C}$ if no adversary can win the game $\mathsf{Game}^{\mathsf{sel}}_{\mathcal{HS},\mathcal{A}}(1^{\lambda})$ with non-negligible probability.

From selective to full security. We remark that it is possible to convert a signature scheme satisfying selective unforgeability into one that satisfies (full) unforgeability by the standard complexity leveraging arguments (as used in, e.g., Identity-Based [BB04] and Attribute-Based [GVW13] encryption schemes). The reduction suffers a loss proportional to the length of the tag and the challenge dataset of messages.

3 Preliminaries

Notation. Let PPT denote probabilistic polynomial-time. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q and we represent $\mathbb{Z}_q$ as integers in $(-q/2, q/2]$. We let $\mathbb{Z}_q^{n\times m}$ denote the set of $n \times m$ matrices with entries in $\mathbb{Z}_q$. We use bold capital letters (e.g. $\mathbf{A}$) to denote matrices, bold lowercase letters (e.g. $\mathbf{x}$) to denote vectors. The notation $\mathbf{A}^{\mathsf{T}}$ denotes the transpose of the matrix $\mathbf{A}$.

If $\mathbf{A}_1$ is an $n \times m$ matrix and $\mathbf{A}_2$ is an $n \times m'$ matrix, then $[\mathbf{A}_1 \| \mathbf{A}_2]$ denotes the $n \times (m + m')$ matrix formed by concatenating $\mathbf{A}_1$ and $\mathbf{A}_2$. A similar notation applies to vectors. When doing matrix-vector multiplication we always view vectors as column vectors.

We say a function $f(n)$ is *negligible* if it is $O(n^{-c})$ for all $c > 0$, and we use $\mathrm{negl}(n)$ to denote a negligible function of n. We say $f(n)$ is *polynomial* if it is $O(n^c)$ for some $c > 0$, and we use $\mathrm{poly}(n)$ to denote a polynomial function of n. We say an event occurs with *overwhelming probability* if its probability is $1 - \mathrm{negl}(n)$. The function $\lg x$ is the base 2 logarithm of x. The notation $\lfloor x \rceil$ denotes the nearest integer to x, rounding towards 0 for half-integers.

Randomness Extraction We will use the following lemma, which is a generalization of the leftover hash lemma due to Dodis et al. [DRS04]. Agrawal, Boneh and Boyen [ABB10b] proved the lemma for prime moduli q and Agrawal, Freeman and Vaikuntanathan [AFV11] observed that it holds for any square-free q.

Lemma 3.1 ([ABB10b, Lemma 4]). *Suppose that $m > (n+1)\lg q + \omega(\log n)$ that that $q > 2$ is square free. 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 matrices chosen uniformly at random in $\mathbb{Z}_q^{n \times m}, \mathbb{Z}_q^{n \times k}$ respectively. Then, for all vectors $\mathbf{w} \in \mathbb{Z}^m$, the distribution $(\mathbf{A}, \mathbf{AR}, \mathbf{R}^T\mathbf{w})$ is statistically close to distribution $(\mathbf{A}, \mathbf{B}, \mathbf{R}^T\mathbf{w})$.*

Norm of a Random Matrix Let S^m denote the m-sphere: i.e. the set of vectors in $\mathbb{R}^{m+1}$ of length 1. We define the norm of a matrix $\mathbf{R} \in \mathbb{R}^{m \times m}$ to be $\sup_{\mathbf{x} \in S^{m-1}} ||\mathbf{Rx}||$.

Lemma 3.2 ([LPRTJ05, Fact 2.4], [ABB10b, Lemma 5]). *Let $\mathbf{R} \in \{-1, 1\}^{m \times m}$ be chosen at random. Then, $\Pr[||\mathbf{R}|| > 12\sqrt{2m}] < e^{-2m}$.*

3.1 Trapdoors for Lattices and SIS Problem

Gaussian distributions. Let $D_{\mathbb{Z}^m,\sigma}$ be the truncated discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter σ, that is, we replace the output by $\mathbf{0}$ whenever the $||\cdot||_\infty$ norm exceeds $\sqrt{m} \cdot \sigma$. Note that $D_{\mathbb{Z}^m,\sigma}$ is $\sqrt{m} \cdot \sigma$-bounded.

Lemma 3.3 (Lattice Trapdoors [Ajt99, GPV08, MP12]). *There is an efficient randomized algorithm $\mathsf{TrapSamp}(1^n, 1^m, q)$ that, given any integers $n \geq 1$, $q \geq 2$, and sufficiently large $m = \Omega(n \log q)$, outputs a parity check matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a 'trapdoor' matrix $\mathbf{T} \in \mathbb{Z}^{m \times m}$ such that the distribution of $\mathbf{A}$ is $\mathrm{negl}(n)$-close to uniform.*

There is an efficient algorithm $\mathsf{NearestPlane}$ that takes as input a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ together with its trapdoor $\mathbf{T} \in \mathbb{Z}^{m \times m}$, and a matrix $\mathbf{U} \leftarrow \mathbb{Z}_q^{n \times m}$ and outputs a matrix $\mathbf{R} \in \mathbb{Z}^{m \times m}$ such that: (a) $\mathbf{AR} = \mathbf{U}$; and (b) $||\mathbf{R}|| \leq ||\mathbf{T}|| \cdot O(m)$.

3.2 Sampling algorithms

We will use the following algorithms to sample short vectors from specific lattices. Looking ahead, the algorithm $\mathsf{SampleLeft}$ [ABB10b, CHKP12] will be used to sample keys in the real system, while the algorithm $\mathsf{SampleRight}$ [ABB10b] will be used to sample keys in the simulation.

Algorithm $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \mathbf{u}, \alpha)$**:**

Inputs: a full rank matrix $\mathbf{A}$ in $\mathbb{Z}_q^{n\times m}$, a "short" basis $\mathbf{T_A}$ of $\Lambda_q^{\perp}(\mathbf{A})$, a matrix $\mathbf{B}$ in $\mathbb{Z}_q^{n\times m_1}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a Gaussian parameter α. (1)

Output: Let $\mathbf{F} := (\mathbf{A} \parallel \mathbf{B})$. The algorithm outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+m_1}$ in the coset $\Lambda_{\mathbf{F}+\mathbf{u}}$.

Theorem 3.4 ([ABB10b, Theorem 17], [CHKP12, Lemma 3.2])**.** *Let $q > 2$, $m > n$ and $\alpha > \|\widetilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log(m+m_1)})$. Then* $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \mathbf{u}, \alpha)$ *taking inputs as in (1) outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+m_1}$ distributed statistically close to $D_{\Lambda_{\mathbf{F}+\mathbf{u}},\alpha}$, where $\mathbf{F} := (\mathbf{A} \parallel \mathbf{B})$.*

Algorithm $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \mathbf{u}, \alpha)$**:**

Inputs: matrices $\mathbf{A}$ in $\mathbb{Z}_q^{n\times k}$ and $\mathbf{R}$ in $\mathbb{Z}^{k\times m}$, a full rank matrix $\mathbf{B}$ in $\mathbb{Z}_q^{n\times m}$, a "short" basis $\mathbf{T_B}$ of $\Lambda_q^{\perp}(\mathbf{B})$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a Gaussian parameter α. (2)
Output: Let $\mathbf{F} := (\mathbf{A} \parallel \mathbf{AR} + \mathbf{B})$. The algorithm outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+k}$ in the coset $\Lambda_{\mathbf{F}+\mathbf{u}}$.

Often the matrix $\mathbf{R}$ given to the algorithm as input will be a random matrix in $\{1,-1\}^{m\times m}$. Let S^m be the m-sphere $\{\mathbf{x} \in \mathbb{R}^{m+1} : \|\mathbf{x}\| = 1\}$. We define $s_R := \|\mathbf{R}\| := \sup_{\mathbf{x}\in S^{m-1}} \|\mathbf{R}\cdot\mathbf{x}\|$.

Theorem 3.5 ([ABB10b, Theorem 19])**.** *Let $q > 2, m > n$ and $\alpha > \|\widetilde{\mathbf{T_B}}\| \cdot s_R \cdot \omega(\sqrt{\log m})$. Then* $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \mathbf{u}, \alpha)$ *taking inputs as in (2) outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+k}$ distributed statistically close to $D_{\Lambda_{\mathbf{F}+\mathbf{u}},\alpha}$, where $\mathbf{F} := (\mathbf{A} \parallel \mathbf{AR} + \mathbf{B})$.*

Definition 3.1 ([MR07])**.** *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 ($\mathsf{SIS}_{q,m,\beta}$) is: given an integer q, a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ chosen uniformly at random and a real β, find a non-zero integer vector $\mathbf{z} \in \mathbb{Z}^m \backslash \{\mathbf{0}\}$ such that $\mathbf{Az} = \mathbf{0} \mod q$ and $||\mathbf{z}|| \le \beta$.*

Micciancio and Regev [MR07] showed that solving the average-case $\mathsf{SIS}_{q,m,\beta}$ problem for certain parameters is equivalent to solving worst-case instances on hard lattice problems.

4 Our Fully Homomorphic Signature Scheme

Circuit Representation. Let $\mathcal{C}_\lambda$ be a collection of Boolean circuits each having at most $\ell = \ell(\lambda)$ input wires and one output wire. For each $C \in \mathcal{C}_\lambda$, we index the wires of C in the following way. The input wires are indexed 1 to ℓ, the internal wires have indices $\ell+1, \ell+2, \ldots, |C|-1$ and the output wire has index $|C|$, which also denotes the size of the circuit. We assume that the circuit is composed of NAND gates. Each gate g is indexed as a tuple (u, v, w) where u and v are the incoming wire indices, and $w > \max(u, v)$ is the outgoing wire index. The "fan-out wires" in the circuit are given a single number. That is, if the outgoing wire of a gate feeds into the input of multiple gates, then all these wires are indexed the same. (See e.g. [BHR12, Fig 4].)

4.1 Our Construction

Our $\mathcal{C}$-homomorphic signature scheme $\mathcal{HS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Verify})$ works as follows.

- $\mathsf{Setup}(1^\lambda, 1^\ell, 1^{d_{\max}})$: The key-generation algorithm takes the security parameter, the maximum number of inputs in the circuit family and the maximum depth $d_{\max}$, and proceeds as follows.

  1. Set the parameters $n = n(\lambda, d_{\max}), q = q(n, d_{\max}), m = m(n, d_{\max})$. Let $s_1 = s_1(n)$ and $s_2 = s_2(n)$ denote Gaussian parameters and let $B = B(n, d_{\max}) \in \mathbb{Z}$ denote an upper bound on the size of signatures. (See Section 4.3 for details on how to choose these parameters). These parameters are implicitly known to all of the algorithms below.
  2. Sample $\ell + 1$ random matrices: $\mathbf{A}, \{\mathbf{D}_i\}_{i\in[\ell]}$ in $\mathbb{Z}_q^{n\times m}$.
  3. Sample two matrices with associated trapdoors:
  $$(\mathbf{A}^*, \mathbf{T}_{\mathbf{A}^*}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q) \text{ and } (\mathbf{B}, \mathbf{T}_{\mathbf{B}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$$
  4. Output the public key $\mathsf{pk} = (\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \mathbf{T}_{\mathbf{B}}, \{\mathbf{D}_i\}_{i\in[\ell]})$ and the secret signing key $\mathbf{T}_{\mathbf{A}^*}$.

- $\mathsf{Sign}(\mathsf{sk}, t, i, \mu)$: The signing algorithm takes the secret key sk, the tag of the signing dataset t, the message index i and the message μ, and proceeds as follows.

  1. Let $\mathbf{A}_t := [\mathbf{A}^*|\mathbf{A} + t\mathbf{B}]$ denote the "dataset lattice".
  2. Let $[\mathbf{R}_2|\mathbf{R}_1] \leftarrow \mathsf{SampleLeft}(\mathbf{A}^*, \mathbf{A} + t\mathbf{B}, \mathbf{T}_{\mathbf{A}^*}, \mathbf{D}_i + \mu\mathbf{B}, s_2)$ such that:
  $$[\mathbf{A}^*|\mathbf{A} + t\mathbf{B}]\begin{bmatrix}\mathbf{R}_2\\ \hline \mathbf{R}_1\end{bmatrix} = \mathbf{A}^*\mathbf{R}_2 + (\mathbf{A} + t\mathbf{B})\mathbf{R}_1 = \mathbf{D}_i + \mu\mathbf{B}$$
  3. Output the signature $\sigma = [\mathbf{R}_2|\mathbf{R}_1]$.

- $\mathsf{Eval}(\mathsf{pk}, t, \vec{\mu}, \vec{\sigma}, C)$: The evaluation algorithm takes the public key pk, the dataset tag t, the set of messages $\vec{\mu}$, signatures $\vec{\sigma}$ and the circuit $C \in \mathcal{C}$. It computes a homomorphic signature recursively gate by gate, input to output as follows:

  1. For each wire i, let $\mathbf{D}_i \in \mathbb{Z}_q^{n\times m}$ be the "public key" associated with that wire. (Such matrices for the input wires are fixed by the Setup algorithm). Let $\mathbf{A}_t := [\mathbf{A}^*|\mathbf{A} + t\mathbf{B}] \in \mathbb{Z}_q^{n\times 2m}$ denote the "dataset lattice".
  2. Let $g = (u, v, w)$ be a NAND gate carrying input values x, y. By induction, we have $(\mathbf{R}_u, \mathbf{R}_v) \in \mathbb{Z}^{2m\times m}$ such that:
  $$\mathbf{A}_t\mathbf{R}_u = \mathbf{D}_u + x\mathbf{B} \text{ and } \mathbf{A}_t\mathbf{R}_v = \mathbf{D}_v + y\mathbf{B}$$
  3. Define the public key associated with the output wire as $\mathbf{D}_w := \mathbf{D}_v\widetilde{\mathbf{D}}_u - \mathbf{B}$ where
  $$\widetilde{\mathbf{D}}_u \leftarrow \mathsf{SampleD}(\mathbf{B}, \mathbf{T}_{\mathbf{B}}, \mathbf{D}, s_1) \in \mathbb{Z}^{m\times m}$$
  Recall that this means $\mathbf{B}\widetilde{\mathbf{D}}_u = \mathbf{D}_u$.
  4. Compute the homomorphic signature:
  $$\mathbf{R}_w = \mathbf{R}_v \cdot \widetilde{\mathbf{D}}_u - y\mathbf{R}_u$$

5. Finally, output $\mathbf{R}_{|C|} \in \mathbb{Z}^{2m \times m}$ as the homomorphic signature.

- $\mathsf{Verify}(\mathsf{pk}, t, \mu, \sigma, C)$: The verification algorithm takes as input the public key pk, message μ, signature σ and a circuit C. It accepts only if the following conditions are satisfied:

  1. Parse $\sigma = \mathbf{R} \in \mathbb{Z}^{2m \times m}$ and verify $||\mathbf{R}||_\infty \leq B$.
  2. Let $\mathbf{A}_t := [\mathbf{A}^* | \mathbf{A} + t\mathbf{B}] \in \mathbb{Z}_q^{n \times 2m}$ and let $\mathbf{D}_C$ denote the public key associated with the circuit C as computed by the evaluation algorithm defined above. Verify that

$$\mathbf{A}_t \mathbf{R} = \mathbf{D}_C + \mu \mathbf{B} \mod q$$

4.2 Correctness

We show that for the choice of parameters specified above, the verification algorithm accepts an honestly computed homomorphic signature. Consider a particular dataset t of of messages $\{\mu_i\}_{i \in [\ell]}$, signatures $\{\sigma_i\}_{i \in [\ell]}$ and an evaluation of circuit C. Let $\mathbf{A}_t$ be the dataset lattice, $\mathbf{D}_C$ denote the public key derived with respect to circuit C and $\sigma = \mathbf{R} \in \mathbb{Z}^{2m \times m}$ be the homomorphically computed signature, as per the scheme description. We show that:

$$\mathbf{A}_t \cdot \mathbf{R} = \mathbf{D}_C + C(\vec{\mu})\mathbf{B}$$

We start with an inductive claim. Consider a NAND gate $g = (u, v, w)$ carrying input values x, y, respectively.

Claim 4.0.1. *Let $\mathbf{R}_u, \mathbf{R}_v$ be signatures of values x, y respectively under public keys $\mathbf{D}_u, \mathbf{D}_v$ such that $\mathbf{A}_t \mathbf{R}_u = \mathbf{D}_u + x\mathbf{B}$ and $\mathbf{A}_t \mathbf{R}_v = \mathbf{D}_v + y\mathbf{B}$. Let $\mathbf{R}_w = \mathbf{R}_v \cdot \widetilde{\mathbf{D}}_u - y\mathbf{R}_u$ and $\mathbf{D}_w := \mathbf{D}_v \cdot \widetilde{\mathbf{D}}_u - \mathbf{B}$. Then,*

$$\mathbf{A}_t \cdot \mathbf{R}_w = \mathbf{D}_w + (x \text{ NAND } y)\mathbf{B}$$

Furthermore, $||\mathbf{R}_w|| \leq ||\mathbf{R}_v|| \cdot \mathsf{poly}(m) + ||\mathbf{R}_u||$.

Proof.

$$\begin{aligned}
\mathbf{A}_t \cdot \mathbf{R}_w &= \mathbf{A}_t \cdot \left(\mathbf{R}_v \cdot \widetilde{\mathbf{D}}_u - y\mathbf{R}_u\right) \\
&= \mathbf{A}_t \cdot \mathbf{R}_v \cdot \widetilde{\mathbf{D}}_u - y\mathbf{A}_t \cdot \mathbf{R}_u \\
&= (\mathbf{D}_v + y\mathbf{B}) \cdot \widetilde{\mathbf{D}}_u - y(\mathbf{D}_u + x\mathbf{B}) \\
&= \mathbf{D}_v \cdot \widetilde{\mathbf{D}}_u + y\mathbf{D}_u - y\mathbf{D}_u - xy\mathbf{B} \\
&= \mathbf{D}_v \cdot \widetilde{\mathbf{D}}_u - xy\mathbf{B} \\
&= \mathbf{D}_w + (1 - xy)\mathbf{B} = \mathbf{D}_w + (x \text{ NAND } y)\mathbf{B}
\end{aligned}$$

The bound on the size of $\mathbf{R}_w$ follows from the fact that $||\widetilde{\mathbf{D}}_u|| \leq \mathrm{poly}(m)$. □

By induction, from Claim 4.0.1 it follows that for the output signature $\sigma = \mathbf{R}$ we have that

$$\mathbf{A}_w \cdot \mathbf{R} = \mathbf{D}_C + C(\vec{\mu})\mathbf{B}$$

In addition, the size of the signature is $||\mathbf{R}||_\infty \leq B = m^{O(d_{\max})}$.

4.3 Parameter Selection

Let λ denote the security parameter. We set the lattice parameter $n = \text{poly}(\lambda)$. We set the size of the bound on the signature to $B = \omega(2^{d_{\max}})$. Now, we set the modulus $q = q(n) = n^{O(d_{\max})} > B$ and $m = O(n \log q)$. We set the Gaussian parameters $s_1 = O(\sqrt{n \log q})$. To use the indistinguishability of the outputs of SampleLeft and SampleRight algorithms we set $s_1 = \omega(m \log q \sqrt{\log m})$. To satisfy correctness, we now need to establish the maximum size of the homomorphic signatures. By definition of SampleLeft algorithm the signature size is $||\sigma||_\infty \leq s_2\sqrt{m}$. Now for any two signatures σ_u, σ_v as input to gate $g = (u, v, w)$ the output signature is computed as:

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

where $||\widetilde{\mathbf{D}}_u||_\infty \leq s_1\sqrt{m}$. Hence, the size of the output signature is bounded by $O(s_1 s_2 m^3)$. And more generally, for depth $d_{\max}$ circuit, the size of the output signature is bounded by $O(s_1 s_2 m^{O(d_{\max})})$.

4.4 Security Proof

Assume there exists and adversary $\mathcal{A}$ that wins the selective-unforgeability security game 2.3. We construct an adversary $\mathcal{A}^*$ that breaks the $\mathsf{SIS}_{q,m,\beta}$ problem for lattice defined by the basis $\mathbf{A}^* \in \mathbb{Z}_q^{n \times m}$, where q, m as as in the scheme above and we specify β later. $\mathcal{A}^*$ will run the simulated experiments $(\mathsf{Setup}^*, \mathsf{Sign}^*)$. Let $(t^*, \vec{\mu}^*)$ be the challenge tag and the vector of messages (which could be empty) on which the adversary $\mathcal{A}$ will make a forgery. If $\vec{\mu}^* = \perp$, this corresponds to a type (a) forgery: that is, the adversary will never request any signatures for t^*. Otherwise, it corresponds to a type (b) forgery. Let q be an upper bound on the number of signing queries, and let $v^* \in [q]$ be chosen at random (representing a query guess on which the adversary will ask for signatures for the challenge dataset $\vec{\mu}^*$ on which it will make a type (b) forgery).

- $\mathsf{Setup}^*(1^\lambda, \ell, d_{\max})$:

  1. Set the parameters $n = n(\lambda, d_{\max}), q = q(n, d_{\max}), m = m(n, d_{\max})$. Let $s_1 = s_1(n), s_2 = s_2(n)$ denote the Gaussian parameters. Let $\beta = \beta(n, d_{\max})$ denote the upper bound on the size of signatures. (See Section 4.3). These parameters are implicitly known to all of the algorithms below.
  2. Sample a matrix with associated trapdoor:

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

  3. Sample $\mathbf{U} \in \{-1, 1\}^{m \times m}$ at random and let $\mathbf{A} = \mathbf{A}^*\mathbf{U} - t^*\mathbf{B} \mod q$.
  4. **Type (a) forgery**: If $\vec{\mu}^* = \perp$, then for all i, choose matrix $\mathbf{U}_i \in \{-1, 1\}^{m \times m}$ at random and set

  $$\mathbf{D}_i = \mathbf{A}^*\mathbf{U}_i$$

  Type (b) forgery: Otherwise, sample $[\mathbf{R}_{i,2}|\mathbf{R}_{i,1}]^T \leftarrow (D_{\mathbb{Z}^m, s_2})^{2m}$ and set

  $$\mathbf{D}_i = [\mathbf{A}^*|\mathbf{A}^*\mathbf{U}]\begin{bmatrix}\mathbf{R}_{i,2} \\ \hline \mathbf{R}_{i,1}\end{bmatrix} - \mu_i^*\mathbf{B}$$

  Note that, $\mathbf{R}_i = [\mathbf{R}_{i,2}|\mathbf{R}_{i,1}]^T$ corresponds to a signature σ_i of message μ_i^* for tag t^* since $\mathbf{A}_t = [\mathbf{A}^*|\mathbf{A} + t^*\mathbf{B}] = [\mathbf{A}^*|\mathbf{A}^*\mathbf{U}]$ and therefore,

  $$\mathbf{A}_t \cdot \mathbf{R}_i = \mathbf{D}_i + \mu_i^*\mathbf{B}$$

  as required.

5. Initialize a query counter $v = 0$.
6. Output the public key $\mathsf{pk} = (\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \mathbf{T_B}, \{\mathbf{D}_i\}_{i\in[\ell]})$.

- $\mathsf{Sign}^*(\vec{\mu})$:
  1. If the query counter $v \neq v^*$ or $\vec{\mu}^* = \perp$ (indicating that adversary will make a **type (a) forgery**), then:
     (a) Choose a random tag t. Abort the simulation if $t = t^*$ (which happens only with negligible probability).
     (b) For all $i \in [\ell]$, compute the signature $\sigma_i = [\mathbf{R}_2|\mathbf{R}_1]^T$, where

$$[\mathbf{R}_2|\mathbf{R}_1]^T \leftarrow \mathsf{SampleRight}\big(\mathbf{A}^*, (t-t^*)\mathbf{B}, \mathbf{U}, \mathbf{T_B}, \mathbf{D}_i + \mu_i\mathbf{B}, s_2\big)$$

     By the correctness of $\mathsf{SampleRight}$, we have:

$$[\mathbf{A}^*|\mathbf{A}+t\mathbf{B}]\begin{bmatrix}\mathbf{R}_2\\ \hline \mathbf{R}_1\end{bmatrix} = [\mathbf{A}^*|\mathbf{A}^*\mathbf{U}+(t-t^*)\mathbf{B}]\begin{bmatrix}\mathbf{R}_2\\ \hline \mathbf{R}_1\end{bmatrix} = \mathbf{D}_i + \mu\mathbf{B}$$

     (c) Update the query counter v and output the set of signatures $\vec{\sigma} = \{\sigma_i\}_{i\in[\ell]}$.
  2. Otherwise, if $\vec{\mu} \neq \vec{\mu}^*$ then abort (indicating wrong query guess for the challenge dataset). Else, output the set of signatures $\vec{\sigma} = \{\sigma_i = [\mathbf{R}_{i,2}|\mathbf{R}_{i,1}]^T\}$ chosen at the setup phase.

From Lemma 4.1, the output of the above experiments is statistically indistinguishable from the real game. Therefore, the winning adversary outputs a forgery (σ^*, μ^*, C^*). To generate the SIS solution for $\mathbf{A}^* \in \mathbb{Z}_q^{n\times m}$ we distinguish between the two cases:

- **Type (a) forgery**: In this case, the adversary never asked to sign any messages corresponding to the challenge tag t^*. However,

$$[\mathbf{A}^*|\mathbf{A}^*\mathbf{U}]\begin{bmatrix}\mathbf{R}_2\\ \hline \mathbf{R}_1\end{bmatrix} = \mathbf{D}_C + \mu^*\mathbf{B}$$

  where $\mathbf{D}_C$ is derived from $\{\mathbf{D}_i\}$ and C as per the evaluation algorithm. In Lemma 4.2, we show that $\mathbf{D}_C = \mathbf{A}^*\mathbf{U}_C + k\mathbf{B}$ for some small norm matrix $\mathbf{U}_C$ and integer k. Hence, we have

$$\mathbf{A}^*\mathbf{R}_2 + \mathbf{A}^*\mathbf{U}\mathbf{R}_1 = \mathbf{A}^*\mathbf{U}_C + (k+\mu^*)\mathbf{B}$$

  Rearranging, we obtain:

$$\mathbf{A}^*\big(\mathbf{R}_2 + \mathbf{U}\mathbf{R}_1 - \mathbf{U}_C\big) = (k+\mu^*)\mathbf{B}$$

  If $k + \mu^* = 0$ (which can only happen if the adversary makes a forgery for an input wire for message $\mu = 0$), then output $(\mathbf{R}_2 + \mathbf{U}\mathbf{R}_1 - \mathbf{U}_C)$ as the SIS solution. Otherwise, output $(\mathbf{R}_2 + \mathbf{U}\mathbf{R}_1 - \mathbf{U}_C)\mathbf{T_B}$ as the solution. Note that in type (a) forgery, the adversary gets no information about $\mathbf{U}_C$ since all signatures are generated completely independently by $\mathsf{SampleRight}$ algorithm. Hence, with all but negligible probability the adversary outputs $[\mathbf{R}_2|\mathbf{R}_1]^T$ such that $(\mathbf{R}_2 + \mathbf{U}\mathbf{R}_1) \neq \mathbf{U}_C$. Therefore, the output is a non-zero SIS solution.

- **Type (b) forgery**: In this case, the adversary requested a set of signatures $\vec{\sigma}^* = \{\sigma_i = [\mathbf{R}_2^i|\mathbf{R}_1^i]^T\}$ for the dataset $\vec{\mu}^*$. Its output is $(\sigma^* = [\mathbf{R}_2^*|\mathbf{R}_1^*]^T, \mu^*, C^*)$ such that $C(\vec{\mu}^*) \neq \mu^*$. Let $\sigma' = [\mathbf{R}_2'|\mathbf{R}_1']^T \leftarrow \mathsf{Eval}(\mathsf{pk}, t^*, \vec{\mu}^*, \vec{\sigma}^*, C^*)$ be computed honestly. Then, we have

$$\begin{aligned} \mathbf{A}_t \left[\frac{\mathbf{R}_2'}{\mathbf{R}_1'}\right] &= \mathbf{D}_C + C^*(\vec{\mu}^*)\mathbf{B} \text{ and} \\ \mathbf{A}_t \left[\frac{\mathbf{R}_2^*}{\mathbf{R}_1^*}\right] &= \mathbf{D}_C + \mu^*\mathbf{B} \end{aligned}$$

  Hence,

$$\mathbf{A}_t \left[\frac{\mathbf{R}_2' - \mathbf{R}_2^*}{\mathbf{R}_1' - \mathbf{R}_1^*}\right] = \big(C^*(\vec{\mu}^*) - \mu^*\big)\mathbf{B}$$

  Therefore, $\sigma' - \sigma^*$ is non-zero (otherwise, $C^*(\vec{\mu}^*) = \mu^*$ contradicting valid forgery). Expanding $\mathbf{A}_t$ from its definition, we have

$$\begin{aligned} \mathbf{A}^*(\mathbf{R}_2' - \mathbf{R}_2^*) + \mathbf{A}^*\mathbf{U}(\mathbf{R}_1' - \mathbf{R}_1^*) &= \big(C^*(\vec{\mu}^*) - \mu^*\big)\mathbf{B} \iff \\ \mathbf{A}^*\big((\mathbf{R}_2' - \mathbf{R}_2^*) + \mathbf{U}(\mathbf{R}_1' - \mathbf{R}_1^*)\big) &= \big(C^*(\vec{\mu}^*) - \mu^*\big)\mathbf{B} \end{aligned}$$

  Output $\big((\mathbf{R}_2' - \mathbf{R}_2^*) + \mathbf{U}(\mathbf{R}_1' - \mathbf{R}_1^*)\big)\mathbf{T}_\mathbf{B}$ as the SIS solution for lattice $\mathbf{A}^* \in \mathbb{Z}_q^{n\times m}$.

Lemma 4.1. *Let* $(\mathsf{pk}, \{\vec{\sigma}_i\})$ *be the output in the real execution and* $(\mathsf{pk}^*, \{\vec{\sigma}_i\}^*)$ *be the output in the simulated execution by the* Setup, Sign *algorithms respectively. We show that the two distributions are statistically indistinguishable.*

Proof. First, we argue the claim for the **type (a) forgery** simulation. We can summarize the difference in the execution of the algorithms as follows.

- In real Setup, matrix $\mathbf{A}^* \in \mathbb{Z}_q^{n\times m}$ is chosen with the trapdoor by running TrapGen algorithm. In simulated Setup*, $\mathbf{A}^*$ are chosen uniformly at random (by the average-case SIS generator).
- In real Setup, matrices $(\mathbf{A}, \{\mathbf{D}_i\}_{i\in[\ell]})$ are chosen uniformly at random. In simulated Setup*, matrix $\mathbf{A} = \mathbf{A}^*\mathbf{U} - t^*\mathbf{B}$ for a uniformly random $\mathbf{U} \in \{-1, 1\}^{m\times m}$. Also, for all i, $\mathbf{D}_i = \mathbf{A}^*\mathbf{U}_i$ for uniformly random $\mathbf{U}_i \in \{-1, 1\}^{m\times m}$. The public key pk is defined as $(\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \{\mathbf{D}_i\}_{i\in[\ell]}, \mathbf{P})$.
- Finally, the real Sign algorithm generates each vector of signatures $\vec{\sigma}_i$ by using SampleLeft algorithm and a trapdoor for matrix $\mathbf{A}^*$ (on each message independently). Whereas, the simulated Sign* algorithm generates the signatures using SampleRight algorithm and a trapdoor for $\mathbf{B}$.

We now argue that the distribution $\big(\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \{\mathbf{D}_i\}_{i\in[\ell]}, \mathbf{P}, \{\vec{\sigma}_i\}_{i\in[q]}\big)$ is statistically indistinguishable in the two experiments. Observe that by Lemma 3.3, for sufficiently large $m = \Omega(n \log q)$ (see Section 4.3 for all parameter selections), $\mathbf{A}^* \in \mathbb{Z}_q^{n\times m}$ is distributed statistically close to uniform. Now, let $\mathbf{A}' = [\mathbf{A}|\mathbf{D}_1|\ldots|\mathbf{D}_\ell] \in \mathbb{Z}_q^{n\times(\ell+1)m}$. Then, by Lemma 3.1, it follows that

$$\big(\mathbf{A}^*, \mathbf{A}'\big) \overset{s}{\approx} \big(\mathbf{A}^*, [\mathbf{A}^*\mathbf{U} - t^*\mathbf{B}|\mathbf{A}^*\mathbf{U}_1|\ldots|\mathbf{A}^*\mathbf{U}_\ell]\big)$$

for randomly matrices $\mathbf{A}^*, \mathbf{A}'$ from $\mathbb{Z}_q^{n\times m}, \mathbb{Z}_q^{n\times(\ell+1)m}$ and $[\mathbf{U}|\mathbf{U}_1|\ldots|\mathbf{U}_\ell] \in \{-1, 1\}^{m\times(\ell+1)m}$ chosen at random. Tuple $(\mathbf{B}, \mathbf{P})$ is generated identically in both executions. Therefore, we conclude that pk in real is indistinguishable from pk* in simulated experiment.

Now, consider a signing oracle query on dataset $\vec{\mu}$, given the public key. Note that the simulation aborts with negligible probability. Otherwise, let $\mathbf{A}_t = [\mathbf{A}^*|\mathbf{A} + t\mathbf{B}]$ for a randomly chosen tag t. Let $\mathbf{C} = \mathbf{D}_i + \mu_i\mathbf{B} \in \mathbb{Z}_q^{n\times m}$ be the coset defined by message μ_i. By Theorems 3.4, 3.5, for sufficiently large Gaussian parameter s_2 (see Section 4.3), the output of SampleLeft and SampleRight is distributed statistically close to $D_{\Lambda_{\mathbf{A}_t+\mathbf{C}},s_2}$. This proves the claim for **type (a) forgery**.

Next, we argue that the output of the above algorithms is indistinguishable from real for **type (b) simulation**. We start by summarizing the differences in the executions.

- In real Setup, matrix $\mathbf{A}^* \in \mathbb{Z}_q^{n\times m}$ is chosen with the trapdoor by running TrapGen algorithm. In simulated Setup*, $\mathbf{A}^*$ are chosen uniformly at random (by the average-case SIS generator).
- In real Setup, matrices $(\mathbf{A}, \{\mathbf{D}_i\}_{i\in[\ell]})$ are chosen uniformly at random. In simulated Setup*, matrix $\mathbf{A} = \mathbf{A}^*\mathbf{U} - t^*\mathbf{B}$ for a uniformly random $\mathbf{U} \in \{-1,1\}^{m\times m}$. Also, for all $i \in [\ell]$,
$$\mathbf{D}_i = [\mathbf{A}^*|\mathbf{A}^*\mathbf{U}]\begin{bmatrix}\mathbf{R}_{i,2}\\ \mathbf{R}_{i,1}\end{bmatrix} - \mu_i^*\mathbf{B}$$
for $[\mathbf{R}_{i,2}|\mathbf{R}_{i,1}]^T \leftarrow (D_{\mathbb{Z}^m,s_2})^{2m}$. The public key pk is defined as $(\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \{\mathbf{D}_i\}_{i\in[\ell]}, \mathbf{P})$.
- Finally, the real Sign algorithm generates each vector of signatures $\vec{\sigma}_i$ by using SampleLeft algorithm and a trapdoor for matrix $\mathbf{A}^*$ (on each message independently). Whereas, for all tags $t \neq t^*$ the simulated Sign* algorithm generates the signatures using SampleRight algorithm and a trapdoor for $\mathbf{B}$. For tag $t = t^*$ the simulator outputs the set of signatures $\vec{\sigma} = \{\sigma_i = [\mathbf{R}_{i,2}|\mathbf{R}_{i,1}]^T$ chosen at the setup.

We now argue that the distribution $\left(\mathbf{A}, \mathbf{A}^*, \mathbf{B}, \{\mathbf{D}_i\}_{i\in[\ell]}, \mathbf{P}, \{\vec{\sigma}_i\}_{i\in[q]}\right)$ is statistically indistinguishable in the two experiments. Observe that by Lemma 3.3, for sufficiently large $m = \Omega(n \log q)$ (see Section 4.3 for all parameter selections), $\mathbf{A}^* \in \mathbb{Z}_q^{n\times m}$ is distributed statistically close to uniform. Now, let $\mathbf{A}' = [\mathbf{A}|\mathbf{D}_1|\ldots|\mathbf{D}_\ell] \in \mathbb{Z}_q^{n\times(\ell+1)m}$. Then, from Lemma 3.1 and Lemma 3.3, it follows that

$$\left(\mathbf{A}^*, \mathbf{A}'\right) \overset{s}{\approx} \left(\mathbf{A}^*, [\mathbf{A}^*\mathbf{U} - t^*\mathbf{B}|\mathbf{A}^*\mathbf{R}_{1,2} + \mathbf{A}^*\mathbf{U}\mathbf{R}_{1,1} - \mu_1^*\mathbf{B}|\ldots|\mathbf{A}^*\mathbf{R}_{\ell,2} + \mathbf{A}^*\mathbf{U}\mathbf{R}_{\ell,1} - \mu_\ell^*\mathbf{B}]\right)$$

for randomly matrices $\mathbf{A}^*, \mathbf{A}'$ from $\mathbb{Z}_q^{n\times m}, \mathbb{Z}_q^{n\times(\ell+1)m}$. Tuple $(\mathbf{B}, \mathbf{P})$ is generated identically in both executions. Therefore, we conclude that pk in real is indistinguishable from pk* in simulated experiment.

Now, consider a signing oracle query on dataset $\vec{\mu}$, given the public key. If $\vec{\mu} \neq \vec{\mu}^*$ then the signatures are generated using the trapdoor for $\mathbf{B}$. From Theorems 3.4 and 3.5 we conclude that the output is distributed statistically close. In addition, without the public key, the signatures on the challenge dataset are distributed statistically indistinguishable from real by Theorem 3.4. Put together, the signatures and the public key are indistinguishable from real. This proves the claim for **type (b) forgery**.

□

Lemma 4.2. *Let C be an arbitrary circuit taking at most ℓ bit input and outputting one bit. Let $\mathbf{A}^*, \mathbf{B} \in \mathbb{Z}_q^{n\times m}$. Fix the public keys for all input wires as $\mathbf{D}_i = \mathbf{A}^*\mathbf{U}_i$ for $\mathbf{U}_i \in \{-1,1\}^{m\times m}$ for all $i \in [\ell]$. For each gate $g = (u, v, w)$ assume input public keys $\mathbf{D}_u, \mathbf{D}_v$ are fixed and let*

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

where $\mathbf{P}$ is of low norm such that $\mathbf{BP} = \mathsf{P2}(\mathbf{I}_n) \in \mathbb{Z}_q^{n\times m}$. 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 $k \in \mathbb{Z}$ (in fact, $k \in \{0,1\}$).

Proof. We proceed by the induction on the circuit depth: for all wires w, the public key is of the form $\mathbf{D}_w = \mathbf{A}^*\mathbf{U}_i + i_w\mathbf{B}$ where $i_w \in \mathbb{Z}$. Clearly, the base case holds as for all input wires, $\mathbf{D}_i = \mathbf{A}^*\mathbf{U}_i + 0\mathbf{B}$. Now, consider a gate $g = (u, v, w)$ where $\mathbf{D}_u = \mathbf{A}^*\mathbf{U}_u + i_u\mathbf{B}$ and $\mathbf{D}_v = \mathbf{A}^*\mathbf{U}_v + i_v\mathbf{B}$ for integers i_u, i_v. Then,

$$\begin{aligned}
\mathbf{D}_w &:= \mathbf{D}_v\mathbf{P}\mathsf{BD}(\mathbf{D}_u) - \mathbf{B} \\
&= \big(\mathbf{A}^*\mathbf{U}_v + i_v\mathbf{B}\big)\mathbf{P}\mathsf{BD}(\mathbf{D}_u) - \mathbf{B} \\
&= \mathbf{A}^*\mathbf{U}_v\mathbf{P}\mathsf{BD}(\mathbf{D}_u) + i_v\mathbf{B}\mathbf{P}\mathsf{BD}(\mathbf{D}_u) - \mathbf{B} \\
&= \mathbf{A}^*\mathbf{U}_v\mathbf{P}\mathsf{BD}(\mathbf{D}_u) + i_v\mathsf{P2}(\mathbf{I}_n)\mathsf{BD}(\mathbf{D}_u) - \mathbf{B} \\
&= \mathbf{A}^*\mathbf{U}_v\mathbf{P}\mathsf{BD}(\mathbf{D}_u) + i_v\mathbf{D}_u - \mathbf{B} \\
&= \mathbf{A}^*\mathbf{U}_v\mathbf{P}\mathsf{BD}(\mathbf{D}_u) + \mathbf{A}^*\mathbf{U}_u + i_u\mathbf{B} - \mathbf{B} \\
&= \mathbf{A}^*\underbrace{\big(\mathbf{U}_v\mathbf{P}\mathsf{BD}(\mathbf{D}_u) + \mathbf{U}_u\big)}_{\text{small norm}} + i_w\mathbf{B}
\end{aligned}$$

Hence, by induction we obtain that $\mathbf{D}_C = \mathbf{A}^*\mathbf{U}_C + k\mathbf{B}$ for some small norm matrix $\mathbf{U}_C$ as claimed. Similarly, we can calculate the size of the matrix $\mathbf{U}_C$. Initially, for all input wires, $\mathbf{U}_i \in \{-1, 1\}^{m\times m}$. Now, for a gate $g = (u, v, w)$, we have $\mathbf{U}_w = (\mathbf{U}_v\mathbf{P}\mathsf{BD}(\mathbf{D}_u) + \mathbf{U}_u)$. Hence, $||\mathbf{U}_w||_\infty = O(s_1m^3)$. By induction, we obtain that the size of the output matrix $||\mathbf{U}_C||_\infty = O(s_1m^{O(d_{\max})})$.

□

5 Extensions

5.1 Arithmetic Circuits and Larger Message Space

In the construction of Attribute-Based Encryption from fully key-homomorphic encryption, Boneh et al. [BGG+14] show how to support arithmetic circuits where the values on wires can come from a large field. We note that the same technique of using deterministic Babai's algorithm can be used in our construction to support arithmetic circuits with weighted gates. More formally, first we add the trapdoor for $\mathbf{B}$ in the public parameters (this does not affect the security since in our proof, the matrix $\mathbf{B}$ and its trapdoor are known by the simulated). Now, say the user holds two signatures $\mathbf{R}_u$ and $\mathbf{R}_v$ corresponding to wires values x and y ($\in \mathbb{Z}_q$), respectively, satisfying:

$$\mathbf{A}_t \cdot \mathbf{R}_u = \mathbf{D}_u + x\mathbf{B} \text{ and } \mathbf{A}_t \cdot \mathbf{R}_v = \mathbf{D}_v + y\mathbf{B}$$

Consider an addition gate $g = \alpha_1 x + \alpha_2 y$ for constants α_1, α_2. First, the user computes $\mathbf{W}_u$ (using deterministic Babai's algorithm [Bab86]) such that $\mathbf{B} \cdot \mathbf{W}_u = \alpha_1\mathbf{B}$. Similarly, the user computes $\mathbf{W}_v$ such that $\mathbf{B} \cdot \mathbf{W}_v = \alpha_2\mathbf{B}$. At last, the user compute the homomorphic signature as $\mathbf{R}_w = \mathbf{R}_u \cdot \mathbf{W}_u + \mathbf{R}_v \cdot \mathbf{W}_v$. Hence,

$$\begin{aligned}
\mathbf{A}_t \cdot (\mathbf{R}_u \cdot \mathbf{W}_u + \mathbf{R}_v \cdot \mathbf{W}_v) &= (\mathbf{D}_u + x\mathbf{B}) \cdot \mathbf{W}_u + (\mathbf{D}_v + y\mathbf{B}) \cdot \mathbf{W}_v \\
&= (\mathbf{D}_u \cdot \mathbf{W}_u + \mathbf{D}_v \cdot \mathbf{W}_v) + (\alpha_1 x + \alpha_2 y)\mathbf{B}
\end{aligned}$$

And hence we set the public key for the output wire as $(\mathbf{D}_u \cdot \mathbf{W}_u + \mathbf{D}_v \cdot \mathbf{W}_v)$. It is important to note that the public key is deterministically derived with respect to the circuit, and does not depend on the wire values x or y (it depends α_1, α_2 which are circuit specifications). Homomorphic signatures for weighted multiplication can be computed similarly.

5.2 Extendability

Our main scheme presented in Section 4 is defined with respect to a circuit family with a fixed number of inputs ℓ. We can extend the construction to a family with *unbounded* number of inputs, but in the random oracle model. That is, instead of fixing the public keys $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$ for all input wires, the public parameters specify a description of a hash function $H : \mathbb{Z} \to \mathbb{Z}_q^{m \times n}$. To assign a message μ for an input wire with index i, we first compute the public key for the wire $\mathbf{A}_i = H(i)$ and proceed signing as before. In the random oracle model, the output $\mathbf{A}_i$ is uniformly distributed over $\mathbb{Z}_q^{m \times n}$ and our proof remains virtually identical.

References

[ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (h)ibe in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

[ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*, pages 21–40, 2011.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9, 1999.

[Bab86] L. Babai. On lovsz lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

[BF11a] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 149–168. Springer, 2011.

[BF11b] 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, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2011.

[BFKW09] Dan Boneh, David Mandell Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 68–87. Springer, 2009.

[BGG+14] 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 Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556. Springer, 2014.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security*, pages 784–796, 2012.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[CHKP12] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *J. Cryptology*, 25(4):601–639, 2012.

[CJL09] Denis Xavier Charles, Kamal Jain, and Kristin Lauter. Signatures for network coding. *IJICoT*, 1(1):3–14, 2009.

[CKLR11] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2011.

[CKV10] Kai-Min Chung, Yael Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and JanL. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer Berlin Heidelberg, 2004.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GKKR10] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 142–160. Springer, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 545–554, New York, NY, USA, 2013. ACM.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *STOC*, pages 723–732. ACM, 1992.

[KRR13a] Yael Tauman Kalai, Ran Raz, and Ron Rothblum. How to delegate computations: The power of no-signaling proofs. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:183, 2013.

[KRR13b] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *STOC*, pages 565–574. ACM, 2013.

[LPRTJ05] A.E. Litvak, A. Pajor, M. Rudelson, and N. Tomczak-Jaegermann. Smallest singular value of random matrices and geometry of random polytopes. *Advances in Mathematics*, 195(2):491 – 523, 2005.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718, 2012.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.