Compact Attribute Based Encryption and SIMD-Operations Supported Homomorphic Signatures

Xiong Fan [*] Xavier Boyen [†]

Abstract

We construct the first (key-policy) attribute-based encryption (ABE) system for circuits from the standard Learning With Errors (LWE) assumption, that has *compact* public parameters and ciphertexts. In particular, for up to $O(\log \lambda)$-length attribute strings, the size of public parameters and ciphertexts are independent of attribute length and are thus constant. For $\mathsf{poly}(\lambda)$-length attribute, the size of both components are sublinear in the length of attributes. This improves previous ABE schemes from lattices substantially.

We also construct the first homomorphic signature scheme from the standard Short Integer Solution (SIS) assumption that supports *signature packing* and single-instruction-multiple-data (SIMD) operations on packed signatures. In addition to SIMD operations, we introduce permuting techniques to move signature elements across these packed signature efficiently. Therefore, through our SIMD operations and permutation, we are able to implement circuit evaluation over signatures in a batched manner.

1 Introduction

Encryption has traditionally been regarded as a way to ensure confidentiality of an end-to-end communication. However, with the emergence of complex networks and cloud computing, recently the cryptographic community has been rethinking the notion of encryption to address security concerns that arise in these more complex environments. Attribute-based encryption (ABE) [SW05, GPSW06] is a powerful generalization of identity-based encryption [Sha84, BF01] and a special case of functional encryption [BSW11]. In (key-policy) ABE, every secret key is associated with some function $f : \mathcal{X} \to \mathcal{Y}$ and an encryption of a message μ is associated with a public attribute vector $\boldsymbol{x} \in \mathcal{X}$. The encryption of μ can be decrypted using sk_f if and only if $f(\boldsymbol{x}) = 0$. Intuitively, the security requirement is collusion resistance: a coalition of users learns nothing about the plaintext μ if none of their individual keys are authorized to decrypt the ciphertext. Besides its potential applications in practice, ABE also can be used as a building block in many cryptographic primitives, such as verifiably outsourcing computations [PRV12] and single-use functional encryption [GKP$^+$13].

In recent years, there have been much progress in constructing secure and efficient ABE schemes from different assumptions and supporting different families of functions. The first constructions [GPSW06, LOS$^+$10, OT10, LW12, Wat12, Boy13, HW13, Att14, CGW15, Att16] apply to predicates computable by Boolean formulas which are a subclass of log-space computations. More recently, there has been

[*]Cornell University, Email: `xfan@cs.cornell.edu`. This material is based upon work supported by IBM under Agreement 4915013672. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsors.

[†]Queensland University of Technology. Email: `xb@cs.stanford.edu`. Research funded under ARC Discovery Project grant number DP-140103885 and ARC Future Fellowship grant number FT-140101145 from the Australian Research Council.

important progress in expanding the range of supported families of functions to all polynomial-size circuits [GVW13, BGG$^+$14, BV], Branching Programs [GV15] and Turing Machines [BL16] from well studied assumptions, i.e., Learning With Errors (LWE) [Reg05]. From the above facts, we can say that our understanding about instantiating ABE for different families of functions is quite good. However, for improving the efficiency of existing lattice-based ABE constructions, such as reducing the size of public parameters and ciphertexts, our understanding is limited. The state-of-the-art construction for polynomial-sized circuits [BGG$^+$14] has public parameters and ciphertexts whose size linearly depends on the attribute length. As the main motivation for studying ABE stems from its potential deployment in complex networks and cloud computing, the size of transmitted data is a bottleneck in current constructions. This brings us to the following question:

Question 1: *Can we optimize the size of public parameters and ciphertexts of ABE constructions for polynomial-sized circuits?*

Another application motivated by the prevalence of cloud computing is outsourced computation, where the user can outsource their data to an untrusted remote server, while also allowing the server to perform useful computations over this data. There are two important issues to address in this scenario: *privacy* and *authenticity*. Homomorphic encryption [Gen09] handles the privacy issue, while homomorphic signature [JMSW02] tackles authenticity in the outsourced computation setting. A homomorphic signature (HS) scheme allows a user to sign some large dataset x using his secret signing key. Then he can distribute the signed data σ_x to some untrusted server that can perform arbitrary computations $y = f(x)$ over the data and homomorphically derive a signature $\sigma_{f,y}$ on the result. The derived signature $\sigma_{f,y}$ should be *short*, with length independent of the size of the data x, and the fact that y is the correct output of the computation $f(x)$ can be verified using the tuple $(f, y, \sigma_{f,y})$ and user's public verification key. Intuitively, the security requirement is that the adversary cannot forge new valid signatures except for signatures homomorphically derived from known ones.

Similar as the progress made in ABE construction, there exist HS constructions supporting various operations on the signature from different assumptions, such as linear functions [BFKW09, AL11, BF11b], fixed-degree polynomials [BF11a] and arbitrary circuits [GVW15b, BFS14]. However, the issue of optimizing the size of signatures, or more specifically the *ratio* of signed data size to message size (or their computation complexity), still remains unsolved. In the setting of homomorphic encryption, as shown in some works [SV14, GHS12, HAO15], a similar problem is addressed by packing many plaintexts into one ciphertext and designing a complete set of operations—*Add, Mult* and *Permute*—to operate on it. The newly designed computations operate on a packed ciphertext in a Single-Instruction-Multi-Data (SIMD) manner, and allow the evaluation of arbitrary circuits on encrypted data while keeping the ciphertexts packed. This appealing property of homomorphic encryption significantly improves its efficiency, and is an essential part of the FHE software library HElib [HS14, HS15]. Therefore, in order to improve the efficiency of homomorphic signatures, an intriguingly question is:

Question 2: *Can we construct a homomorphic signature scheme that can pack multiple signatures into one and support SIMD operations of arbitrary circuits over packed signatures?*

1.1 Our Contributions

In this work, we positively answer the two questions above by constructing an attribute-based encryption scheme for circuits whose size of ciphertexts if sublinear in the length of attributes, and a (leveled) homomorphic signature scheme for circuits that supports signature packing and SIMD-style operations. Our results can be summarized in the following informal theorem.

Theorem 1.1 (Informal)**.** *Let λ be the security parameter.*

- *Under the subexponential Learning With Error assumption, there is an ABE scheme for the family of functions with* $\mathsf{poly}(\lambda)$*-depth and (up to)* $\log(q)$*-fan-in circuits, where (1) the prime modulus q and lattice dimension* n, m *are of size polynomial in the security parameter* λ*, (2) the public parameters contain two matrices in* $\mathbb{Z}_q^{n\times m}$ *and one vector in* $\mathbb{Z}_q^n$*, and (3) the ciphertext contains only one vector in* $\mathbb{Z}_q^{2m+1}$*.*
- *Under the subexponential Short Integer Solution assumption, there is a (leveled) HS scheme supporting SIMD-style* $\mathsf{poly}(\lambda)$*-depth operations and message packing up to* $O(\log q)$ *signed messages without increasing the size of public key and signatures, where q here is a prime modulus of size polynomial in the security parameter* λ*.*

Technical Highlight. To demonstrate the intuition of our construction, we first briefly recall two constructions of attribute-based encryption [BGG$^+$14] and homomorphic signatures [GVW15b]. These two schemes can be regarded as applications of GSW-FHE [GSW13] and its simplification [AP14] in different settings. In the GSW FHE scheme. an encryption of x can be expressed as $\mathbf{C}_x = \mathbf{AR} + x\mathbf{G} \in \mathbb{Z}_q^{n\times m}$, where $\mathbf{A}$ is in the public key, $\mathbf{R}$ is a small matrix and $\mathbf{G} = (1, 2, ..., 2^{\lfloor \log q \rfloor}) \otimes \mathbf{I}_n$ is the gadget matrix introduced in [MP12]. The computation with respect to the encryption $\mathbf{C}_x$ can be performed in a surprisingly simple way: $\mathbf{C}_{x+y} = \mathbf{C}_x + \mathbf{C}_y$ and $\mathbf{C}_{xy} = \mathbf{C}_x \cdot \mathbf{G}^{-1}(\mathbf{C}_y)$. Since the $\mathbf{G}^{-1}$ operation produces a small-norm matrix and $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{C}_y) = \mathbf{C}_y$, the ciphertexts of $\mathbf{C}_{x+y}$ and $\mathbf{C}_{xy}$ can be expressed as $\mathbf{AR}_+ + (x+y)\mathbf{G}$ and $\mathbf{AR}_\times + xy\mathbf{G}$ for some small matrices $\mathbf{R}_+$ and $\mathbf{R}_\times$. The computation method here inspires both the constructions of ABE and HS. In an ABE scheme, the encryption of a message μ associated with attribute $\boldsymbol{x} = (x_1, ..., x_\ell) \in \mathbb{Z}_q^\ell$ is dealt in the dual-Regev encryption [Reg05, GPV08] way by using the following matrix as the public key $[\mathbf{A}|\mathbf{D}_1 + x_1\mathbf{G}|\cdots|\mathbf{D}_\ell + x_\ell\mathbf{G}] \in \mathbb{Z}_q^{n\times(\ell+1)m}$, where $\mathbf{A}$ and $\{\mathbf{D}_i\}$ are in public parameters. In the decryption algorithm, we want to evaluate $f(\boldsymbol{x})$ homomorphically for some policy function f. To illustrate how to compute gate-by-gate the function f, assume we have ciphertext for attribute (x, y) under the above public key: $\boldsymbol{c}_{\boldsymbol{x}} = (\boldsymbol{c}_0|\boldsymbol{c}_x|\boldsymbol{c}_y)$, where $\boldsymbol{c}_0 = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{e}, \boldsymbol{c}_x = \boldsymbol{s}^\mathsf{T}(\mathbf{D}_1 + x\mathbf{G}) + \boldsymbol{e}_1$ and $\boldsymbol{c}_y = \boldsymbol{s}^\mathsf{T}(\mathbf{D}_2 + y\mathbf{G}) + \boldsymbol{e}_2$. To compute the addition gate in f, one takes the sum of ciphertexts $\boldsymbol{c}_x$ and $\boldsymbol{c}_y$. The result is the encryption under the evaluated matrix $((\mathbf{D}_1 + \mathbf{D}_2) + (x+y)\mathbf{G})$, where $(\mathbf{D}_1 + \mathbf{D}_2)$ can be viewed as $\mathbf{D}_+$ denoting the addition gate. To compute the multiplication gate in f, one computes $\boldsymbol{c}_y \cdot \mathbf{G}^{-1}(-\mathbf{D}_1) + y \cdot \boldsymbol{c}_x$. The result is the encryption under the evaluated matrix $(\mathbf{D}_2 \cdot \mathbf{G}^{-1}(-\mathbf{D}_1) + xy \cdot \mathbf{G})$, where $\mathbf{D}_2 \cdot \mathbf{G}^{-1}(-\mathbf{D}_1)$ can be viewed as $\mathbf{D}_\times$ denoting the multiplication gate. One nice property here is the matrix $\mathbf{D}_f$ denoting the output gate of function f can be computed from public parameters $\{\mathbf{D}_i\}$ deterministically and independently of its input $\boldsymbol{x}$. Thus, if $f(\boldsymbol{x}) = 0$, then at the end of ciphertext evaluation, the ciphertext $\boldsymbol{c}_{f(\boldsymbol{x})} = \boldsymbol{s}^\mathsf{T}(\mathbf{D}_f + 0 \cdot \mathbf{G}) + \boldsymbol{e}_{(f,\boldsymbol{x})}$. In order to successfully decrypt the ciphertext when $f(\boldsymbol{x}) = 0$, the secret key for function f should be a good trapdoor for (the lattice orthogonal to) $[\mathbf{A}|\mathbf{D}_f]$.

Similar ideas are also employed in the (leveled) homomorphic signature construction of [GVW15b]. The signature for a message x_i is a low-norm matrix $\mathbf{R}_{x_i}$ satisfying $\mathbf{A}_\tau\mathbf{R}_x = \mathbf{D}_i + x_i\mathbf{G}$, where $\mathbf{A}_\tau$ denotes the dataset τ and i is the index of message x_i in this dataset. To evaluate the signature according to a function f gate-by-gate, the same homomorphic computing paradigm is used, i.e., for addition, $\mathbf{R}_{x_i+x_j} = \mathbf{R}_{x_i} + \mathbf{R}_{x_j}$, and $\mathbf{R}_{x_ix_j} = \mathbf{R}_{x_i} \cdot \mathbf{G}^{-1}(-\mathbf{D}_j) + x_j \cdot \mathbf{R}_i$ for multiplication. Since the $\mathbf{G}^{-1}(\cdot)$ operation outputs a low-norm matrix, the evaluated signature $\mathbf{R}_{f(\boldsymbol{x})}$ can still be a valid signature with respect to $\mathbf{D}_f + f(\boldsymbol{x})\mathbf{G}$.

Our approach begins with the observation of the newly proposed vector encoding used in [AFL16]. In that work, a vector encoding is used to optimize the size of the public key in an adaptively secure IBE scheme. let $\boldsymbol{v} = (v_1, ..., v_d) \in \mathbb{Z}_q^d$ be a vector. The encoding of vector $\boldsymbol{v}$ is $\mathbf{E}_{\boldsymbol{v}} = [v_1\mathbf{I}_n|\cdots|v_d\mathbf{I}_n] \cdot \mathbf{G}_{dn,\ell,m}$, where the $dn \times m$-dimensional matrix $\mathbf{G}_{dn,\ell,m}$ is the generalized gadget matrix $\mathbf{G}_{dn,\ell,m} = (1, \ell, ..., \ell^{\lfloor \log_\ell q \rfloor}) \otimes \mathbf{I}_{dn}$ mentioned in [MP12, AFL16]. We note that this vector encoding is versatile to support various kinds of operations. To name a few operations here, let the matrices $\mathbf{E}_{\boldsymbol{v}_1}$ and $\mathbf{E}_{\boldsymbol{v}_2}$ be encodings of vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. Clearly, $\mathbf{E}_{\boldsymbol{v}_1} + \mathbf{E}_{\boldsymbol{v}_2}$ is the encoding of $\boldsymbol{v}_1 + \boldsymbol{v}_2$. For scalar multiplication, i.e., $a \cdot \boldsymbol{v}$, where $a \in \mathbb{Z}_q$, we compute $\mathbf{E}_{a\boldsymbol{v}} = \mathbf{E}_{\boldsymbol{v}} \cdot \mathbf{G}_{dn,\ell,m}^{-1}(a\mathbf{G}_{dn,\ell,m})$. In addition to some more vector operations, such as inner product and coordinate-wise multiplication we describe in detail in Section 3, the encoding also supports *unpacking*, *packing* and *permutation*. For the unpacking operation, given a vector encoding $\mathbf{E}_{\boldsymbol{v}}$ and an index $i \in [d]$, one can unpack the i-th coordinate to extract an encoding of v_i. Similarly, for the packing operation, given encodings of $v_1, ..., v_d$, we can obtain a packed vector encoding $\mathbf{E}_{\boldsymbol{v}}$, where $\boldsymbol{v} = (v_1, ..., v_d)$. For permutation, given an encoding $\mathbf{E}_{\boldsymbol{v}}$ and permutation π on $[d]$, we compute the permuted vector encoding $\mathbf{E}_{\boldsymbol{v}_\pi}$ ($\boldsymbol{v}_\pi = (v_{\pi(1)}, ..., v_{\pi(d)})$) as $\mathbf{E}_{\boldsymbol{v}_\pi} = \mathbf{E}_{\boldsymbol{v}} \cdot \mathbf{G}^{-1}_{dn,\ell,m}(\pi(\mathbf{I}_{dn}) \cdot \mathbf{G}_{dn,\ell,m})$, where $\pi(\mathbf{I}_{dn})$ means applying the permutation π on the position of the $\mathbf{I}_n$ component of $\mathbf{I}_{dn}$. We refer the readers to Section 3 for details. More importantly, this encoding and its supported operations are compatible with the GSW-FHE scheme, and thus can be applied in the settings of ABE and HS.

In the compact ABE scheme we describe in Section 4, the encryption of a message μ associated with attribute $\boldsymbol{x} = (x_1, ..., x_d) \in \mathbb{Z}_q^d$ is still computed in the dual-Regev way but with respect to the public key $[\mathbf{A}|\mathbf{D} + \mathbf{E}_{\boldsymbol{x}}] \in \mathbb{Z}_q^{n\times 2m}$, where $\mathbf{E}_{\boldsymbol{x}}$ is the vector encoding of attribute $\boldsymbol{x}$. During decryption, we first unpack the ciphertext $\boldsymbol{c}_{\boldsymbol{x}} = \boldsymbol{s}^\mathsf{T}(\mathbf{D} + \mathbf{E}_{\boldsymbol{x}}) + \boldsymbol{e}$ to obtain the i-th component $\boldsymbol{c}_{x_i} = \boldsymbol{s}^\mathsf{T}(\mathbf{D}_i + x_i\mathbf{G}_{n,2,m}) + \boldsymbol{e}_i$, where the matrix $\mathbf{D}_i$ can be computed deterministically from $\mathbf{D}, i$ and independently from $f, \boldsymbol{x}$. After obtaining each encryption attribute on an input wire, we can perform the homomorphic computing paradigm on these unpacked ciphertexts. The key generation algorithm is also modified accordingly by computing the matrix encoding the output wire of function f from these unpacked matrices $\{\mathbf{D}_i\}$.

In the SIMD-enabled (leveled) HS scheme we describe in Section 5, we generate the signature $\mathbf{R}_{\boldsymbol{x}}$ for message vector $\boldsymbol{x}$, satisfying $\mathbf{A}_\tau\mathbf{R}_{\boldsymbol{x}} = \mathbf{D} + \mathbf{E}_{\boldsymbol{x}}$. We define a set of SIMD operations in Section 2.2 and instantiate these operations in Section 5. We refer the reader to these sections for details. Unlike prior SIMD-enabled homomorphic encryption constructions [SV14, GHS12], our HS construction does *not* rely on polynomial ring structures, which are vulnerable to insecure instantiations as shown in [Pei16]. However, due to the merits of more compact space and efficient computation on ring structures, it may remain an interesting question to further optimize the ABE and HS schemes using polynomial rings, if the risks can be tolerated.

1.2 Other Related Work

Predicate Encryption. The primitive called predicate encryption [BW07, KSW08] provides a stronger privacy guarantee than ABE by hiding the attribute vector $\boldsymbol{x}$. There has been a few constructions of predicate encryption supporting inner-product encryption [KSW08, AFV11] and polynomial-size circuits [GVW15a].

Homomorphic MACs. There has also been progress in constructing *homomorphic message authentication* with private verification for larger classes of homomorphism. This notion is proposed in the work of [GW13] and constructed using fully homomorphic encryption. However, their construction only remains secure in a setting without *verification queries*. Later on, several works [CF13, BFR13, CFGN14] show how to get homomorphic MACs that remain secure even in the presence of verification queries, but only for restricted homomorphims. More recently, the work of [FMNP16] shows how to construct multi-key homomorphic signatures for circuits of bounded polynomial depth and multi-key homomorphic MACs for low-degree arithmetic circuits.

2 Preliminaries

In this section, we first recall the syntax and security definition of (key-policy) attribute-based encryption, then we define some SIMD operations for homomorphic signatures and their security requirements. Next, we review some lattice background, including assumptions, sampling algorithms, and the notion of gadget matrix, that are essential to our constructions.

Notation. Let λ denote the security parameter, and PPT denote probabilistic polynomial time. Bold uppercase letters are used to denote matrices $\mathbf{M}$, and bold lowercase letters for vectors $\boldsymbol{v}$. We write $\widetilde{\mathbf{M}}$ to denote the Gram-Schmidt orthogonalization of $\mathbf{M}$, and $[n]$ to denote the set $\{1, ..., n\}$. We use $\mathbf{I}_n$ to denote the $n \times n$ identity matrix. We say a function $\mathsf{negl}(\cdot) : \mathbb{N} \to (0, 1)$ is negligible, if for every constant $c \in \mathbb{N}$, $\mathsf{negl}(n) < n^{-c}$ for sufficiently large n.

2.1 Attribute-Based Encryption

In this part, we recall the syntax and security definition of (key-policy) attribute-based encryption (ABE). An ABE scheme for a family of functions $\mathcal{F} : \{0, 1\}^k \to \{0, 1\}$ consists a tuple of PPT algorithms $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with details as follows:

- $\mathsf{Setup}(1^\lambda)$: On input a security parameter λ, the setup algorithm outputs public parameters pp and master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, f)$: On input a master secret key msk and a function $f \in \mathcal{F}$, it outputs a secret key sk_f.
- $\mathsf{Enc}(\mathsf{pp}, \boldsymbol{x}, \mu)$: On input public parameters pp, an attribute vector $\boldsymbol{x}$ and a message μ, it outputs a ciphertext $\mathsf{ct}_{\boldsymbol{x}}$.
- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_{\boldsymbol{x}})$: On input a secret key sk_f and a ciphertext $\mathsf{ct}_{\boldsymbol{x}}$, it outputs the corresponding plaintext μ if $f(\boldsymbol{x}) = 0$; otherwise, it outputs $\perp$.

Definition 2.1 (Correctness)**.** *We say the ABE described above is correct, if for any* $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda)$, *any message μ, any circuit $f \in \mathcal{F}$, and any attribute $\boldsymbol{x}$ where $f(\boldsymbol{x}) = 0$, we have* $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_{\boldsymbol{x}}) = \mu$, *where* $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ *and* $\mathsf{ct}_{\boldsymbol{x}} \leftarrow \mathsf{Enc}(\mathsf{pp}, \boldsymbol{x}, \mu)$.

Security Definition. We present the definition of selective security of attribute-based encryption by first describing an experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{ABE}}(1^\lambda)$ between an adversary $\mathcal{A}$ and a challenger for an ABE scheme Π.

- **Setup**: The adversary $\mathcal{A}$ discloses the challenge attribute $\boldsymbol{x}^*$ to the challenger, then the challenger runs $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda)$ and sends pp to $\mathcal{A}$.
- **Key query phase I**: Proceeding adaptively, the adversary $\mathcal{A}$ submits key queries for functions $f \in \mathcal{F}$ to the challenger, with the restriction that $f(\boldsymbol{x}^*) \neq 0$. The challenger then sends back $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ to the adversary.
- **Challenge phase**: The adversary submits two challenge messages (μ_0^*, μ_1^*). The challenger first chooses a random bit $b \in \{0, 1\}$ and then sends back $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pp}, \boldsymbol{x}^*, \mu_b)$ to adversary $\mathcal{A}$.
- **Key query phase II**: The adversary $\mathcal{A}$ may continue his secret key queries for functions $f \in \mathcal{F}$ adaptively with the restriction that $f(\boldsymbol{x}^*) \neq 0$ for all function queries f.
- **Guess**: Finally, the adversary $\mathcal{A}$ outputs his guess b' for the bit b.

The adversary wins the experiment if $b' = b$.

Definition 2.2. *An ABE scheme Π for a family of functions $\mathcal{F}$ is selectively secure if no* PPT *adversary $\mathcal{A}$ can win the experiment* $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{ABE}}(1^\lambda)$ *with non-negligible probability.*

2.2 Homomorphic Signatures

We recall the definitions of homomorphic signatures that appeared in various works [BF11a, GVW15b, BFS14]. We denote the message space by $\mathcal{M}$, and let $f : \mathcal{M}^d \to \mathcal{M}$ denote a function that takes d messages and output a result message in $\mathcal{M}$. A homomorphic signature scheme for the function family $\mathcal{F}$ is a tuple of polynomial-time algorithms $\Sigma = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Verify})$ specified as follows:

- $\mathsf{Setup}(1^\lambda, 1^d)$: On input a security parameter λ and the maximum size d of a dataset whose messages can be signed, it outputs a public key pk and a secret key sk.
- $\mathsf{Sign}(\mathsf{sk}, i, \mu)$: On input a secret key sk, a message $\mu \in \{0, 1\}$ and its corresponding index $i \in [d]$ in the dataset, it outputs a signature σ.
- $\mathsf{Eval}(\mathsf{pk}, \boldsymbol{\mu} = (\mu_1, ..., \mu_d), \boldsymbol{\sigma} = (\sigma_1, ..., \sigma_d), f)$: On input a public key pk, a sequence of messages $\boldsymbol{\mu}$ and a corresponding sequence of signatures $\boldsymbol{\sigma}$, and a function $f \in \mathcal{F}$, it outputs a derived signature σ' which corresponds to the evaluated message $f(\boldsymbol{\mu})$.
- $\mathsf{Verify}(\mathsf{pk}, \mu, \sigma, f)$: Given a public key pk, a message $\mu = f(\boldsymbol{\mu})$, its signature σ, and a function $f \in \mathcal{F}$, it outputs 0 (reject) or 1 (accept).

Definition 2.3 (Correctness)**.** *We say that the $\mathcal{F}$-homomorphic signature Σ is correct, if for any function $f \in \mathcal{F}$, any set of messages $\boldsymbol{\mu} = (\mu_1, ..., \mu_d) \in \mathcal{M}^d$, and any index $i \in [d]$, we have*

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

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

SIMD Operations for Homomorphic Signatures. In this part, we introduce SIMD operations for homomorphic signatures, namely $(\mathsf{VSign}, \mathsf{VAdd}, \mathsf{VMult}, \mathsf{Unpack}, \mathsf{Permute}, \mathsf{CMult})$ as follows.:

- $\mathsf{VSign}(\mathsf{sk}, \boldsymbol{\mu} = (\mu_1, ..., \mu_d))$: Given secret key sk and message vector $\boldsymbol{\mu}$, the vector-signing algorithm outputs a signature σ for the message vector $\boldsymbol{\mu}$.
- $\mathsf{VAdd}(\mathsf{pk}, \sigma, \boldsymbol{\mu}, \boldsymbol{a} = (a_1, ..., a_d))$: Given a valid signature σ on a message vector $\boldsymbol{\mu}$ and a vector $\boldsymbol{a} \in \mathbb{Z}_q^d$, the vector-adding algorithm outputs a signature σ' for the message $\sum_{i=1}^d a_i \mu_i$.
- $\mathsf{VMult}(\sigma, \boldsymbol{\mu}, \boldsymbol{a} = (a_1, ..., a_d))$: Given a valid signature σ on message vector $\boldsymbol{\mu}$ and a vector $\boldsymbol{a} \in \mathbb{Z}_q^d$, the vector-adding algorithm outputs a signature σ' for message $\boldsymbol{\mu}' = (a_1\mu_1, ..., a_d\mu_d)$.
- $\mathsf{Unpack}(\sigma, \boldsymbol{\mu}, i)$: Given a valid signature σ on message vector $\boldsymbol{\mu}$ and an index $i \in [d]$, the unpacking algorithm output a valid signature σ_i for message μ_i.
- $\mathsf{Permute}(\sigma, \boldsymbol{\mu}, \pi)$: Given a valid signature σ on message vector $\boldsymbol{\mu}$ and a permutation π on set $[d]$, the permutation algorithm outputs a valid signature σ' for message $\boldsymbol{\mu}' = (\mu_{\pi(1)}, ..., \mu_{\pi(d)})$.
- $\mathsf{CMult}(\sigma, \boldsymbol{\mu}, S)$: Given a valid signature σ on message vector $\boldsymbol{\mu}$ and a subset $S \subset [d]$, the componentwise-multiplication algorithm outputs a valid signature σ' for message $\mu' = \prod_{i \in S} \mu_i$.

Existential Unforgeability. Before presenting the security definition of homomorphic signatures, we first recall the definition of *admissible functions* introduced in [GVW15b]. In a *leveled* homomorphic signatures scheme, each signature σ_i will have some associated "noise-level" β_i. The initial signatures produced by $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, i, \mu_i)$ will have a small "noise" level β_{init}. The noise level β of homomorphically evaluated signature $\sigma = \mathsf{Eval}(\mathsf{pk}, \boldsymbol{\mu} = (\mu_1, ..., \mu_d), \boldsymbol{\sigma} = (\sigma_1, ..., \sigma_d), g)$ depends on the noise levels of signatures σ_i, the function g and messages μ_i. If the noise level β of an evaluated signature σ exceeds some pre-specified threshold β_{max}, the correctness condition defined above may not hold.

Definition 2.4 (Admissible functions). *A function $g \in \mathcal{F}$ is called admissible on values $\mu_1, ..., \mu_d$, if for any signatures σ_i of message μ_i with noise level $\beta_i \leq \beta_{max}$, the noise level β of the evaluated signature $\sigma = \mathsf{Eval}(\mathsf{pk}, \boldsymbol{\mu} = (\mu_1, ..., \mu_d), \boldsymbol{\sigma} = (\sigma_1, ..., \sigma_d), g)$ satisfies $\beta \leq \beta_{max}$.*

Existential unforgeability is defined using the following experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{HS}}(1^\lambda)$ between an adversary $\mathcal{A}$ and a challenger. We note that the challenge can either use the normal signing algorithm Sign or the SIMD signing algorithm VSign to compute the signature.

- **Setup**: The challenger runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$ and sends pk to adversary $\mathcal{A}$.
- **Signing query**: The adversary $\mathcal{A}$ chooses messages $(\mu_1, ..., \mu_d) \in \mathcal{M}^d$ and sends them to challenger. The challenger computes $(\sigma_1, ..., \sigma_d) \leftarrow \mathsf{Sign}(\mathsf{sk}, (\mu_1, ..., \mu_d))$ (or $\sigma \leftarrow \mathsf{VSign}(\mathsf{sk}, \boldsymbol{\mu})$) and sends back the signatures $(\sigma_1, ..., \sigma_d)$ (or σ).
- **Forgery**: The adversary $\mathcal{A}$ outputs a function $f \in \mathcal{F}$, a value y' and signature σ'

We say the adversary wins the experiment if all of the following hold: (1) f is admissible on messages $(\mu_1, ..., \mu_d)$, (2) $y' \neq f(\mu_1, ..., \mu_d)$, (3) $\mathsf{Verify}(\mathsf{pk}, y', \sigma') = 1$.

Definition 2.5 (Adaptive existential unforgeablility in single-dataset scenario)**.** *We say a homomorphic signature scheme Σ is adaptively unforgeable against single-dataset query with respect to a function family $\mathcal{F}$ if no* PPT *adversary $\mathcal{A}$ can win the experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{HS}}(1^\lambda)$ with non-negligible probability.*

Remark 2.6. We point out some relaxions or extensions of the model that appeared in previous work [BF11a, GVW15b, BFS14]:

- **Selective single-dateset security**: In our homomorphic signature construction shown in Section 5, we focus on selective security in the single dataset setting. In the selective security model, the adversary chooses the message tuple $(\mu_1, ..., \mu_d)$ to be signed before seeing the public key pk. A generic transformation, similar to chameleon hash function, could transform a selectively secure homomorphic signature scheme into an adaptively secure one, as shown in [GVW15b].
- **Adaptive multi-dataset security**: A more general model, called multi-dataset homomorphic signatures, allows the signer to sign many different datasets, where each dataset is associated with a tag τ, and the verifier is assumed to know the tag of the dataset over which he wishes to verify computation. The homomorphic operations are only allowed within the same dataset; and, in this security model, the adversary is required to submit its signing queries on all messages belonging to the same dataset in one batch, before moving to the next dataset. The work of [GVW15b] shows how to construct multi-dataset homomorphic signatures from a single-dataset scheme.
- **Adaptive queries at the message-level**: Extending multi-dataset homomorphic signatures, a stronger security model introduced in [BFS14] allows the adversary to make signing queries on messages specified adaptively one at a time and in any order, regardless of the dataset to which they belong. The work of [BFS14] shows a transformation from multi-dataset homomorphic signatures satisfying dataset-level security to a scheme satisfying message-level security.

The present work mainly focuses on selectively secure homomorphic signatures in single-dataset scenario. As discussed above, we can apply those generic or semi-generic transformations to obtain homomorphic signatures scheme satisfying stronger security notions.

Remark 2.7. There is another security notion of homomorphic signatures, called context hiding. Intuitively, context hiding requires that a signature which certifies y as the output of some function f over original data should not reveal anything about the underlying data beyond the output of the computation. In [GVW15b], is proposed a simulation-based notion of context-hiding property for single-dataset case, where a context-hiding signature σ can be simulated given knowledge of only the function f and the output y, but without any other knowledge of the original messages $\boldsymbol{\mu}$. The simulation remains indistinguishable even given the underlying messages, the underlying signatures, and even the public/secret key of the scheme. As the main focus of this work is augmenting homomorphic signatures with SIMD operations, and the proof of context-hiding of our scheme is similar to that in [GVW15b], we omit the definitions and proof here.

2.3 Lattice Background

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 $\mathbf{B} = \{\boldsymbol{b}_1, ..., \boldsymbol{b}_m\} \subset \mathbb{Z}^m$. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, we define the "$q$-ary" integer lattices:

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

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

Let Λ be a discrete subset of $\mathbb{Z}^m$. 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 let $\rho_{\sigma,\boldsymbol{c}}(\Lambda) = \sum_{\boldsymbol{x}\in\Lambda} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x})$ be the discrete integral of $\rho_{\sigma,\boldsymbol{x}}$ over Λ, and let $\mathcal{D}_{\Lambda,\sigma,\boldsymbol{c}}(\boldsymbol{y}) := \frac{\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y})}{\rho_{\sigma,\boldsymbol{c}}(\Lambda)}$. We abbreviate this as $\mathcal{D}_{\Lambda,\sigma}$ when $\boldsymbol{c} = \mathbf{0}$.

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

Lemma 2.8 ([ABB10]). *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* $\mathsf{Pr}[||\mathbf{R}|| > 12\sqrt{2m}] < e^{-2m}$.
- *Let* $\mathbf{R}$ *be sampled from* $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$*, then we have* $\mathsf{Pr}[||\mathbf{R}|| > \sigma\sqrt{m}] < e^{-2m}$.

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. [DRS04].

Lemma 2.9 ([ABB10]). *Suppose that* $m > (n + 1)\log q + \omega(\log n)$*. 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 matrices 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:*

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

Learning With Errors. The LWE problem was introduced by Regev [Reg05], who showed that solving it *on average* is as hard as (quantumly) solving several standard lattice problems *in the worst case*.

Definition 2.10 (LWE). *For an integer $q = q(n) \geq 2$, and an error distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, the* Learning With Errors problem $\mathsf{LWE}_{n,m,q,\chi}$ *is to distinguish between the following pairs of distributions (e.g. as given by a sampling oracle $\mathcal{O} \in \{\mathcal{O}_{\boldsymbol{s}}, \mathcal{O}_\$\}$):*

$$\{\mathbf{A}, \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{x}\} \text{ and } \{\mathbf{A}, \boldsymbol{u}\}$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^m$, and $\boldsymbol{x} \leftarrow \chi^n$.

Small Integer Solution. The SIS problem was first suggested to be hard on average by Ajtai [Ajt99] and then formalized by Micciancio and Regev [MR04].

Definition 2.11 (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 $\mathbf{A}\boldsymbol{z} = 0 \bmod q$ and $||\boldsymbol{z}|| \leq \beta$.*

Micciancio and Regev [MR04] 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.

2.4 Sampling Algorithms and Gadget Matrix

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

Lemma 2.12 ([GPV08, AP10]). *Let q, n, m be positive integers with $q \geq 2$ and sufficiently large $m = \Omega(n \log q)$. 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*

$$||\mathbf{T_A}|| \leq O(n \log q) \quad \text{and} \quad ||\widetilde{\mathbf{T_A}}|| \leq O(\sqrt{n \log q})$$

except with $\mathsf{negl}(n)$ *probability.*

Lemma 2.13 ([GPV08, CHKP10, ABB10]). *Let $q > 2, m > n$. There are three sampling algorithms as follows:*

- *There is a* PPT *algorithm* $\mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \boldsymbol{u}, s)$, *that takes as input: (1) a rank-n matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, (2) a "short" basis $\mathbf{T_A}$ for lattice $\Lambda_q^\perp(\mathbf{A})$, a vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, (3) a Gaussian parameter $s > ||\widetilde{\mathbf{T_A}}|| \cdot \omega(\sqrt{\log(m)})$; then outputs a vector $\boldsymbol{r} \in \mathbb{Z}^{m+m_1}$ distributed statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(\mathbf{A}),s}$.*

- *There is a* PPT *algorithm* $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \boldsymbol{u}, s)$, *that takes as input: (1) a rank-n matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and any matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times m_1}$, (2) a "short" basis $\mathbf{T_A}$ for lattice $\Lambda_q^\perp(\mathbf{A})$, a vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, (3) a Gaussian parameter $s > ||\widetilde{\mathbf{T_A}}|| \cdot \omega(\sqrt{\log(m + m_1)})$; then outputs a vector $\boldsymbol{r} \in \mathbb{Z}^{m+m_1}$ distributed statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(\mathbf{F}),s}$ where $\mathbf{F} := (\mathbf{A}|\mathbf{B})$.*

- *There is a* PPT *algorithm* $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \boldsymbol{u}, s)$, *that takes as input: (1) a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and a rank-$n$ matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$, a matrix $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$, where $s_\mathbf{R} := ||\mathbf{R}|| = \sup_{\boldsymbol{x}:||\boldsymbol{x}||=1} ||\mathbf{R}\boldsymbol{x}||$, (2) a "short" basis $\mathbf{T_B}$ for lattice $\Lambda_q^\perp(\mathbf{B})$, a vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, (3) a Gaussian parameter $s > ||\widetilde{\mathbf{T_B}}|| \cdot s_\mathbf{R} \cdot \omega(\sqrt{\log m})$; then outputs a vector $\boldsymbol{r} \in \mathbb{Z}^{2m}$ distributed statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(\mathbf{F}),s}$ where $\mathbf{F} := (\mathbf{A}|\mathbf{AR} + \mathbf{B})$.*

Gadget matrix. We now recall the gadget matrix [MP12, AP14], and the extended gadget matrix technique appeared in [AFL16].

Definition 2.14. *Let $m = n \cdot \lceil \log q \rceil$, and define the gadget matrix*

$$\mathbf{G}_{n,2,m} = \boldsymbol{g}_2 \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n \times m}$$

where the vector $\boldsymbol{g}_2 = (1, 2, 4, ..., 2^{\lfloor \log q \rfloor}) \in \mathbb{Z}_q^{\lceil \log q \rceil}$. We will also refer to this gadget matrix as "powers-of-two" matrix. We define the inverse function $\mathbf{G}_{n,2,m}^{-1} : \mathbb{Z}_q^{n \times m} \to \{0,1\}^{m \times m}$ which expands each entry $a \in \mathbb{Z}_q$ of the input matrix into a column of size $\lceil \log q \rceil$ consisting of the bits of binary representations. We have the property that for any matrix $\mathbf{A} \in Z_q^{n \times m}$, it holds that $\mathbf{G}_{n,2,m} \cdot \mathbf{G}_{n,2,m}^{-1}(\mathbf{A}) = \mathbf{A}$.

As mentioned by [MP12] and explicitly described in [AFL16], the results for $\mathbf{G}_{n,2,m}$ and its trapdoor can be extended to other integer powers. In this direction, we can obtain a generalized notation for gadget matrices as follows:

For any modulus $q \geq 2$, for integer base $2 \leq \ell \leq q$, let $\boldsymbol{g}_\ell^T := \left[1, \ell, \ell^2, ..., \ell^{k_\ell - 1}\right] \in \mathbb{Z}_q^{1 \times k_\ell}$ for $k_\ell = \lceil \log_\ell q \rceil$. (Note that the typical base-2 $\boldsymbol{g}^T$ is $\boldsymbol{g}_2^T$.) For row dimension n and ℓ as before, we let $\mathbf{G}_{n,\ell} = \mathbf{g}_\ell^T \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n \times nk_\ell}$. The public trapdoor basis $\mathbf{T}_{\mathbf{G}_{n,\ell}}$ is given analogously. Similar to the above padding argument, $\mathbf{G}_{n,\ell} \in \mathbb{Z}_q^{n \times nk_b}$ can be padded into a matrix $\mathbf{G}_{n,\ell,m} \in \mathbb{Z}_q^{n \times m}$ for $m \geq nk_b$ without increasing the norm of $\widetilde{\mathbf{T}_{\mathbf{G}_{n,\ell,m}}}$ from that of $\widetilde{\mathbf{T}_{\mathbf{G}_{n,\ell}}}$.

Following [Xag13] and [AP14], we now introduce a related function, the Batch Change-of-Base function $\mathbf{G}_{n',\ell',m'}^{-1}(\cdot)$, as follows:

For any modulus $q \geq 2$, and for any integer base $2 \leq \ell' \leq q$, fix the integer $k_{\ell'} := \lceil \log_{\ell'}(q) \rceil$. For any integers $n' \geq 2$ and $m' \geq n'k_{\ell'}$ the function $\mathbf{G}_{n',\ell',m'}^{-1}(\cdot)$ takes as input a matrix from $\mathbb{Z}_q^{n' \times m'}$, first computes a matrix in $\{0, 1, ..., \ell' - 1\}^{n' \log_{\ell'}(q) \times m'}$ using the typical $\mathbf{G}^{-1}$ procedure (except with base-ℓ' output), then pads with rows of zeroes as needed to form a matrix in $\{0, 1, ..., \ell' - 1\}^{m' \times m'}$. For example, the typical base-2 $\mathbf{G}^{-1} = \mathbf{G}_{n,2,m}^{-1}$ takes $\mathbb{Z}_q^{n \times m}$ to $\{0,1\}^{m \times m}$ as expected.

3 Vector Encoding and Its Supported Operations

In this section, we present the vector encoding method introduced in [AFL16] and demonstrate its versatility in computation by listing its supported operations. Consider the vector space $\mathbb{Z}_q^d$. For a vector $\boldsymbol{v} = (v_1, ..., v_d) \in \mathbb{Z}_q^d$, we define the following encoding algorithm which maps a d-dimensional vector to an $n \times m$ matrix.

$$\mathsf{encode}(\boldsymbol{v}) = \mathbf{E}_{\boldsymbol{v}} = \left[v_1 \mathbf{I}_n | \cdots | v_d \mathbf{I}_n\right] \cdot \mathbf{G}_{dn,\ell,m} \in \mathbb{Z}_q^{n \times m}$$

Similarly, we also define the encoding for an integer $a \in \mathbb{Z}_q$ as: $\mathsf{encode}(a) = \mathbf{E}_a = a \cdot \mathbf{G}_{n,2,m} \in \mathbb{Z}_q^{n \times m}$.

Remark 3.1. *Here the encoding structure $\mathbf{E}_{\boldsymbol{v}}$ requires parameters n, d, ℓ, m, q specified in the application. For notational simplicity, we do not include these parameters as indices in the notation $\mathbf{E}_{\boldsymbol{v}}$. We assume that these parameters are known to all the following algorithms. Concrete setting of these parameters should be specified for each individual application.*

The above encoding supports the following two types of operations naturally: (1) vector space operations, and (2) packing and unpacking. Below we define these procedures.

Vector space operations. In this part, we show how to do some vector space operations over the vector encoding described above. The operations include: addition, scalar multiplication, inner product, and coordinate-wise multiplication and permutation. Let the matrices $\mathbf{E}_{\boldsymbol{v}}$ and $\mathbf{E}_a$ be the respective encodings of a vector $\boldsymbol{v}$ and an integer a.

- **Addition**: Given encodings $\mathbf{E}_{\boldsymbol{v}_1}$ and $\mathbf{E}_{\boldsymbol{v}_2}$, output the encoding of $\boldsymbol{v}_1 + \boldsymbol{v}_2$ as $\mathbf{E}_{\boldsymbol{v}_1} + \mathbf{E}_{\boldsymbol{v}_2}$
- **Scalar multiplication**: Given encodings $\mathbf{E}_{\boldsymbol{v}}$ and an integer $a \in \mathbb{Z}_q$, output the encoding of $a \cdot \boldsymbol{v}$ as

$$\mathbf{E}_{a\cdot\boldsymbol{v}} = \mathbf{E}_{\boldsymbol{v}} \cdot \mathbf{G}^{-1}_{dn,\ell,m}\Big(a \cdot \mathbf{G}_{dn,\ell,m}\Big) = \big[av_1\mathbf{I}_n|\cdots|av_d\mathbf{I}_n\big] \cdot \mathbf{G}_{dn,\ell,m}$$

- **Inner product**: Given an encoding $\mathbf{E}_{\boldsymbol{v}_1}$ and a vector $\boldsymbol{v}_2$, output the encoding of $\langle \boldsymbol{v}_1, \boldsymbol{v}_2\rangle$ as

$$\mathbf{E}_{\langle \boldsymbol{v}_1, \boldsymbol{v}_2\rangle} = \mathbf{E}_{\boldsymbol{v}_1} \cdot \mathbf{G}^{-1}_{dn,\ell,m}\left(\begin{bmatrix} \boldsymbol{v}_{21}\mathbf{I}_n \\ \vdots \\ \boldsymbol{v}_{2d}\mathbf{I}_n \end{bmatrix} \cdot \mathbf{G}_{n,2,m}\right) = \langle \boldsymbol{v}_1, \boldsymbol{v}_2\rangle \cdot \mathbf{G}_{n,2,m}$$

- **Coordinate-wise multiplication**: Given an encoding $\mathbf{E}_{\boldsymbol{v}_1}$ and a vector $\boldsymbol{v}_2$, output the encoding of $\boldsymbol{v}_1 \otimes \boldsymbol{v}_2 = (v_{11}v_{21}, ..., v_{1d}v_{2d})$ as

$$\begin{aligned}\mathbf{E}_{\boldsymbol{v}_1 \otimes \boldsymbol{v}_2} &= \mathbf{E}_{\boldsymbol{v}_1} \cdot \mathbf{G}^{-1}_{dn,\ell,m}\left(\begin{bmatrix} v_{21}\mathbf{I}_n & & \\ & \ddots & \\ & & v_{2d}\mathbf{I}_n \end{bmatrix} \cdot \mathbf{G}_{dn,\ell,m}\right) \\ &= [v_{11}v_{21}\mathbf{I}_n|\cdots|v_{1d}v_{2d}\mathbf{I}_n] \cdot \mathbf{G}_{dn,\ell,m}\end{aligned}$$

- **Permutation**: Given an encoding $\mathbf{E}_{\boldsymbol{v}}$ and a permutation π on the set $[d]$, output the encoding of the vector $\boldsymbol{v}_\pi = (v_{\pi(1)}, ..., v_{\pi(d)})$ as

$$\mathbf{E}_{\boldsymbol{v}_\pi} = \mathbf{E}_{\boldsymbol{v}} \cdot \mathbf{G}^{-1}_{dn,\ell,m}(\pi(\mathbf{I}_{dn}) \cdot \mathbf{G}_{dn,\ell,m})$$

  where $\pi(\mathbf{I}_{dn})$ is computed by applying the permutation π on the position of $\mathbf{I}_n$ component of $\mathbf{I}_{dn}$.

Packing and Unpacking. Intuitively, the packing procedure transforms d encodings of integers to an encoding of a d-dimensional vector, while the unpacking procedure unpacks the vector into d encodings of integers. Let d be a parameter implicitly included in the algorithms. The formal syntax is the following:

- $\mathsf{Pack}(\{\mathbf{E}_{a_i}\}_{i=1}^d)$: On input d encodings $\mathbf{E}_{a_i}$ for $a_i \in \mathbb{Z}_q$, the procedure outputs an encoding $\mathbf{E}_{\boldsymbol{a}}$ for a vector $\boldsymbol{a} = (a_1, ..., a_d)$.
- $\mathsf{Unpack}(\mathbf{E}_{\boldsymbol{v}}, i)$: On input an encoding $\mathbf{E}_{\boldsymbol{v}}$ for a vector $\boldsymbol{v}$ and an index $i \le d$, the procedure outputs an encoding $\mathbf{E}_{v_i}$ for the i-th coordinate v_i.

We instantiate the above two algorithms as follows. For $i \in [d]$, define the matrix $\mathbf{U}_i$ as the $dn \times n$ extended unit matrix:

$$\mathbf{U}_i^{\mathsf{T}} = [\mathbf{0}_n|\cdots|\mathbf{0}_n|\mathbf{I}_n|\mathbf{0}_n|\cdots|\mathbf{0}_n] \tag{1}$$

i.e. only the i-th block is the identity matrix $\mathbf{I}_n$ and other blocks are zero matrices $\mathbf{0}_n$.

- $\mathsf{Pack}(\{\mathbf{E}_{a_i}\}_{i=1}^d)$ outputs $\sum_{i\in[d]} \mathbf{E}_{a_i} \cdot \mathbf{G}^{-1}_{n,2,m}\left(\mathbf{U}_i^{\mathsf{T}} \cdot \mathbf{G}_{dn,\ell,m}\right) = [a_1\mathbf{I}_n|\cdots|a_d\mathbf{I}_n] \cdot \mathbf{G}_{dn,\ell,m}$.
- $\mathsf{Unpack}(\mathbf{E}_{\boldsymbol{v}}, i)$ outputs $\mathbf{E}_{v_i} = \mathbf{E}_{\boldsymbol{v}} \cdot \mathbf{G}^{-1}_{dn,\ell,m}(\mathbf{U}_i\mathbf{G}_{n,2,m}) = v_i\mathbf{G}_{n,2,m}$.

4 Application in Compact Attribute Based Encryption

In this section, we show how to use the vector encoding to optimize the size of public parameters and ciphertexts in the ABE construction.

4.1 Evaluating Algorithms

Before proceeding to describe the construction of ABE scheme, we first define three evaluation algorithms $(\mathsf{PubEval}, \mathsf{TrapEval}, \mathsf{CtEval})$, which will be used in the construction and security proof. The following definition about evaluating GSW-ciphertexts respective to some circuits is implicitly used in various constructions, such as attribute-based encryption [BGG+14, GV15] and predicate encryption [GVW15a]. This definition was formalized in [Yam17] regarding partitioning functions (e.g. hash function families) used in the IBE constructions. Here, we extend the definition to general circuits, e.g., $f : \mathcal{X}^d \to \mathcal{Y}$.

Definition 4.1 (δ-expanding evaluation)**.** *The deterministic algorithms* $(\mathsf{PubEval}, \mathsf{TrapEval}, \mathsf{CtEval})$ *are* δ*-expanding with function (circuit with* u *inputs)* $f : \mathcal{X}^d \to \mathcal{Y}$ *if they are efficient and satisfy the following properties:*

- $\mathsf{PubEval}(\{\mathbf{D}_i \in \mathbb{Z}_q^{n\times m}\}_{i\in[d]}, f)$*: On input matrices* $\{\mathbf{D}_i\}_{i\in[d]}$ *and a function* $f \in \mathcal{F}$*, the public evaluation algorithm outputs* $\mathbf{D}_f \in \mathbb{Z}_q^{n\times m}$ *as the result.*

- $\mathsf{TrapEval}(\boldsymbol{x} \in \mathcal{X}^d, \mathbf{A} \in \mathbb{Z}_q^{n\times m}, \{\mathbf{R}_i\}_{i\in[d]}, f)$*: the trapdoor evaluation algorithm outputs* $\mathbf{R}_f$*, such that*

$$\mathsf{PubEval}(\{\mathbf{A}\mathbf{R}_i + x_i\mathbf{G}_{n,2,m}\}_{i\in[d]}, f) = \mathbf{A}\mathbf{R}_f + f(\boldsymbol{x})\mathbf{G}_{n,2,m}$$

  Furthermore, we have $||\mathbf{R}_f|| \leq \delta \cdot \max_{i\in[d]} ||\mathbf{R}_i||$.

- $\mathsf{CtEval}(\{\boldsymbol{c}_i\}_{i=1}^d, \boldsymbol{x}, f)$*: On input vectors* $\{\boldsymbol{c}_i\}_{i=1}^d \in \mathbb{Z}_q^m$*, an attribute* $\boldsymbol{x}$ *and function* f*, the ciphertext evaluation algorithm outputs* $\boldsymbol{c}_{f(\boldsymbol{x})} \in \mathbb{Z}_q^m$*, such that*

$$\mathsf{CtEval}(\{\boldsymbol{s}^\mathsf{T}(\mathbf{D}_i + x_i\mathbf{G}_{n,2,m}) + \boldsymbol{e}_i\}_{i\in[d]}, \boldsymbol{x}, f) = \boldsymbol{s}^\mathsf{T}(\mathbf{D}_f + f(\boldsymbol{x})\mathbf{G}_{n,2,m}) + \boldsymbol{e}'$$

  where $\boldsymbol{x} = (x_1, ..., x_d)$ *and* $\mathbf{D}_f = \mathsf{PubEval}(\{\mathbf{D}_i \in \mathbb{Z}_q^{n\times m}\}_{i\in[d]}, f)$*. Furthermore, we require* $||\boldsymbol{e}'|| \leq \delta \cdot \max_{i\in[d]} ||\boldsymbol{e}_i||$.

The definition can be extended to δ-expanding with a family of functions $\mathcal{F}$. I.e., $(\mathsf{PubEval}, \mathsf{TrapEval})$ are δ-expanding with $\mathcal{F}$ if and only if for all $f \in \mathcal{F}$, the algorithms are δ-expanding with f.

The authors of [BGG+14, GV15] show how to instantiate the algorithms $(\mathsf{PubEval}, \mathsf{TrapEval}, \mathsf{CtEval})$ for general circuits $f : \mathcal{X}^d \to \mathcal{Y}$. We state their results as the following fact:

Fact 4.2 ([BGG+14, GV15])**.** *For an integer* d*, let* $\boldsymbol{x} = (x_1, \ldots, x_d)$ *be an input vector. There exist algorithms* $(\mathsf{PubEval}, \mathsf{TrapEval}, \mathsf{CtEval})$ *such that the following holds.*

- *Let function* $\mathsf{Add}_d : \mathbb{Z}_q^d \to \mathbb{Z}_q$ *be defined as* $\mathsf{Add}_d(\boldsymbol{x}) = \sum_{i=1}^d x_i$*. The algorithms are* d*-expanding with* Add_d.

- *Let function* $\mathsf{Mult}_{d,p} : [0, p-1]^d \to \mathbb{Z}_q$ *(each* $x_i \in [0, p-1]$ *for some bounded* $p \leq q$*) be defined as* $\mathsf{Mult}_d(\boldsymbol{x}_i) = \prod_{i=1}^d x_i$*. The algorithms are* $md(p-1)^d$*-expanding with* Mult_d .

Evaluation Algorithms for Packed Encodings. We now turn to our packed compact encodings. Our optimized evaluation algorithms $(\mathsf{PubEval_{cp}}, \mathsf{TrapEval_{cp}}, \mathsf{CtEval_{cp}})$ are described as follows:

- $\mathbf{D}_f \leftarrow \mathsf{PubEval_{cp}}(f, \mathbf{D})$: On input a function $f \in \mathcal{F}$ and matrix $\mathbf{D} \in \mathbb{Z}_q^{n\times m}$, where the number of input wires of circuit C is d, the evaluation algorithm does:
    1. For $i \in [d]$, compute the matrix $\mathbf{D}_i = \mathbf{D} \cdot \mathbf{G}^{-1}_{dn,\ell,m}(\mathbf{U}_i \mathbf{G}_{n,2,m})$ for the i-th input wire.
    2. Compute and output $\mathbf{D}_f = \mathsf{PubEval}(\{\mathbf{D}_i\}_{i\in[d]}, f)$

- $\boldsymbol{c}_{f(\boldsymbol{x})} \leftarrow \mathsf{CtEval_{cp}}(f, \boldsymbol{c}, \boldsymbol{x})$: On input a circuit $f \in \mathcal{F}$, a ciphertext $\boldsymbol{c}$ and its associated attribute $\boldsymbol{x} = (x_1, ..., x_d)$
    1. For $i \in [d]$, unpack the ciphertext as
    $$\boldsymbol{c}_i = \boldsymbol{c} \cdot \mathbf{G}^{-1}_{dn,\ell,m}(\mathbf{U}_i \mathbf{G}_{n,2,m}) = \boldsymbol{s}^{\mathsf{T}}(\mathbf{D}_i + x_i \mathbf{G}_{n,2,m}) + \boldsymbol{e}\mathbf{G}^{-1}_{dn,\ell,m}(\mathbf{U}_i \mathbf{G}_{n,2,m})$$
    2. Compute and output $\boldsymbol{c}_{f(\boldsymbol{x})} = \mathsf{CtEval}(\{\boldsymbol{c}_i\}_{i=1}^d, \boldsymbol{x}, f)$.

- $\mathbf{R}_f \leftarrow \mathsf{TrapEval_{cp}}(f, \boldsymbol{x}, \mathbf{R}, \mathbf{A})$: On input a function $f \in \mathcal{F}$, a vector $\boldsymbol{x} = (x_1, ..., x_d) \in \mathbb{Z}_q^d$, a matrix $\mathbf{R} \in \{-1, 1\}^{m\times m}$ and a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$,
    1. For $i \in [d]$, compute the matrix $\mathbf{R}_i = \mathbf{R} \cdot \mathbf{G}^{-1}_{dn,\ell,m}(\mathbf{U}_i \mathbf{G}_{n,2,m})$ for the i-th input wire.
    2. Compute and output $\mathbf{R}_f = \mathsf{TrapEval}(\boldsymbol{x}, \mathbf{A}, \{\mathbf{R}_i\}_{i\in[d]}, f)$.

Before showing the exact δ-expanding of algorithms $(\mathsf{PubEval_{cp}}, \mathsf{TrapEval_{cp}}, \mathsf{CtEval_{cp}})$, we prove a useful lemma about the norm of a worst-case estimate on $||\mathbf{G}^{-1}_{n,2,m}(\mathbf{U}_i^{\mathsf{T}} \cdot \mathbf{G}_{dn,\ell,m})||$

Lemma 4.3. *Let $\mathbf{U}_i$ be the $dn \times n$ extended unit matrix defined in Equation (1) for $i \in [d]$. We have $||\mathbf{G}^{-1}_{dn,\ell,m}(\mathbf{U}_i \mathbf{G}_{n,2,m})|| \le \ell \log_\ell q$.*

Proof. This bound can be proved by unfolding the formula as

$$\begin{aligned}\mathbf{G}^{-1}_{dn,\ell,m}(\mathbf{U}_i \mathbf{G}_{n,2,m}) &= \mathbf{G}^{-1}_{dn,\ell,m}([\mathbf{0}_{n\times m}|\cdots|\boldsymbol{g}_2 \otimes \mathbf{I}_n|\mathbf{0}_{n\times m}|\cdots|\mathbf{0}_{n\times m}]^{\mathsf{T}}) \\ &= [\mathbf{0}_m|\cdots|\mathbf{0}_m|\mathbf{X}\otimes \mathbf{I}_n|\mathbf{0}_m|\cdots|\mathbf{0}_m]^{\mathsf{T}} \in \mathbb{Z}_q^{m\times m}\end{aligned}$$

where $\mathbf{X} \in [\ell]^{\log_\ell q \times \log q}$. By worst-case analysis on $||\mathbf{X}||$, we have $||\mathbf{G}^{-1}_{dn,\ell,m}(\mathbf{U}_i \mathbf{G}_{n,2,m})|| \le \ell \log_\ell q$. □

The exact expansion factors δ for the δ-expanding algorithms $(\mathsf{PubEval_{cp}}, \mathsf{TrapEval_{cp}}, \mathsf{CtEval_{cp}})$ with addition and multiplication, are as follows:

Lemma 4.4. *For integer d, let $\boldsymbol{x} = (x_1, \ldots, x_d)$ be an input vector. Our algorithms $(\mathsf{PubEval_{cp}}, \mathsf{TrapEval_{cp}}, \mathsf{CtEval_{cp}})$ described above satisfy:*

- *Let function $\mathsf{Add}_d : \mathbb{Z}_q^d \to \mathbb{Z}_q$ be defined as $\mathsf{Add}_d(\boldsymbol{x}) = \sum_{i=1}^d x_i$. The algorithms are $(d\ell \log_\ell q)$-expanding with Add_u.*
- *Let function $\mathsf{Mult}_{d,p} : [0, p-1]^d \to \mathbb{Z}_q$ (each $x_i \in [0, p-1]$ for some bounded $p \le q$) be defined as $\mathsf{Mult}_d(\boldsymbol{x}_i) = \prod_{i=1}^d x_i$. The algorithms are $(md(p-1)^d \ell \log_\ell q)$-expanding with Mult_d.*

Proof. As we described above, our optimized evaluation algorithms $(\mathsf{PubEval_{cp}}, \mathsf{TrapEval_{cp}}, \mathsf{CtEval_{cp}})$ consist of two steps: (1) unpack the input to obtain the matrix/vector corresponding to each input wire of the function f; (2) run $(\mathsf{PubEval}, \mathsf{TrapEval}, \mathsf{CtEval})$ on these matrices/vectors. Combining Lemma 4.3 and Fact 4.2, we complete the proof. □

4.2 The Basic ABE Construction

Informally, we call an ABE scheme *basic* if its attribute space is $\mathbb{Z}_q^d$, where $d = \log \lambda$. The ABE scheme $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ can be described in the following:

- $\mathsf{Setup}(1^\lambda, 1^{\mathcal{F}})$: On input a security parameter λ and the description of a supported function family $\mathcal{F}$, the setup algorithm first generates a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ with associated trapdoor $\mathbf{T}_\mathbf{A} \in \mathbb{Z}_q^{m\times m}$ using $(\mathbf{A}, \mathbf{T}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(q, n, m)$, and then samples a random matrix $\mathbf{D} \leftarrow \mathbb{Z}_q^{n\times m}$ and a random vector $\boldsymbol{u} \in \mathbb{Z}_q^n$. It outputs the public parameter pp and master secret key msk as
$$\mathsf{pp} = (\mathbf{A}, \mathbf{D}, \boldsymbol{u}), \qquad \mathsf{msk} = \mathbf{T}_\mathbf{A}$$
- $\mathsf{KeyGen}(\mathsf{msk}, f)$: On input a master secret key msk and a function $f \in \mathcal{F}$, the key generation algorithm computes $\mathbf{D}_f = \mathsf{PubEval}_{\mathsf{cp}}(f, \mathbf{D})$, then samples a low-norm vector $\boldsymbol{r}_f \in \mathbb{Z}^{2m}$ using $\boldsymbol{r}_f \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{D}_f, \boldsymbol{u}, s)$ such that $[\mathbf{A}|\mathbf{D}_f] \cdot \boldsymbol{r}_f = \boldsymbol{u} \bmod q$. It outputs secret key $\mathsf{sk}_f = \boldsymbol{r}_f$.
- $\mathsf{Enc}(\mathsf{pp}, \boldsymbol{x}, \mu)$: On input a public parameter pp, an attribute $\boldsymbol{x} \in \mathbb{Z}_p^d$ and a message μ, the encryption algorithm randomly samples a vector $\boldsymbol{s} \in \mathbb{Z}_q^m$ and a matrix $\mathbf{R} \in \{-1, 1\}^{m\times m}$, then encodes the attribute vector $\boldsymbol{x} \in \mathbb{Z}_q^d$ as
$$\mathsf{encode}(\boldsymbol{x}) = \mathbf{E}_{\boldsymbol{x}} = \big[x_1\mathbf{I}_n | \cdots | x_d\mathbf{I}_n\big] \cdot \mathbf{G}_{dn,\ell,m}$$
and outputs the ciphertext $\mathsf{ct}_{\boldsymbol{x}} = (\boldsymbol{c}_0, \boldsymbol{c}_1, c_2) \in \mathbb{Z}_q^{2m+1}$ where
$$(\boldsymbol{c}_0, \boldsymbol{c}_1) = \boldsymbol{s}^\mathsf{T}[\mathbf{A}|\mathbf{D} + \mathbf{E}_{\boldsymbol{x}}] + (\boldsymbol{e}_0, \boldsymbol{e}_0^\mathsf{T}\mathbf{R}), \quad c_2 = \boldsymbol{s}^\mathsf{T}\boldsymbol{u} + e_1 + \lceil q/2 \rfloor \mu$$
with error terms $\boldsymbol{e}_0 \leftarrow \mathcal{D}_{\mathbb{Z}^m,s}$, $e_1 \leftarrow \mathcal{D}_{\mathbb{Z},s}$.
- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_{\boldsymbol{x}})$: On input a secret key sk_f and ciphertext $\mathsf{ct}_{\boldsymbol{x}} = (\boldsymbol{c}_0, \boldsymbol{c}_1, c_2)$, if $f(\boldsymbol{x}) \neq 0$, output $\perp$; otherwise, compute $\boldsymbol{c}_{f(\boldsymbol{x})} = \mathsf{CtEval}_{\mathsf{cp}}(f, \boldsymbol{c}_1, \boldsymbol{x}) \in \mathbb{Z}_q^m$ and output the decrypted message as
$$\mathsf{Round}_{\bmod 2}\big(2/q \cdot (c_2 - \langle(\boldsymbol{c}_0, \boldsymbol{c}_{f(\boldsymbol{x})}), \boldsymbol{r}_f\rangle)\big)$$

Correctness. We prove the correctness of our compact ABE scheme as follows:

Lemma 4.5. *The ABE scheme* Π *described above is correct w.r.t. the requirements of Definition 2.1.*

Proof. When the attribute vector $\boldsymbol{x}$ of the ciphertext $\mathsf{ct}_{\boldsymbol{x}}$ satisfies the function f underlying the secret key sk_f, i.e., when $f(\boldsymbol{x}) = 0$, then decryption will produce
$$\begin{aligned}\mu' &= \mathsf{Round}_{\bmod 2}\big(2/q \cdot (c_2 - \langle(\boldsymbol{c}_0, \boldsymbol{c}_{f(\boldsymbol{x})}), \boldsymbol{r}_f\rangle)\big)\\ &= \mathsf{Round}_{\bmod 2}\left(\frac{2}{q}\lceil q/2\rceil\mu + \frac{2}{q}\underbrace{(e_1 - \langle(\boldsymbol{e}_0, \boldsymbol{e}_0'), \boldsymbol{r}_f\rangle)}_{\text{small}}\right) = \mu \in \{0,1\}\end{aligned}$$
where $\boldsymbol{c}_{f(\boldsymbol{x})} = \mathsf{CtEval}_{\mathsf{cp}}(f, \boldsymbol{c}_1, \boldsymbol{x})$. The second equality follows from the definitions of $\boldsymbol{c}_0, \boldsymbol{c}_f$ and the δ-expanding property of algorithm $\mathsf{CtEval}_{\mathsf{cp}}$. The third equality follows if $e_1 - \langle(\boldsymbol{e}_0, \boldsymbol{e}_0'), \boldsymbol{r}_f\rangle$ is indeed small w.r.t. the modulus q, which will hold w.h.p. by setting the parameters appropriately as below. $\square$

Param	Description	Setting
λ	security parameter	
n	lattice row dimension	λ
m	lattice column dimension	$n^{1+\delta}$
q	modulus	$n^{5+\epsilon}m^4$
p	attribute range	$\omega(1)$
s	sampling algorithm and error width	$n^{2+\epsilon}m$
ℓ	integer-base parameter	n

Table 1: Basic ABE Parameters Setting

Parameter selection. To support a function circuit of depth $t(\lambda) = \log \lambda$, we set the system parameters according to Table 1, where $\epsilon > 0$ is an arbitrarily small constant. These values are chosen in order to satisfy the following constraints:

- To ensure correctness, we require $|e_1 - \langle(\boldsymbol{e}_0, \boldsymbol{e}'_0), (\boldsymbol{r}_{f1}, \boldsymbol{r}_{f2})\rangle| < q/4$; here we bound the dominating term:
$$|\boldsymbol{e}_0'^{\mathsf{T}}\boldsymbol{r}_{f2}| \leq ||\boldsymbol{e}_0'^{\mathsf{T}}|| \cdot ||\boldsymbol{r}_{f2}|| \approx s\sqrt{m}mt(p-1)^t\ell\log_\ell q \cdot s\sqrt{m} = n^{5+\epsilon}m^4 < q/4.$$
- For $\mathsf{SampleLeft}$, we know $||\widetilde{\mathbf{T}_{\mathbf{A}}}|| = O(\sqrt{n\log(q)})$; this requires that the sampling width s satisfy $s > \sqrt{n\log(q)} \cdot \omega(\sqrt{\log(m)})$.
- For $\mathsf{SampleRight}$, we need $s > ||\widetilde{\mathbf{T}_{\mathbf{G}_{n,2,m}}}|| \cdot ||\mathbf{R}_f||\omega(\sqrt{\log m}) = n^{1+\epsilon}\omega(\sqrt{\log m})mt(p-1)^t\ell\log_\ell q$.
- To apply Regev's worst-case reduction, we need $s > \sqrt{n}\omega(\log(n))$ (s here is an absolute value, not a ratio). Therefore, we need $s > n^{2+\epsilon}m$.
- To apply the Leftover Hash Lemma, we need $m \geq (n+1)\log(q) + \omega(\log(n))$.

Remark 4.6. Our ABE construction can support $\mathsf{poly}(\lambda)$-depth circuit by assuming hardness of the subexponential LWE problem. Using the standard LWE assumption, we can pack $\log q = O(\log \lambda)$ attributes into one ciphertext. Under the subexponential LWE assumption, we can pack a slightly larger number $\log q = O(\log \lambda^{1+\delta})$ of attributes into one ciphertext.

4.3 Security Proof

We now turn to proving security of our ABE construction.

Theorem 4.7. *Given the three algorithms* ($\mathsf{PubEval}_{\mathsf{cp}}$, $\mathsf{TrapEval}_{\mathsf{cp}}$,
$\mathsf{CtEval}_{\mathsf{cp}}$) *of Definition 4.1, then, under the* (n, q, χ)*-LWE assumption, the ABE scheme described above is selectively secure in the sense of Definition 2.2.*

Proof. The proof proceeds in a sequence of hybrid experiments where the first experiment is identical to the ABE game from Definition 2.2. The last experiment in the sequence trivially collapses any adversary's advantage to zero. We show that a PPT adversary cannot distinguish between the hybrids which will prove that the adversary has negligible advantage in winning the original ABE security game. The LWE problem is used in proving that hybrids 2 and 3 are indistinguishable. The hybrid sequence is described as follows:

Hybrid H_0: This is the original ABE security game from Definition 2.2 between an adversary $\mathcal{A}$ against our scheme and an ABE challenger.

Hybrid H_1:In hybrid H_1, we change how the matrix $\mathbf{D} \in \mathbb{Z}_q^{n\times m}$ in the public parameters is generated. Recall that in hybrid H_0, $\mathbf{D}$ is sampled randomly from $\mathbb{Z}_q^{n\times m}$, and in the challenge ciphertext generation, we sample a random matrix $\mathbf{R} \in \{-1,1\}^{m\times m}$. In hybrid H_1, upon receiving the challenge attribute $\boldsymbol{x}^* = (x_1^*, ..., x_d^*) \in \mathbb{Z}_q^d$, we set $\mathbf{D}$ to be

$$\mathbf{D} = \mathbf{A}\mathbf{R} - \mathbf{E}_{\boldsymbol{x}^*}, \quad \mathbf{E}_{\boldsymbol{x}^*} = \left[x_1^*\mathbf{I}_n | \cdots | x_d^*\mathbf{I}_n\right] \cdot \mathbf{G}_{dn,\ell,m}$$

where the matrix $\mathbf{R}$ is sampled randomly from $\{-1,1\}^{m\times m}$. The same random matrix $\mathbf{R}$ is also used in the generation of the challenge ciphertext.

Hybrid H_2: In hybrid H_2, we change how the matrix $\mathbf{A}$ in pp is generated. Recall that in hybrid H_1, $\mathbf{A}$ is generated by the algorithm $\mathsf{TrapGen}(q,n,m)$. In hybrid H_2, we sample $\mathbf{A}$ randomly from $\mathbb{Z}_q^{n\times m}$. Lacking a trapdoor for $\mathbf{A}$, the challenger now uses algorithm $\mathsf{SampleRight}$ along with the trapdoor $\mathbf{T}_{\mathbf{G}_{n,2,m}}$ to answer key queries. Consider a key query for function $f \in \mathcal{F}$, where $f(\boldsymbol{x}^*) \neq 0$. To respond, the challenger does:

1. Run $\mathbf{R}_f \leftarrow \mathsf{TrapEval_{cp}}(f, \boldsymbol{x}^*, \mathbf{R}, \mathbf{A})$. By the δ-expanding property of $\mathsf{TrapEval_{cp}}$ (c.f. Definition 4.1), $\mathbf{R}_f$ is a low-norm matrix and satisfies $\mathbf{A}\mathbf{R}_f - f(\boldsymbol{x}^*)\mathbf{G}_{n,2,m} = \mathbf{D}_f$, for some $\mathbf{D}_f \leftarrow \mathsf{PubEval_{cp}}(f, \mathbf{D})$ with the correct distribution.
2. Compute $\boldsymbol{r}_f \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}_{n,2,m}, \mathbf{R}_f, \boldsymbol{u}, s)$, such that $[\mathbf{A}|\mathbf{A}\mathbf{R}_f - f(\boldsymbol{x}^*)\mathbf{G}_{n,2,m}] \cdot \boldsymbol{r}_f = \boldsymbol{u} \bmod q$.

Hybrid H_3: Hybrid H_3 is identical to hybrid H_2 except that the challenge ciphertext $(\boldsymbol{c}_0^*, \boldsymbol{c}_1^*, c_2)$ is chosen as a random independent vector in $\mathbb{Z}_q^{2m+1}$. Since the challenge ciphertext is always a fresh random sample from the ciphertext space, the adversary $\mathcal{A}$'s advantage in this hybrid is zero.

Analysis of the Hybrid Sequence. We prove the indistinguishability between two consecutive hybrids in the following lemmas.

Lemma 4.8. *Hybrid* H_0 *and* H_1 *are statistically indistinguishable.*

Proof. We show that hybrid H_1 is statistically close to H_0 using Lemma 2.9. The only difference between the two hybrids is how the matrix $\mathbf{D}$ and the error vector $\boldsymbol{e}_0^\mathsf{T}\mathbf{R}$ are generated. All the other matrices and vectors are generated identically. In H_0, $(\mathbf{D}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R})$ together with the public matrix $\mathbf{A}$ look like $(\mathbf{A}, \mathbf{D}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R})$, yet in H_1, they look like $(\mathbf{A}, \mathbf{A}\mathbf{R} - \mathbf{E}_{\boldsymbol{x}^*}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R})$, where $\mathbf{E}_{\boldsymbol{x}^*} = \left[x_1^*\mathbf{I}_n | \cdots | x_d^*\mathbf{I}_n\right] \cdot \mathbf{G}_{dn,\ell,m}$. Therefore, it suffices to show

$$(\mathbf{A}, \mathbf{D}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R}) \approx (\mathbf{A}, \mathbf{A}\mathbf{R} - \mathbf{E}_{\boldsymbol{x}^*}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R})$$

By Lemma 2.9, we know that the following two distributions are statistically close:

$$(\mathbf{A}, \mathbf{D}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R}) \approx (\mathbf{A}, \mathbf{A}\mathbf{R}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R})$$

where $\mathbf{D} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$. Thus, we have

$$(\mathbf{A}, \mathbf{D} - \mathbf{E}_{\boldsymbol{x}^*}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R}) \approx (\mathbf{A}, \mathbf{A}\mathbf{R} - \mathbf{E}_{\boldsymbol{x}^*}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R})$$

As $(\mathbf{A}, \mathbf{D} - \mathbf{E}_{\boldsymbol{x}^*}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R})$ is identically distributed as $(\mathbf{A}, \mathbf{D}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R})$, we have

$$(\mathbf{A}, \mathbf{D}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R}) \approx (\mathbf{A}, \mathbf{A}\mathbf{R} - \mathbf{E}_{\boldsymbol{x}^*}, \boldsymbol{e}_0^\mathsf{T}\mathbf{R})$$

□

Lemma 4.9. *Hybrid* H_1 *and* H_2 *are statistically indistinguishable.*

Proof. We show that hybrid H_2 is statistically close to H_1 using Lemmas 2.12 and 2.13. The differences are that, in hybrid H_1, (1) the pubic matrix $\mathbf{A}$ is generated using algorithm $\mathsf{TrapGen}$, and (2) the secret-key queries are answered using algorithm $\mathsf{SampleLeft}$, while in hybrid H_2, $\mathbf{A}$ is sampled uniformly from $\mathbb{Z}_q^{n\times m}$ (without known trapdoor), and the secret-key queries are answered using algorithm $\mathsf{SampleRight}$.

By Lemma 2.12, the matrix $\mathbf{A}$ in hybrid H_1 is distributed close to the uniform distribution over $\mathbb{Z}_q^{n\times m}$ seen in hybrid H_2. Also, by Lemma 2.13, with the Gaussian parameter s appropriately set as above, the outputs of $\mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ are distributed identically. Therefore, the hybrids H_1 and H_2 are statistically indistinguishable to the adversary. □

Lemma 4.10. *Assuming hardness of the LWE problem, the hybrids* H_2 *and* H_3 *are computationally indistinguishable.*

Proof. Suppose there exists an adversary with non-negligible advantage in distinguishing hybrid H_2 from H_3, then we can construct a reduction $\mathcal{B}$ that breaks the LWE assumption using $\mathcal{A}$. Recall from Definition 2.10 that an LWE instance is provided as a sampling oracle $\mathcal{O}$ that is either a uniformly random generator $\mathcal{O}_\$$ or an LWE sampler $\mathcal{O}_{\boldsymbol{s}}$ holding some random LWE secret $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$. The reduction $\mathcal{B}$ uses the adversary $\mathcal{A}$ to distinguish the two oracles as follows:

- **Invocation.** Reduction $\mathcal{B}$ requests $m+1$ instances from the challenge oracle $\mathcal{O}$, i.e., pairs $\{(\boldsymbol{a}_i, b_i)\}_{i=1}^{m+1}$.
- **Setup.** To construct the public parameter pp, the reduction $\mathcal{B}$ does:
  1. Obtain a challenge attribute $\boldsymbol{x}^*$ from the adversary $\mathcal{A}$.
  2. Set the matrix $\mathbf{A} = \{(\boldsymbol{a}_i)\}_{i=1}^{m} \in \mathbb{Z}_q^{n\times m}$ from the first m pairs.
  3. Set the vector $\boldsymbol{u} = \boldsymbol{a}_{m+1}$ from the remaining $(m+1)$-th pair.
  4. Construct the matrix $\mathbf{D}$ as in hybrid H_2 (see H_1).
  5. Send $\mathsf{pp} = (\mathbf{A}, \mathbf{D}, \boldsymbol{u})$ to the adversary $\mathcal{A}$.
- **Secret key queries.** Reduction $\mathcal{B}$ answers secret-key queries from $\mathcal{A}$ as in hybrid H_2.
- **Challenge ciphertext.** When the adversary $\mathcal{A}$ sends its challenge message pair (μ_0^*, μ_1^*), the reduction $\mathcal{B}$ does:
  1. Set the vector $\boldsymbol{b} = \{b_i\}_{i=1}^{m}$ from the first m integers b_i.
  2. Set the challenge ciphertext $(\boldsymbol{c}_0^*, \boldsymbol{c}_1^*, c_2^*)$ as

$$\boldsymbol{c}_0^* = \boldsymbol{b}, \quad \boldsymbol{c}_1^* = \boldsymbol{b}^{\mathsf{T}}\mathbf{R}, \quad c_2^* = b_{m+1} + \mu_{b^*}\lceil q/2 \rceil$$

     where the matrix $\mathbf{R}$ is the same one used to create $\mathbf{D}$ in the public parameter pp as shown above.
  3. Send the challenge ciphertext $(\boldsymbol{c}_0^*, \boldsymbol{c}_1^*, c_2^*)$ to adversary $\mathcal{A}$.
- **Guess.** After being allowed to make additional queries subject to the natural restriction, $\mathcal{A}$ guesses if it is interacting with a challenger in an H_2 or H_3 context. If the former, $\mathcal{B}$ outputs "LWE sampler" as the answer to the LWE challenge it is trying to solve; if the latter, $\mathcal{B}$ outputs "random sampler".

This concludes the reduction. □

Combined, the lemmas show the indistinguishability of the full sequence $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$, which proves the security theorem. □

4.4 ABE Construction for Polynomial-Length Attributes

In this part, we describe how to extend our basic ABE construction to build an ABE scheme supporting $\mathsf{poly}(\lambda)$-length attributes at the cost of somewhat less compact public keys and ciphertexts.

We use an additional parameter t to index the scheme, where t is determined by the length of the attributes and will be set after the description. In particular, the attribute space here would be $(\mathbb{Z}_q^d)^t$ The construction $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is as follows:

- $\mathsf{Setup}(1^\lambda, 1^{\mathcal{F}})$: On input a security parameter λ and the description of a supported function family $\mathcal{F}$, the setup algorithm first generates a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ with an associated trapdoor $\mathbf{T}_\mathbf{A} \in \mathbb{Z}_q^{m\times m}$ using $(\mathbf{A}, \mathbf{T}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(q, n, m)$, and draws t random matrices $\{\mathbf{D}_i\}_{i=1}^t \leftarrow \mathbb{Z}_q^{n\times m}$ and a random vector $\boldsymbol{u} \in \mathbb{Z}_q^n$. The public parameter pp and master secret msk are output as

$$\mathsf{pp} = (\mathbf{A}, \{\mathbf{D}_i\}_{i=1}^t, \boldsymbol{u}), \qquad \mathsf{msk} = \mathbf{T}_\mathbf{A}$$

- $\mathsf{KeyGen}(\mathsf{msk}, f)$: On input a master secret key msk and a function $f \in \mathcal{F}$, the key generation algorithm computes $\mathbf{D}_f = \mathsf{PubEval}(f, \{\mathbf{D}\}_{i=1}^t)$, then samples a low-norm vector $\boldsymbol{r}_f \in \mathbb{Z}^{2m}$ using $\boldsymbol{r}_f \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{D}_f, \boldsymbol{u}, s)$ such that $[\mathbf{A}|\mathbf{D}_f] \cdot \boldsymbol{r}_f = \boldsymbol{u} \bmod q$. It outputs a secret key $\mathsf{sk}_f = \boldsymbol{r}_f$.

- $\mathsf{Enc}(\mathsf{pp}, \boldsymbol{x}, \mu)$: On input a public parameter pp, an attribute $\boldsymbol{x} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_t) \in (\mathbb{Z}_q^d)^t$ and a message μ, the encryption algorithm chooses a random vector $\boldsymbol{s} \in \mathbb{Z}_q^m$ and t matrices $\mathbf{R}_i \leftarrow \{-1, 1\}^{m\times m}$ for $i \in [t]$, then encodes the attribute $\boldsymbol{x} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_t) \in (\mathbb{Z}_q^d)^t$ as

$$\forall i \in [t], \mathsf{encode}(\boldsymbol{x}_i) = \mathbf{E}_{\boldsymbol{x}_i} = \left[x_{i1}\mathbf{I}_n | \cdots | x_{id}\mathbf{I}_n\right] \cdot \mathbf{G}_{dn,\ell,m}$$

  The ciphertext is $\mathsf{ct}_{\boldsymbol{x}} = (\boldsymbol{c}_0, \{\boldsymbol{c}_{1i}\}_{i=1}^t, c_2) \in \mathbb{Z}_q^{2m+1}$ where

$$(\boldsymbol{c}_0, \boldsymbol{c}_{11}, .., \boldsymbol{c}_{1t}) = \boldsymbol{s}^\mathsf{T}[\mathbf{A}|\mathbf{D}_1 + \mathbf{E}_{\boldsymbol{x}_1}| \cdots |\mathbf{D}_t + \mathbf{E}_{\boldsymbol{x}_t}] + (\boldsymbol{e}_0, \boldsymbol{e}_0^\mathsf{T}\mathbf{R}_1, \cdots, \boldsymbol{e}_0^\mathsf{T}\mathbf{R}_t)$$

$$c_2 = \boldsymbol{s}^\mathsf{T}\boldsymbol{u} + e_1 + \lceil q/2 \rceil \mu$$

  with discrete Gaussian error terms $\boldsymbol{e}_0 \leftarrow \mathcal{D}_{\mathbb{Z}^m,s}, e_1 \leftarrow \mathcal{D}_{\mathbb{Z},s}$.

- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_{\boldsymbol{x}})$: On input a secret key sk_f and a ciphertext $\mathsf{ct}_{\boldsymbol{x}} = (\boldsymbol{c}_0, \{\boldsymbol{c}_{1i}\}_{i=1}^t, c_2)$, if $f(\boldsymbol{x}) \neq 0$, output $\perp$. Otherwise, compute $\boldsymbol{c}_{f(\boldsymbol{x})} = \mathsf{CtEval}(f, \{\boldsymbol{c}_{1i}\}_{i=1}^t, \boldsymbol{x}) \in \mathbb{Z}_q^m$, and output

$$\mathsf{Round}_{\bmod 2} \frac{2}{q}\left(c_2 - \langle(\boldsymbol{c}_0, \boldsymbol{c}_{f(\boldsymbol{x})}), \boldsymbol{r}_f\rangle\right)$$

Correctness and security for $\mathsf{poly}(\lambda)$-length attributes easily follow by analogy with the foregoing proofs for $(d \log q)$-bit attributes, and we omit them here. To support $\mathsf{poly}(\lambda)$-length attributes, it suffices to set $t = \mathsf{poly}(\lambda)/\log q$, while the remaining parameters can be set similarly as before.

5 Application to SIMD-Enabled Homomorphic Signatures

In this section, we describe (fully) homomorphic signatures for vector messages that support efficient "parallel" SIMD operations. Our schemes and security proofs below refer to the evaluation algorithms $(\mathsf{PubEval}, \mathsf{TrapEval})$ given previously in Definition 4.1.

Our SIMD-enabled fully homomorphic signature construction $\Sigma = (\mathsf{Setup}, \mathsf{VSign}, \mathsf{VAdd}, \mathsf{VMult}, \mathsf{Unpack}, \mathsf{CMult}, \mathsf{Permu}$ works follows:

- $\mathsf{Setup}(1^\lambda, 1^d)$: On input a security parameter λ and a maximum number d of messages permitted in a dataset, the setup algorithm first sets the parameters n, m, q, s, B as in Section 5.1. Then it generates a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ along with a trapdoor $\mathbf{T_A} \in \mathbb{Z}_q^{m\times m}$, using $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(q, n, m)$, and samples a random matrix $\mathbf{D} \in \mathbb{Z}_q^{n\times m}$. It outputs a public verification key $\mathsf{pk} = (\mathbf{A}, \mathbf{D})$ and a secret signing key $\mathsf{sk} = \mathbf{T_A}$.

- $\mathsf{VSign}(\mathsf{sk}, \boldsymbol{\mu} = (\mu_1, ..., \mu_d))$: On input a secret key sk and a message vector $\boldsymbol{\mu}$, the vector-signing algorithm samples a matrix $\mathbf{R} \in \mathbb{Z}^{m\times m}$, using

$$\mathbf{R} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \mathbf{D} + \mathbf{E}_{\boldsymbol{\mu}}, s)$$

where $\mathbf{E}_{\boldsymbol{\mu}} = \left[\mu_1\mathbf{I}_n|\cdots|\mu_d\mathbf{I}_n\right] \cdot \mathbf{G}_{dn,\ell,m}$ is the encoding of the message $\boldsymbol{\mu}$. It output a signature $\sigma_{\boldsymbol{\mu}} = \mathbf{R}$.

- $\mathsf{VAdd}(\mathsf{pk}, \sigma, \boldsymbol{\mu}, \boldsymbol{a})$: Given a signature/message pair $(\sigma = \mathbf{R}, \boldsymbol{\mu})$ and a vector $\boldsymbol{a} = (a_1, ..., a_d) \in \mathbb{Z}_q^d$, the vector-adding algorithm first checks the validity of the signature by running $\mathsf{Verify}(\mathsf{pk}, \sigma, \boldsymbol{\mu})$ and outputs $\perp$ if invalid. Otherwise it computes and outputs a signature $\sigma_{\langle\boldsymbol{\mu},\boldsymbol{a}\rangle}$ for message $\sum_{i=1}^d a_i\mu_i$ as

$$\mathbf{R}_{\langle\boldsymbol{\mu},\boldsymbol{a}\rangle} = \mathbf{R}\cdot\mathbf{G}_{dn,\ell,m}^{-1}\left(\begin{bmatrix} a_1\mathbf{I}_n \\ \vdots \\ a_d\mathbf{I}_n\end{bmatrix}\cdot\mathbf{G}_{n,2,m}\right)$$

- $\mathsf{VMult}(\mathsf{pk}, \sigma, \boldsymbol{\mu}, \boldsymbol{a})$: Given a signature/message pair $(\sigma = \mathbf{R}, \boldsymbol{\mu})$ and a vector $\boldsymbol{a} \in \mathbb{Z}_q^d$, the vector-multiplication algorithm first checks the validity of the signature by running $\mathsf{Verify}(\mathsf{pk}, \sigma, \boldsymbol{\mu})$ and outputs $\perp$ if invalid. Otherwise it computes and outputs a signature $\sigma_{\boldsymbol{\mu}\otimes\boldsymbol{a}}$ for message $\boldsymbol{\mu}\otimes\boldsymbol{a} = (\mu_1a_1, ..., \mu_da_d)$ as

$$\mathbf{R}_{\boldsymbol{\mu}\otimes\boldsymbol{a}} = \mathbf{R}\cdot\mathbf{G}_{dn,\ell,m}^{-1}\left(\begin{bmatrix} a_1\mathbf{I}_n & & \\ & \ddots & \\ & & a_d\mathbf{I}_n\end{bmatrix}\cdot\mathbf{G}_{dn,\ell,m}\right)$$

- $\mathsf{Unpack}(\mathsf{pk}, \sigma, \boldsymbol{\mu}, i)$: Given a signature/message pair $(\sigma = \mathbf{R}, \boldsymbol{\mu})$ and an index $i \in [d]$, the unpacking algorithm first checks the validity of the signature by running $\mathsf{Verify}(\mathsf{pk}, \sigma, \boldsymbol{\mu})$ and outputs $\perp$ if invalid. Otherwise it computes and outputs a signature σ_{μ_i} as

$$\mathbf{R}_{\mu_i} = \mathbf{R}\cdot\mathbf{G}_{dn,\ell,m}^{-1}(\mathbf{U}_i\mathbf{G}_{n,2,m})$$

where the matrix $\mathbf{U}_i$ is the $dn \times n$ extended unit matrix defined in Equation (1).

- $\mathsf{CMult}(\mathsf{pk}, \sigma, \boldsymbol{\mu}, S)$: Given a signature/message pair $(\sigma = \mathbf{R}, \boldsymbol{\mu})$ and a set $S \subset [d]$, the component-wise multiplication algorithm first checks the validity of the signature by running $\mathsf{Verify}(\mathsf{pk}, \sigma, \boldsymbol{\mu})$ and outputs $\perp$ if invalid. Otherwise for $i \in [S]$, it first runs the unpacking algorithm to obtain $\sigma_{\mu_i} = \mathbf{R}_i = \mathsf{Unpack}(\mathsf{pk}, \sigma, \boldsymbol{\mu}, i)$, and then computes and outputs a signature $\sigma_{\prod_{i\in S}\mu_i}$ on the product message as

$$\mathbf{R}_{\prod_{i\in S}} = ((\mu_{i_2}\cdot\mathbf{R}_{i_1} + \mathbf{R}_{i_2}\cdot\mathbf{G}_{n,2,m}^{-1}(-\mathbf{D}_{i_1}))\cdot\mathbf{G}_{n,2,m}^{-1}(-\mathbf{D}_{i_3}) + \mu_{i_1}\mu_{i_2}\cdot\mathbf{R}_{i_3})$$
$$\cdots + \mu_{i_1}\cdots\mu_{i_{|S|-1}}\mathbf{R}_{i_{|S|}} \qquad \text{where } \mathbf{D}_i = \mathbf{D}\cdot\mathbf{G}_{dn,\ell,m}^{-1}(\mathbf{U}_i\mathbf{G}_{n,2,m})$$

- $\mathsf{Permute}(\mathsf{pk}, \sigma, \boldsymbol{\mu}, \pi)$: Given the signature/message pair $(\sigma = \mathbf{R}, \boldsymbol{\mu})$ and a permutation π on set $[d]$, the permutation algorithm first checks the validity of the signature by running $\mathsf{Verify}(\mathsf{pk}, \sigma, \boldsymbol{\mu})$ and outputs $\perp$ if invalid. Otherwise compute and output a signature $\sigma_{\pi(\boldsymbol{\mu})}$ on the permuted message as

$$\mathbf{R}_{\pi(\boldsymbol{\mu})} = \mathbf{R}\cdot\mathbf{G}_{dn,\ell,m}^{-1}(\pi(\mathbf{I}_{dn})\cdot\mathbf{G}_{dn,\ell,m})$$

- $\mathsf{Verify}(\mathsf{pk}, \mu, \sigma, f)$: Given a message $\mu = f(\boldsymbol{\mu})$, a signature $\sigma = \mathbf{R}_\mu$ and a function f, the verification algorithm outputs 1 if both

$$|\sigma_\mu| \leq B \qquad \text{and} \qquad \mathbf{A}\mathbf{R}_\mu = \mathbf{D}_f + \mathbf{E}_\mu \bmod q$$

where $\mathbf{D}_f = \mathsf{PubEval}(\mathbf{D}, f)$.

5.1 Correctness and Parameters Selection

Before showing the correctness of the fully homomorphic SIMD signature scheme, we compute the exact δ-expanding coefficients for algorithms $(\mathsf{PubEval}, \mathsf{TrapEval})$ with respect to the relevant operations $(\mathsf{VAdd}, \mathsf{VMult}, \mathsf{Unpack}, \mathsf{CMult}, \mathsf{Permute})$ as follows:

Lemma 5.1. *For an integer d, the algorithms* $(\mathsf{PubEval}, \mathsf{TrapEval})$ *satisfy all of the following:*

- *Define* $\mathsf{VAdd}_d : \mathbb{Z}_p^d \times \mathbb{Z}_q^d \to \mathbb{Z}_q$ *as* $\mathsf{VAdd}(\boldsymbol{\mu}, \boldsymbol{a}) = \sum_{i=1}^d \mu_i a_i$*. The algorithms* $(\mathsf{PubEval}, \mathsf{TrapEval})$ *are* $(d\ell \log_\ell q)$*-expanding for the function* VAdd_d*.*

- *Define* $\mathsf{VMult}_d : \mathbb{Z}_p^d \times \mathbb{Z}_q^d \to \mathbb{Z}_q^d$ *as* $\mathsf{VMult}(\boldsymbol{\mu}, \boldsymbol{a}) = (\mu_1 a_1, ..., \mu_d a_d)$*. The algorithms* $(\mathsf{PubEval}, \mathsf{TrapEval})$ *are* $(\ell \log_\ell q)$*-expanding for the function* VMult_d*.*

- *Define* $\mathsf{Unpack}_d : \mathbb{Z}_p^d \times [d] \to \mathbb{Z}_p$ *as* $\mathsf{Unpack}(\boldsymbol{\mu}, i) = \mu_i$*. The algorithms* $(\mathsf{PubEval}, \mathsf{TrapEval})$ *are* $(\ell \log_\ell q)$*-expanding for the function* Unpack_d*.*

- *Define* $\mathsf{CMult}_d : \mathbb{Z}_p^d \times 2^{[d]} \to \mathbb{Z}_q^d$ *as* $\mathsf{CMult}(\boldsymbol{\mu}, S) = \prod_{i \in S} \mu_i$*. The algorithms* $(\mathsf{PubEval}, \mathsf{TrapEval})$ *are* $(m|S|(p-1)^{|S|}$
$\ell \log_\ell q)$*-expanding for the function* CMult_d*.*

- *Define* $\mathsf{Permute}_d : \mathbb{Z}_p^d \times \pi \to \mathbb{Z}_p^d$ *as* $\mathsf{Permute}(\boldsymbol{\mu}, \pi) = (\mu_{\pi(1)}, ..., \mu_{\pi(d)})$*. The algorithms* $(\mathsf{PubEval}, \mathsf{TrapEval})$ *are* $(\ell \log_\ell q)$*-expanding for the function* $\mathsf{Permute}$*.*

Proof. We examine each case in sequence.

The computation of $(\mathsf{PubEval}, \mathsf{TrapEval})$ for the function VAdd_d proceeds by multiplying

$$\mathbf{G}^{-1}_{dn,\ell,m}([a_1\mathbf{G}_{n,2,m}| \cdots |a_d\mathbf{G}_{n,2,m}]^\mathsf{T}) = [\mathbf{X}_1 \otimes \mathbf{I}_n| \cdots |\mathbf{X}_d \otimes \mathbf{I}_n]$$

where $\mathbf{X}_i \in [\ell]^{\log_\ell q \times \log q}$, for $i \in [d]$. Similarly as in Lemma 4.3, we can bound this term by $d\ell \log_\ell q$.

The computation of $(\mathsf{PubEval}, \mathsf{TrapEval})$ for the function VMult_d is by multiplying

$$\mathbf{G}^{-1}_{dn,\ell,m}\left(\begin{bmatrix} a_1\boldsymbol{g}_\ell \otimes \mathbf{I}_n & & \\ & \ddots & \\ & & a_d\boldsymbol{g}_\ell \otimes \mathbf{I}_n \end{bmatrix}\right)$$

where by using a similar worst-case analysis as before, we can bound this term by $\ell \log_\ell q$.

The unpack case is exactly the same as in Lemma 4.3.

The computation of $(\mathsf{PubEval}, \mathsf{TrapEval})$ for the function CMult_d works by first applying the unpacking algorithm for index $i \in S$, then calculating the procedure as shown in the scheme. We obtain the stated exact expansion coefficient for the function CMult by combining the results from Fact 4.2 and the unpacking case.

Lastly, the computation of $(\mathsf{PubEval}, \mathsf{TrapEval})$ for the function $\mathsf{Permute}$ goes by multiplying $\mathbf{G}^{-1}_{dn,\ell,m}(\pi(\mathbf{I}_{dn}) \cdot \mathbf{G}_{dn,\ell,m})$. By a similar analysis as in the VMult_d case, we obtain the stated bound $\ell \log_\ell q$. □

Lemma 5.2 (Correctness)**.** *The homomorphic signature scheme Σ described above is correct (cf. Definition 2.3) for admissible functions (cf. Definition 2.4).*

Proof. Correctness follows from the description of the algorithm $\mathsf{PubEval}$ for different admissible functions shown above, with the parameters given below. □

Parameter selection. Let λ be the security parameter. Our homomorphic signatures can be based on the hardness of the standard SIS problem, but in that case they will only support $\mathsf{poly}\log(\lambda)$-depth functions. To support functions of circuit depth $t = \mathsf{poly}(\lambda)$, we need to rely on some sub-exponential SIS hardness assumption. The parameters are set in Table 2.

Param	Description	Setting
λ	security parameter	
n	lattice row dimension	2^{λ^δ}
m	lattice column dimension	$n^{1+\delta}$
q	modulus	$n^{\tilde{\Theta}(t)}$
p	message range	$\omega(1)$
s	sampling algorithm and error width	$\omega(m \log q)$
B	signature bound	$\omega(2^t)$
β	SIS solution bound	$2^{\mathsf{poly}(\lambda)}$
ℓ	integer-base parameter	n

Table 2: Homomorphic Signatures for $\mathsf{poly}(\lambda)$-depth Parameters Setting

The same rules used to compute the parameters in the ABE setting also apply here. Also as discussed in [MR04], even if we set β to be super-polynomial (e.g., $\beta = 2^{\mathsf{poly}(\lambda)}, q > \beta$), SIS is still widely believed to be hard. Again, this is only necessary to support circuits of super-polylog (e.g., polynomial) depth.

5.2 Unforgeability Proof

We prove the unforgeability of our homomorphic signature scheme Σ in the following theorem.

Theorem 5.3. *Assuming the hardness of the* $\mathsf{SIS}_{q,n,m,\beta}$ *problem, the homomorphic signature scheme* Σ *described satisfies selective unforgeability as defined in Definition 2.5 (with modification specified in Remark 2.6).*

Proof. Assume there exists a PPT adversary $\mathcal{A}$ who wins the unforgeability security experiment, we construct a reduction $\mathcal{B}$ which uses the adversary $\mathcal{A}$ to break the SIS assumption. The description of the reduction $\mathcal{B}$ is as follows:

- **Invocation**: The reduction $\mathcal{B}$ is invoked on a random instance $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ of the $\mathsf{SIS}_{q,n,m,\beta}$ problem, and is asked to return a solution $\mathbf{R}^* \in \mathbb{Z}^{m\times m}$ such that $\mathbf{A}\cdot\mathbf{R}^* = 0 \pmod q$ and $0 \neq |\mathbf{R}^*| \leq \beta$.

- **Setup and signing query**: The reduction $\mathcal{B}$ invokes an adversary $\mathcal{A}$ and receives a challenge message $\boldsymbol{\mu} = (\mu_1, ..., \mu_d)$. It chooses a matrix $\mathbf{R} \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m},s}$ and sets $\mathbf{D} = \mathbf{AR} - \mathbf{E}_{\boldsymbol{\mu}}$, where $\mathbf{E}_{\boldsymbol{\mu}} = \left[\mu_1\mathbf{I}_n|\cdots|\mu_d\mathbf{I}_n\right]\cdot\mathbf{G}_{dn,\ell,m}$ is the encoding of the message $\boldsymbol{\mu}$. It then sends the verification key $\mathsf{pk} = (\mathbf{A},\mathbf{D})$ and the signature $\sigma_{\boldsymbol{\mu}} = \mathbf{R}$ to $\mathcal{A}$.

- **Forgery**: The reduction $\mathcal{B}$ receives from the adversary $\mathcal{A}$ a forgery tuple $(f, y', \sigma' = \mathbf{R}')$, where f is an admissible function and $y' \neq f(\boldsymbol{\mu})$. It first computes $\mathbf{R}_f \leftarrow \mathsf{TrapEval}(\boldsymbol{\mu}, \mathbf{A}, \mathbf{R}, f)$. By correctness of the homomorphic signature, we have $\mathbf{AR}' = \mathbf{AR}_f + (\mathbf{E}_{y'} - \mathbf{E}_{f(\mu)})\mathbf{G}$, where the gadget matrix $\mathbf{G}$ here can be indexed by either 2 or ℓ, depending on the forgery function f. By re-arranging the equation, we have

$$\mathbf{A}(\mathbf{R}' - \mathbf{R}_f)\cdot\mathbf{T}_{\mathbf{G}} = (\mathbf{E}_{y'} - \mathbf{E}_{f(\mu)})\mathbf{G}\cdot\mathbf{T}_{\mathbf{G}} = 0$$

  Therefore, the reduction $\mathcal{B}$ can output as solution to the SIS instance $\mathbf{A}$, the matrix $\mathbf{R}^* = (\mathbf{R}' - \mathbf{R}_f)\cdot\mathbf{T}_{\mathbf{G}}$.

Lemma 5.4. *Let* $(\mathsf{pk}_{\mathsf{real}}, \sigma_{\mathsf{real}})$ *be the output in the real execution, and* $(\mathsf{pk}_{\mathsf{sim}}, \sigma_{\mathsf{sim}})$ *be the output of simulated execution described above. We show that the two distributions are statistically indistinguishable.*

Proof. The differences between real execution and simulation can be summarized below:

- For the public key, in the real Setup, the matrix $\mathbf{A}$ is sampled from algorithm $\mathsf{TrapGen}$ along with its trapdoor $\mathbf{T_A}$, and the matrix $\mathbf{D}$ is sampled uniformly random from $\mathbb{Z}_q^{n\times m}$. In the simulated Setup, the matrix $\mathbf{A}$ is chosen randomly by the SIS generator without its trapdoor, and the matrix $\mathbf{D}$ is set to be $\mathbf{D} = \mathbf{AR} - \mathbf{E}_{\boldsymbol{\mu}}$, where $\mathbf{R} \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m},s}$ and $\mathbf{E}_{\boldsymbol{\mu}} = \left[\mu_1\mathbf{I}_n|\cdots|\mu_d\mathbf{I}_n\right]\cdot\mathbf{G}_{dn,\ell,m}$.
- For the signature, in the real case, σ_{real} is computed by the algorithm $\mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \mathbf{D} + \mathbf{E}_{\boldsymbol{\mu}}, s)$. In the simulated case, the signature σ_{sim} is pre-sampled from the distribution $\mathcal{D}_{\mathbb{Z}^{m\times m},s}$ in the setup phase.

We now argue that the distribution $(\mathbf{A}, \mathbf{D}, \mathbf{R}_{\mathsf{real}})$ in the real scenario is statistically indistinguishable from $(\mathbf{A}, \mathbf{AR}{-}\mathbf{E}_{\boldsymbol{\mu}}, \mathbf{R}_{\mathsf{sim}})$ in the simulated one. By the property of the algorithm $\mathsf{SamplePre}$ stated in Lemma 2.13, we know that $\mathbf{R}_{\mathsf{real}}$ is statistically indistinguishable from $\mathbf{R}_{\mathsf{sim}}$. Then by Lemma 2.9, $(\mathbf{A}, \mathbf{D})$ is statistically close to $(\mathbf{A}, \mathbf{AR})$. Combining the two claims, we have

$$(\mathbf{A}, \mathbf{D}, \mathbf{R}_{\mathsf{real}}) \approx (\mathbf{A}, \mathbf{AR} - \mathbf{E}_{\boldsymbol{\mu}}, \mathbf{R}_{\mathsf{sim}})$$

This completes the indistinguishability proof. □

Completing the proof. After arguing the indistinguishability of adversary $\mathcal{A}$'s view in the real and simulated scenarios, we now show that an adversary $\mathcal{A}$ that wins the homomorphic signature experiment can be used to solve the SIS assumption. By the property of $\mathsf{TrapEval}$ defined in Definition 4.1, we have

$$\mathsf{PubEval}(\mathbf{AR} - \mathbf{E}_{\boldsymbol{v}}, f) = \mathbf{AR}_f - \mathbf{E}_{f(\boldsymbol{\mu})}$$

Plugging in the above equation in the verification step where $y' \neq f(\boldsymbol{\mu})$, we have

$$\mathbf{AR}' = \mathbf{AR}_f - \mathbf{E}_{f(\boldsymbol{\mu})} + \mathbf{E}_{y'} \Rightarrow \mathbf{A}(\mathbf{R}' - \mathbf{R}_f)\cdot\mathbf{T_G} = 0$$

It remains to show that $(\mathbf{R}' - \mathbf{R}_f)\cdot\mathbf{T_G} \neq 0$, or equivalently $\mathbf{R}' \neq \mathbf{R}_f$. Per correctness, if $\mathbf{R}' = \mathbf{R}_f$, we have $\mathbf{AR}_f = \mathbf{AR}_f - \mathbf{E}_{f(\boldsymbol{\mu})} + \mathbf{E}_{f(\boldsymbol{\mu})}$, which contradicts the condition that $y' \neq f(\boldsymbol{\mu})$. Then, by setting the parameters as in Section 5.1, we complete the proof. □

We remark that for convenience our proof considers a "matrix" variant of the SIS problem, where the sought answer is a low-norm non-zero matrix in the right-kernel of the SIS challenge $\mathbf{A} \pmod q$. To find a single-vector SIS solution as traditionally required, the reduction $\mathcal{B}$ would simply select a non-zero column of $\mathbf{R}^*$ to get a low-norm non-zero vector in the right-kernel of $\mathbf{A}$, as required.

5.3 Adaptive Security

In this section we have intentionally restricted our attention to a limited security model, in order to focus on the SIMD aspects of our HS construction. We now discuss how to leverage our construction to achieve SIMD-enabled HS signatures with adaptive security.

As discussed in Remark 2.6, we can use similar transformations as mentioned in [GVW15b] to transform our SIMD HS construction to SIMD HS scheme proven secure in the adaptive multi-dataset model. The transformation is based on homomorphic trapdoor function (HTDF) and a regular signature scheme. In an HTDF, the evaluation of function can be computed publicly using the encoding of inputs, and privately using inputs. The security of HTDF is *claw-free*. The authors of [GVW15b] show how to construct HTDF for all circuits based on the SIS assumption. Therefore, we have the following informal theorem for adaptive multi-dataset model:

Theorem 5.5 (Informal)**.** *Assuming the hardness of SIS problem and the existence of a regular signature scheme, there exists an SIMD-enabled HS scheme that is existentially unforgeable in the adaptive multi-dataset model.*

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Gilbert [Gil10], pages 553–572.

[AFL16] Daniel Apon, Xiong Fan, and Feng-Hao Liu. Compact identity based encryption from LWE. Cryptology ePrint Archive, Report 2016/125, 2016. `http://eprint.iacr.org/2016/125`.

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 21–40. Springer, Heidelberg, December 2011.

[Ajt99] Miklós Ajtai. Determinism versus non-determinism for linear time RAMs (extended abstract). In *31st ACM STOC*, pages 632–641. ACM Press, May 1999.

[AL11] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In Catalano et al. [CFGN11], pages 17–34.

[AP10] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. *Theory of Computing Systems*, 48(3):535–553, 2010.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Garay and Gennaro [GG14], pages 297–314.

[ASI16] *ASIACRYPT 2016, Part II*, LNCS. Springer, Heidelberg, December 2016.

[Att14] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In Nguyen and Oswald [NO14], pages 557–577.

[Att16] Nuttapong Attrapadung. Dual system encryption framework in prime-order groups via computational pair encodings. In ASIACRYPT 2016, Part II [ASI16], pages 591–623.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

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

[BF11b] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In Catalano et al. [CFGN11], pages 1–16.

[BFKW09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 68–87. Springer, Heidelberg, March 2009.

[BFR13] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 863–874. ACM Press, November 2013.

[BFS14] Xavier Boyen, Xiong Fan, and Elaine Shi. Adaptively secure fully homomorphic signatures based on lattices. Cryptology ePrint Archive, Report 2014/916, 2014. `http://eprint.iacr.org/2014/916`.

[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 Nguyen and Oswald [NO14], pages 533–556.

[BL16] Xavier Boyen and Qinyi Li. Turing machines with shortcuts: Efficient attribute-based encryption for bounded functions. In *International Conference on Applied Cryptography and Network Security*, pages 267–284. Springer, 2016.

[Boy13] Xavier Boyen. Attribute-based functional encryption on lattices. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 122–142. Springer, Heidelberg, March 2013.

[BRF13] Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors. *45th ACM STOC*. ACM Press, June 2013.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.

[BV] Zvika Brakerski and Vinod Vaikuntanathan. Circuit-ABE from LWE: Unbounded attributes and semi-adaptive security. pages 363–384.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 535–554. Springer, Heidelberg, February 2007.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, Heidelberg, May 2013.

[CFGN11] Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors. *PKC 2011*, volume 6571 of *LNCS*. Springer, Heidelberg, March 2011.

[CFGN14] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic MACs for arithmetic circuits. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 538–555. Springer, Heidelberg, March 2014.

[CGW15] Jie Chen, Romain Gay, and Hoeteck Wee. Improved dual system ABE in prime-order groups via predicate encodings. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 595–624. Springer, Heidelberg, April 2015.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Gilbert [Gil10], pages 523–552.

[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 Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, Heidelberg, May 2004.

[FMNP16] Dario Fiore, Aikaterini Mitrokotsa, Luca Nizzardo, and Elena Pagnin. Multi-key homomorphic authenticators. In ASIACRYPT 2016, Part II [ASI16], pages 499–530.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GG14] Juan A. Garay and Rosario Gennaro, editors. *CRYPTO 2014, Part I*, volume 8616 of *LNCS*. Springer, Heidelberg, August 2014.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In Pointcheval and Johansson [PJ12], pages 465–482.

[Gil10] Henri Gilbert, editor. *EUROCRYPT 2010*, volume 6110 of *LNCS*. Springer, Heidelberg, May 2010.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Boneh et al. [BRF13], pages 555–564.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

[GV15] Sergey Gorbunov and Dhinakaran Vinayagamurthy. Riding on asymmetry: Efficient ABE for branching programs. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 550–574. Springer, Heidelberg, November / December 2015.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Boneh et al. [BRF13], pages 545–554.

[GVW15a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523. Springer, Heidelberg, August 2015.

[GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, Heidelberg, December 2013.

[HAO15] Ryo Hiromasa, Masayuki Abe, and Tatsuaki Okamoto. Packing messages and optimizing bootstrapping in GSW-FHE. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 699–715. Springer, Heidelberg, March / April 2015.

[HS14] Shai Halevi and Victor Shoup. Algorithms in HElib. In Garay and Gennaro [GG14], pages 554–571.

[HS15] Shai Halevi and Victor Shoup. Bootstrapping for HElib. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 641–670. Springer, Heidelberg, April 2015.

[HW13] Susan Hohenberger and Brent Waters. Attribute-based encryption with fast decryption. In Kurosawa and Hanaoka [KH13], pages 162–179.

[JMSW02] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *Cryptographers Track at the RSA Conference*, pages 244–262. Springer, 2002.

[KH13] Kaoru Kurosawa and Goichiro Hanaoka, editors. *PKC 2013*, volume 7778 of *LNCS*. Springer, Heidelberg, February / March 2013.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 146–162. Springer, Heidelberg, April 2008.

[LOS$^+$10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Gilbert [Gil10], pages 62–91.

[LW12] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In Safavi-Naini and Canetti [SNC12], pages 180–198.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In Pointcheval and Johansson [PJ12], pages 700–718.

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004.

[NO14] Phong Q. Nguyen and Elisabeth Oswald, editors. *EUROCRYPT 2014*, volume 8441 of *LNCS*. Springer, Heidelberg, May 2014.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 191–208. Springer, Heidelberg, August 2010.

[Pei16] Chris Peikert. How (not) to instantiate ring-lwe. In *International Conference on Security and Cryptography for Networks*, pages 411–430. Springer, 2016.

[PJ12] David Pointcheval and Thomas Johansson, editors. *EUROCRYPT 2012*, volume 7237 of *LNCS*. Springer, Heidelberg, April 2012.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 422–439. Springer, Heidelberg, March 2012.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Heidelberg, August 1984.

[SNC12] Reihaneh Safavi-Naini and Ran Canetti, editors. *CRYPTO 2012*, volume 7417 of *LNCS*. Springer, Heidelberg, August 2012.

[SV14] Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, pages 1–25, 2014.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.

[Wat12] Brent Waters. Functional encryption for regular languages. In Safavi-Naini and Canetti [SNC12], pages 218–235.

[Xag13] Keita Xagawa. Improved (hierarchical) inner-product encryption from lattices. In Kurosawa and Hanaoka [KH13], pages 235–252.

[Yam17] Shota Yamada. Asymptotically compact adaptively secure lattice ibes and verifiable random functions via generalized partitioning techniques. Cryptology ePrint Archive, Report 2017/096, 2017. `http://eprint.iacr.org/2017/096`.