ATTRIBUTE-BASED ENCRYPTIONS AND FORMAL VERIFICATION OF LATTICE-BASED CRYPTOGRAPHY

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Xiong Fan

December 2019

ATTRIBUTE-BASED ENCRYPTIONS AND FORMAL VERIFICATION OF LATTICE-BASED CRYPTOGRAPHY

Xiong Fan, Ph.D.

Cornell University 2019

Since the early works of Ajtai (STOC'96) and Regev (STOC'05), lattice-based cryptography has proven to be a powerful building block in cryptography. My research focuses on further exploring the expressive power of lattice-based cryptography, as well as formal verification of lattice-based cryptographic schemes.

Deniable encryption (Canetti et al. CRYPTO '97) is an intriguing primitive that provides a security guarantee against not only eavesdropping attacks as required by semantic security, but also stronger coercion attacks performed after the fact. The concept of deniability has later demonstrated useful and powerful in many other contexts, such as leakage resilience, adaptive security of protocols, security against selective opening attacks and coercion resistance in voting systems. Despite its conceptual usefulness, our understanding of how to construct deniable primitives under standard assumptions is restricted. We construct a flexibly bi-deniable Attribute-Based Encryption (ABE) scheme for all polynomial-size Branching Programs from Learning With Errors assumption (Regev STOC'05).

Attribute based encryption (ABE) is an advanced encryption system with a built-in mechanism to generate keys associated with functions which in turn provide restricted access to encrypted data. Most of the known candidates of attribute based encryption model the functions as circuits. This results in sig-

nificant efficiency bottlenecks, especially in the setting where the function associated with the ABE key admits a RAM program whose runtime is sublinear in the length of the attribute. We study the notion of attribute based encryption for random access machines (RAMs), introduced in the work of Goldwasser, Kalai, Popa, Vaikuntanathan and Zeldovich (Crypto 2013) and present a construction satisfying sublinear decryption complexity assuming Learning With Errors.

We then introduce a symbolic approach for proving security of cryptographic constructions based on the Learning With Errors assumption (Regev, STOC 2005). Such constructions are instances of lattice-based cryptography and are extremely important due to their potential role in post-quantum cryptography. Our approach combines a computational logic, deducibility problems, a standard tool for representing the adversary's knowledge and the Dolev-Yao model. The computational logic is used to capture (indistinguishability-based) security notions and drive the security proofs whereas deducibility problems are used as side-conditions to control that rules of the logic are applied correctly. We then use AutoLWE, an implementation of the logic, to deliver very short or even automatic proofs of several emblematic constructions. The main technical novelty beyond AutoLWE is a set of (semi-)decision procedures for deducibility problems, using extensions of Gröbner basis computations for subalgebras in the (non-)commutative setting (instead of ideals in the commutative setting). Our procedures cover the theory of matrices, which is required for lattice-based assumption, as well as the theory of non-commutative rings, fields, and Diffie-Hellman exponentiation, in its standard, bilinear and multilinear forms.

BIOGRAPHICAL SKETCH

Xiong (Leo) grew up in Chongqing, China. In 2010 and 2013, he received a B.S. from Sichuan University in Mathematics and a M.S. from Chinese Academy of Science in Computer Science respectively. He finished his Ph.D. at Cornell University in Computer Science in 2019.

To my family

ACKNOWLEDGEMENTS

First of all, I would like to deeply thank my advisor, Elaine Shi, for her invaluable guidance in the past several years that leads me toward being an independent researcher. She always encourages me to pursue my own research interests and directions. Elaine has a sharp intuition and is full of amazing ideas. Every time I felt frustrated when got stuck on some problems, she can always magically bring me back on track and proceed forward. Her way of doing research, presenting research, thinking about research, as well as her passionate on research will continue to guide me in the future.

I am indebted to my mentors at Yahoo! Labs, Juan Garay and Payman Mohassel, for showing me a wonderful summer in silicon valley, sharing generously and sincerely their opinions and wisdom whenever I solicited for suggestions in research.

I want to thank my mentor at Bell Labs, Vlad Kolesnikov for a lovely summer in New Jersey. I had many inspiring discussions with him. His great experiences helped my learn knowledge and develop interest on other areas such as secure computation.

I am grateful to the cryptography group at IBM for warmly taking me into their family. We had many interesting discussions on different objects during lunch. I enjoyed greatly my collaboration with Shai Halevi and Craig Gentry, whose clear views, sharp thoughts and gentle nature made our discussions both thought-stimulating and delightful.

I would like also to thank all my co-authors: Prabhanjan Ananth, Daniel Apon, Gilles Barthe, Chaya Ganesh, Joshua Gancher, Benjamin Grégoire, Charlie Jacomme, Vlad Kolesnikov, Feng-Hao Liu, Elaine Shi, Qiang Tang and Zhedong Wang. Collaborating with them not only results in several interesting pa-

pers, but also teaches me a lot along the way. My sincere appreciation goes to Andrew Myers and Rafael Pass for serving on my thesis committee and for many valuable comments.

The last paragraph is dedicated to my beloved parents Dali Fan and Hong Jiang, wife Chuchun Liang. Their unconditional support makes this thesis possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Attribute-Based Encryption

Attribute-based encryption (ABE) [117] is a powerful paradigm that provides a mechanism for controlled accessing encrypted data. Unlike a traditional encryption scheme, in an attribute-based encryption scheme, an authority can generate a constrained key SK_P for a program P such that it can decrypt an encryption of message μ, associated with attribute x, only if the condition $P(x) = 0$ is satisfied. The last decade of research in this area [117, 85, 106, 84, 126, 94, 127, 79, 68, 77, 31, 70, 128, 81, 41] has led to several useful applications including verifiable computation [107] and reusable garbled circuits [78]. Special cases of ABE, such as identity-based encryption [30, 125, 60, 37], and generalizations of ABE, such as functional encryption [33, 104, 67], have also been extensively studied.

Current known constructions of ABE offer different flavors of efficiency guarantees and can be based on various cryptographic assumptions. Barring few exceptions, all these constructions [85, 126, 95, 80, 31, 81] model the program associated with the constrained keys as circuits. Real-world programs, however, are composed in the so-called Random Access Machine (RAM) model. In this paper, we consider the natural question of constructing attributed-based encryption scheme for RAM programs.

As in the circuit setting, an attribute-based encryption scheme for RAM programs consists of the setup, key generation, encryption and decryption algorithms. The key generation algorithm takes as input the master secret key, pro-

gram P, database D and produces an attribute key associated with (P, D). The encryption algorithm takes as input attribute x, secret message μ and produces a ciphertext. Decrypting this ciphertext using the key of (P, D) yields the secret message μ if and only if $P^D(x) = 0$.

Toward constructing attribute-based encryption for RAMs, a naïve approach is to convert RAM programs generically to circuits: a RAM program initialized with N words of memory and running in time T can be converted to a circuit of size $O((N + T) \cdot T)$ and depth T. Thus, the approach via naïve RAM-to-circuit conversion would incur a $(N+T)\cdot T$ multiplicative factor in the decryption time. In this paper, we are interested in the common case when T is *sublinear* in N, e.g., imagine that the RAM is initialized with a large database with N entries and the RAM program models a binary search on the database. Goldwasser et al. [77] gave the first feasibility result of ABE for RAM programs with sub-linear decryption time based on the existence of extractable witness encryption and succinct non-interactive arguments of knowledge (SNARK). Recent works [69, 34, 24], however, have brought into question the veracity of the assumptions of extractable witness encryption and SNARKs.

Since building ABE for RAMs on solid cryptographic foundations is an important problem, we ask the following natural question:

> *Is there an ABE for RAMs scheme with sublinear decryption overhead based on standard assumptions? More specifically, we would like the decryption overhead to be* $o(N) \cdot \mathsf{poly}(T, \lambda)$ *where* λ *is the security parameter.*

Deniable encryption, introduced by Canetti et al. [43] at CRYPTO 1997, is an intriguing primitive that allows Alice to privately communicate with Bob in a

way that resists not only eavesdropping attacks as required by semantic security, but also stronger *coercion attacks* performed after the fact. An eavesdropper Eve stages a cocercion attack by additionally approaching Alice (or Bob, or both) *after* a ciphertext is transmitted and demanding to see all secret information: the plaintext, the random coins used by Alice for encryption, and any private keys held by Bob (or Alice) related to the ciphertext. In particular, Eve can use this information to "fully unroll" the *exact transcript* of some deterministic decryption procedure purportedly computed by Bob, as well as verify that the exact coins and decrypted plaintext in fact produce the coerced ciphertext. A secure deniable encryption scheme should maintain privacy of the sensitive data originally communicated between Alice and Bob under the coerced ciphertext (instead substituting a benign yet *convincing* plaintext in the view of Eve), even in the face of such a revealing attack and even if Alice and Bob may not interact during the coercion phase.

Historically, deniable encryption schemes have been challenging to construct. Under standard assumptions, Canetti et al. [43] constructed a sender-deniable[1] PKE where the distinguishing advantage between real and fake openings is an inverse polynomial depending on the public key size. But it was not until 2011 that O'Neill, Peikert, and Waters [105] proposed the first constructions of bi-deniable PKE with *negligible* deniability distinguishing advantage: from simulatable PKE generically, as well as from Learning with Errors (LWE [112]) directly.

Concurrently, Bendlin et al. [25] showed an inherent limitation: any non-interactive public-key encryption scheme may be receiver-deniable (resp. bi-

[1] We differentiate between sender-, receiver-, and bi-deniable schemes. A bi-deniable scheme is both sender- and receiver-deniable.

deniable) only with *non-negligible* $\Omega(1/\mathsf{size}(\mathsf{pk}))$ distinguishing advantage in the deniability experiment. Indeed, O'Neill et al. [105] bypass the impossibility result of [25] by working in the so-called *flexible*[2] model of deniability. In the flexible of deniability, private keys sk are distributed by a central key authority. In the event that Bob is coerced to reveal a key sk that decrypts chosen ciphertext CT^*, the key authority distributes a *faking key* fk to Bob, which Bob can use to generate a fake key sk^* (designed to behave identically to sk except on ciphertext CT^*). If this step is allowed, then O'Neill et al. demonstrate that for their constructions, Eve has at most negligible advantage in distinguishing whether Bob revealed an honest sk or fake sk^*.

A major breakthrough in deniable encryption arrived with the work of Sahai and Waters [116], who proposed the first sender-deniable PKE with negligible distinguishing advantage from indistinguishability obfuscation ($i\mathcal{O}$) for $\mathsf{P/poly}$ [66]. The concept of deniability has been demonstrated useful in the contexts of leakage resilience [55], adaptive security for protocols, and as well as deniable computation (or algorithms) [45, 54, 73]. In addition to coercion resistance, a bi-deniable encryption scheme is a non-committing encryption scheme [44], as well as a scheme secure under selective opening (SOA) attacks [23], which are of independent theoretical interest.

Recently, De Caro, Iovino, and O'Neill [46] gave various constructions of deniable *functional encryption*. First, they show a generic transformation of any IND-secure FE scheme for circuits into a flexibly receiver-deniable FE for circuits. Second, they give a direct construction of receiver-deniable FE for Boolean formulae from bilinear maps. Further, in the stronger *multi-distributional* model

[2]We borrow the name "flexible" from Boneh, Lewi, and Wu [32] as the original term "multi-distributional" of O'Neill et al. [105] is used to define a slightly different security property in the recent work by De Caro et al. [46] than we achieve here.

of deniable functional encryption – where there are special "deniable" set-up and encryption algorithms in addition to the plain ones, and where under coercion, it may non-interactively be made to seem as only the normal algorithms were used – De Caro et al. [46] construct receiver-deniable FE for circuits under the additional (powerful) assumption of different-inputs obfuscation (*diO*).

De Caro et al. [46] also show (loosely speaking) that any receiver-deniable FE implies SIM-secure FE for the same functionality. Following [46], we also emphasize that deniability for functional encryption is a **strictly stronger** property than SIM security, since fixed coerced ciphertexts must decrypt correctly and benignly *in the real world*.

Despite the apparent theoretical utility in understanding the extent to which cryptographic constructions are deniable, our current knowledge of constructing such schemes from standard lattice assumptions is still limited. With the powerful assumption of indistinguishability obfuscation (*iO*), we can obtain at least fully secure sender-deniable PKE and computation [45, 54, 73], or as mentioned above, even a multi-distributional receiver-deniable FE for all circuits from the even stronger assumption of *diO* The second question we ask is

Is there a deniable ABE from lattices that can support expressive functions?

1.2 Formal Verification

Formal methods, and in particular formal verification, have long been used for building and checking mathematical claims of correctness or security for small but possibly very complex to moderately large and complex systems. In con-

trast to pen-and-paper counterparts, formally verified claims deliver higher assurance and independently verifiable proofs that can be replayed by third parties. Over the last 20 years, formal methods have been applied successfully to analyze the security of cryptographic protocols in the Dolev-Yao model [59], an idealized model in which cryptographic constructions are treated algebraically. By abstracting away from the probabilistic nature of cryptographic constructions, the Dolev-Yao model has served as a suitable and practical foundation for highly or fully automated tools [11, 28, 118]. These tools have subsequently been used for analyzing numerous cryptographic protocols, including recently TLS 1.3. [53, 93]. Unfortunately, the Dolev-Yao model is focused on cryptographic protocols and cannot be used for reasoning about cryptographic primitives. A related approach is to use so-called refinement types (a.k.a. logical assertions) for reasoning about implementations written in a functional programming language [121]; this approach has also been used for analyzing TLS 1.3. [26, 56], but is also primarily limited to cryptographic protocols.

An alternative approach is to develop symbolic methods that reason directly in the computational model. This approach applies both to primitives and protocols, and instances of this approach have been instrumented in tools such as CertiCrypt [20], CryptHOL [96] CryptoVerif [29], EasyCrypt [21, 18], and FCF [111] (see also [88, 16] for further approaches not supported by tools). However, these tools require significant user interaction and expertise, in particular when used for reasoning about cryptographic primitives.

A promising approach for analyzing cryptographic primitives in the computational model is to combine computational logics and symbolic tools from the Dolev-Yao model. Prior work has demonstrated that this approach works

well for padding-based (combining one-way trapdoor permutations and random oracles) [15] and pairing-based cryptography [22]. Broadly speaking, computational logics formalize game-playing security proofs; each step of the proof corresponds to a hop, and symbolic side-conditions are used to ensure the validity of the hop. More specifically, computational logics, which can be seen as specializations of [16], are used to capture computational security goals and to drive security proofs whereas side-conditions use symbolic tools such as deducibility and static equivalence to guarantee that the rules of the logic are applied correctly. In particular, a key idea of this approach is to use deducibility for controlling the application of rules for performing reductions to hardness assumptions, and for performing optimistic sampling, a particularly common and useful transformation which simplifies probabilistic experiments by allowing to replace, under suitable circumstances, sub-computations by uniform samplings.

The use of deducibility in side conditions, as opposed to arbitrary mathematical conditions, is a necessary step for automating application of proof rules, and more generally for automating complete proofs. However, the interest of this approach is conditioned by the ability to check the validity of deducibility problems. The problem of deciding deducibility has been studied extensively in the context of symbolic verification in the Dolev-Yao model, where deducibility formalizes the adversary knowledge [97, 101, 108, 92, 119, 114, 49, 115]. This line of work has culminated in the design and implementations of decision procedures for classes of theories that either have some kind of normal form or satisfy a finite variant property. However, existing decidability results are primarily targeted toward algebraic theories that arise in the study of cryptographic protocols. In contrast, deducibility problems for cryptographic constructions require to reason about mathematical theories that may not have a natural notion

of normal form or satisfy the finite variant property.

Thus, a main challenge for computational logics based on deducibility problems is to provide precise and automated methods for checking the latter. There are two possible approaches to address this challenge:

- heuristics: rather than deciding deducibility, one considers weaker conditions that are easier for verification. As demonstrated with AutoG&P, such an approach may work reasonably well in practice. However, it is not fully satisfactory. First, the heuristics may be incomplete and fail to validate correct instances. Second, advanced proof rules that perform multiple steps at once, and proof search procedures, which explores the space of valid derivations, become unpredictable, even for expert users.
- (semi-)decision procedures based on computational mathematics: in this approach, one provides reductions from deducibility problems to computational problems in the underlying mathematical setting. Then, one can reuse (semi-)decision procedures for the computational problems to verify deducibility problems. This approach offers some important advantages. First, it eliminates a potential source of incompleteness, and in particular the possibility that a proof step fails. Second, it is more predictable. Predictability is very important when a high level of automation is sought. Indeed, automation is often achieved through advanced tactics. When they involve multiple heuristics, the outcome of advanced tactics cannot be anticipated, which is a major hurdle to the adoption of formal verification tools. Third, it formalizes connections between known mathematical problems, which may have been extensively studied, and verification problems that may arise for the first time. Lastly, it encourages reusing

existing algorithms and implementations.

The idea using methods from computational mathematics to reason about deducibility is natural. However, we are not aware of prior work that provide answers to the following question:

Is there a connection between computational mathematics and the use of deducibility in a computational logic?

1.3 Our Contributions

In this thesis, we answer the all the above questions affirmatively.

1.3.1 Attribute-Based Encryption for RAMs

We construct an ABE scheme for RAMs with sub-linear decryption overhead from the Learning With Errors (LWE) assumption [112]. Henceforth we assume that the scheme is parameterized with N and T which denote the size of the database and the upper bound on the runtime of the RAM respectively, and a security parameter denoted by λ. Our construction achieves the following:

- There is an initial setup phase that generates a global public parameter of size $\mathsf{poly}(N, T, \lambda)$ and master secret key of size $\mathsf{poly}(T, \lambda)$.
- Anyone that has access to the public parameters can encrypt a message μ to an attribute x of size λ — later x will serve as an input to a RAM

program. The encryption time and ciphertext size is upper bounded by $\mathsf{poly}(T, \lambda)$.

- An authority with master secret key can generate a decryption key $\mathsf{SK}_{P,D}$ given the description of a RAM program P (where the description will include the RAM's next instruction circuit) and a possibly long attribute vector denoted by D of size N, and the size of the secret key $\mathsf{SK}_{P,D}$ is upper bounded $\mathsf{poly}(T, N, \lambda)$.
- Finally, given the ciphertext CT_x that is associated with the attribute x, anyone with the public parameters and the decryption key $\mathsf{SK}_{P,D}$ can decrypt the plaintext message μ if $P^D(x) = 0$; and importantly decryption time is $\mathsf{poly}(T, \lambda)$, i.e., independent of the RAM's initial memory size N. For security, we show that an adversary learns nothing about the encrypted plaintext μ if he does not possess any SK_P such that $P^D(x) = 0$.

More formally, our main theorem is the following:

Theorem 1.3.1 (ABE for RAMs [7])**.** *Assuming the hardness of the Learning With Errors problem (with sub-exponential modulus)*[3]*, there exists an ABE scheme for RAMs with* $\mathsf{poly}(T, \lambda)$ *decryption efficiency, i.e., independent of* N*.*

Moreover, (i) the cost of generating public parameters is $\mathsf{poly}(T, \lambda)$*, i.e., independent of* N*, (ii) the cost of generating secret keys is* $\mathsf{poly}(N, T, \lambda)$ *and, (iii) the cost of generating ciphertexts is* $\mathsf{poly}(T, \lambda)$*.*

While the construction in the above theorem has decryption complexity proportional to the worst case running time of the RAM programs, we can transform this scheme into another scheme where the decryption complexity is *input-*

[3] All known lattice-based ABE for circuits [31] are based on the same assumption.

specific. This is performed by running $\log T$ copies of the scheme by setting the worst case runtime of the first scheme to be 2, second scheme to be 2^2, so on and the runtime of the $(\log T)$-th scheme is set to be T. This idea has been used in prior works (for instance [77]). Note that this increases the size of the public parameters, keys and ciphertexts by a multiplicative factor of $\log T$.

Comparison with [77]. As mentioned earlier, [77] also achieves ABE for RAMs with sub-linear decryption complexity from exotic assumptions. The only drawback in our scheme in comparison with [77] is that the parameters in the construction of [77], specifically the public parameters, ciphertext size and the key sizes do not grow with the maximum time bound. On the other hand, our parameters do grow with the maximum time bound T. There is evidence to suggest that an ABE for RAMs scheme whose parameters do not grow with the maximum time bound can only be based on strong cryptographic assumptions. In particular such a notion would imply succinct randomized encodings for Turing machines [27, 8][4]; a notion, despite numerous attempts, we don't yet know how to build from well-studied assumptions.

Comparison with Circuit-Based Schemes. We compare the parameters we obtain in our scheme with the parameters obtained in the naive approach of RAM-to-circuit conversion and then applying previously known ABE for circuits schemes. Refer to Figure 1.1.

We compare the parameters in our work with previous works. The lattice dimension (n, m, q) is asymptotically the same in all three approaches. The $\tilde{O}$ no-

[4]The works [27, 8] show implication of ABE for Turing machines (as defined in [8]) to succinct randomized encodings (Appendix A.5 in [27]

Schemes	# of $\mathbb{Z}_q^{n\times m}$ Public key	# of $\mathbb{Z}_q^m$ Ciphertext of (x, μ)	Size of Key of (P, D)	Decryption complexity
Via ABE for circuits [31]	$\tilde{O}(\|x\|)$	$\tilde{O}(\|x\|)$	$\|P\| + N + \left(\#\text{ of } \mathbb{Z}_q^{n\times m}\right)^{\tilde{O}(1)}$	$\tilde{O}((T+N)T)$
Via ABE for circuits [41]	$\tilde{O}(1)$	$\tilde{O}(\|x\|)$	$\|P\| + N + \left(\#\text{ of } \mathbb{Z}_q^{n\times m}\right)^{\tilde{O}(1)}$	$\tilde{O}((T+N)T)$
Our Work for RO-RAMs	$\tilde{O}(\|x\|+T)$	$\tilde{O}(\|x\|+T)$	$\|P\| + \left(\#\text{ of } \mathbb{Z}_q^{n\times m}\right)^{\tilde{O}(TN)}$	$\tilde{O}(T)$
Our Work for RAMs	$\tilde{O}(\|x\|+T)$	$\tilde{O}(\|x\|+T)$	$\|P\| + \left(\#\text{ of } \mathbb{Z}_q^{n\times m}\right)^{\tilde{O}(T(N+T))}$	$\tilde{O}(T)$

Figure 1.1: Comparison Table

tation suppresses poly-logarithmic factors (in N and T). The encryptor takes an auxiliary input x and the key generator takes as input a program P and a database D of size N. The decryption complexity is calculated in terms of vector-matrix multiplication over $\mathbb{Z}_q$. The attribute key is generated for a RAM program P with worst case runtime to be T and it takes time t to compute on D. In previous works, an attribute key for P is generated by first transforming it into a circuit of size $(T + N)$ and depth T and then generating an attribute key for the resulting circuit.

While the key sizes in our scheme are larger than the ones obtained via circuit ABE schemes, our scheme has the following advantage: since the same decryption keys, once generated, can be applied to (unbounded polynomially) many ciphertexts, the cost of key generation and its size can be *amortized over multiple decryption queries*. This is especially useful in scenarios, where a client can perform a one-time cost of generating keys and sending it over to the server during the offline phase and during the online phase, can verifiably delegate multiple computations by suitably sending encryptions of its inputs; note that

in this scenario, we are only interested in verifying whether the server has performed the computation correctly and not hiding the computation itself.

Dual ABE for RAMs. We also consider an alternate notion of ABE for RAMs, that we call dual ABE for RAMs. In this notion, the database is part of the ciphertext and not the key. That is, the key generation procedure now only takes as input the master secret key and the RAM program P while the encryption procedure takes as input the database D, the auxiliary input x (in the technical section, we consider x to be part of D) and the secret message μ. As before, we require that it should be possible to recover μ if indeed $P^D(x) = 0$.

We demonstrate a construction of dual ABE for RAMs, also based on the learning with errors problem with the same decryption efficiency as stated in Theorem 1.3.1.

1.3.2 Deniable ABE for Branching Program

To answer our second question, we further narrow this gap by investigating a richer primitive – attribute-based encryption (ABE) [86, 31, 83] – *without* the use of obfuscation as a black box primitive. We hope that the techniques developed in this work can further shed light on deniability for even richer schemes such as functional encryption [33, 66, 31, 81] under standard assumptions. Our main contribution is the construction of a flexibly bi-deniable ABE for poly-sized branching programs (which can compute NC^1 via Barrington's theorem [14]) from the standard Learning with Errors assumption [112].

Theorem 1.3.2 (Deniable ABE)**.** *Under the standard* LWE *assumption, there is a flex-*

ibly bi-deniable attribute-based encryption scheme for all poly-size branching programs.

Recall that in an attribute-based encryption (ABE) scheme for a family of functions $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$, every secret key sk_f is associated with a predicate $f \in \mathcal{F}$, and every ciphertext ct_x is associated with an attribute $\boldsymbol{x} \in \mathcal{X}$. A ciphertext ct_x can be decrypted by a given secret key sk_f to its payload message m only when $f(x) = 0 \in \mathcal{Y}$. Informally, the typical security notion for an ABE scheme is *collusion resistance*, which means no collection of keys can provide information on a ciphertext's message, if the individual keys are not authorized to decrypt the ciphertext in the first place. Intuitively, a bi-deniable ABE must provide both collusion and coercion resistance.

Other contributions of this work can be summarized as:

- A new form of the Extended Learning with Errors (eLWE) assumption [105, 5, 36], which is convenient in the context of Dual Regev type ABE/FE schemes that apply the Leftover Hash Lemma [58] in their security proofs.
- An explicit, tightened noise growth analysis for lattice-based ABE for branching programs. Prior work used the loose l_∞ norm to give a rough upper bound, which is technically insufficient to achieve deniability using our proof techniques. (We require matching upper *and lower* bounds on post-evaluation noise sizes.)

The eLWE assumption above is roughly the standard LWE assumption, but where the distinguisher also receives "hints" on the LWE sample's noise vector $\boldsymbol{e}$ in the form of inner products, i.e., distributions $\left\{\mathbf{A}, \boldsymbol{b} = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{e}^\mathsf{T}, \boldsymbol{z}, \langle \boldsymbol{z}, \boldsymbol{e} \rangle\right\}$ where (intuitively) $\boldsymbol{z}$ is a decryption key in the real system (which are denoted

r elsewhere). Our contribution here is a new reduction from the standard LWE assumption to our correlated variant of extended-LWE, eLWE^{+}, where the adversary requests arbitrary correlations (expressed as a matrix $\mathbf{R}$) between the hints, in the case of a prime poly-size modulus with noise-less hints. We show this by extending the LWE to eLWE reduction of Alperin–Sheriff and Peikert [5] to our setting.

1.3.3 Formal Verification

We propose symbolic methods for proving security of lattice-based cryptographic constructions. These constructions constitute a prime target for formal verification, due to their potential applications in post-quantum cryptography and their importance in the ongoing NIST effort to standardize post-quantum constructions; see e.g., [110] for a recent survey of the field.

We define a logic for proving computational security of lattice-based cryptographic constructions. The logic follows the idea of combining computational proof rules with symbolic side-conditions, as in [15, 22]. One important feature of our logic is that the proof rule for assumptions supports information-theoretic and computational assumptions that are stated using adversaries with oracle accesses. This extension is critical to capture (advanced cases of) the Leftover Hash Lemma [89]. The Leftover Hash Lemma is a powerful information-theoretical tool which allows to replace, under suitable conditions, a subcomputation by a sampling from a uniform distribution. The Leftover Hash Lemma is widely used in cryptographic proofs, in particular in the setting of lattice-based cryptography. We implement our logic in a tool

called AutoLWE (https://github.com/autolwe/autolwe), and use the tool for proving (indistinguishability-based) security for several cryptographic constructions based on the Learning with Errors (LWE) assumption [112].

More specifically, our examples include: dual Regev PKE [75], MP-PKE [100], ABB-(H)IBE [2] and IPE [3]. All of our mechanized proofs are realistically efficient, running in at most three seconds (Fig. 5.13); efficiency in this setting is usually not an issue, since cryptographic constructions typically induce small instances of the deducibility problem. Recent progress on more advanced cryptographic constructions based on lattices, like attribute-based encryption [31] and predicate encryption [81], are closely related to both the structure of the schemes and strategy in the proofs in [75, 2, 3]. The MP-PKE [100] inspires development in some lattice-based constructions, like homomorphic encryption [6] and deniable attribute-based encryption [10].

1.4 Additional Related Work

We describe additional related work in this part.

1.4.1 Attribute-Based Encryption

The constructions of ABE systems have a rich literature. The seminal result of Goyal, Pandey, Sahai and Waters [85] presented the first construction of ABE for boolean formulas from bilinear DDH assumption. Since then, several prominent works achieved stronger security guarantees [95], better efficiency or design guarantees [128, 12, 1] and achieving stronger models of ABE

for a restricted class of functions [91]. The breakthrough work of Gorbunov, Vaikuntanathan and Wee [80] presented the first construction of ABE for all polynomial-sized circuits assuming learning with errors. Following this, several works [31, 41] improved this result in terms of efficiency and also considering stronger security models [80]. In addition to [77], there are a few works that consider ABE in other models of computation. Waters [127] proposed a construction of functional encryption for regular languages and subsequently Agarwal and Singh [4] constructed reusable garbled finite automata from LWE. Ananth and Sahai [9] construct functional encryption for Turing machines assuming sub-exponentially secure functional encryption for circuits. Deshpande et al. [57] present an alternate construction of attribute based encryption for Turing machines under the same assumptions. In work [124], the authors constructed the first decentralized ABE from lattices, where the secret keys can be generated by several different authorities and decryption process takes in several secret keys.

In [63], Fan and Tang proposed a new notion of distributed public key functional encryption, in which the key generation procedure generates n different shares $\{\mathsf{SK}_i^f\}_{i\in[n]}$ instead of a single functional decryption key SK^f . The decryption of a ciphertext CT (an encryption of a message m) under a function f requires first the decryption under the functional key shares $s_i \leftarrow \mathsf{Dec}(\mathsf{SK}_i^f, \mathsf{CT})$ for all $i \in [n]$. These shares $\{s_i\}_{i\in[n]}$ are then used to reconstruct $f(m)$.

1.4.2 Automated Security Proofs

Corin and den Hartog [51] show chosen plaintext security of ElGamal using a variant of a general purpose probabilistic Hoare logic. In a related spirit, Courant, Daubignard, Ene, Lafourcade and Lakhnech [52] propose a variant of Hoare logic that is specialized for proving chosen plaintext security of padding-based encryption, i.e., public-key encryption schemes based on one-way trap-door permutations (such as RSA) and random oracles. Later, Gagné, Lafourcade, Lakhnech and Safavi-Naini [65, 64] adapt these methods to symmetric encryption modes and message authentication codes.

Malozemoff, Katz and Green [98] and Hoang, Katz and Malozemoff [87] pursue an alternative approach for proving security of modes of operations and authenticated encryption schemes. Their approach relies on a simple but effective type system that tracks whether values are uniform and fresh, or adversarially controlled. By harnessing their type system into a synthesis framework, they are able to generate thousands of constructions with their security proofs, including constructions whose efficiency compete with state-of-the-art algorithms that were discovered using conventional methods. Using SMT-based methods, Tiwari, Gascón and Dutertre [123] introduce an alternative approach to synthesize bitvector programs, padding-based encryption schemes and modes of operation.

Our work is most closely related to CIL [16], ZooCrypt [15] and AutoG&P [22]. Computational Indistinguishability Logic (CIL) [16] is a formal logic for reasoning about security experiments with oracle and adversary calls. CIL is general, in that it does not prescribe a syntax for games, and side-conditions are mathematical statements. CIL does not make any provision for mechaniza-

tion, although, as any mathematical development, CIL can be formalized in a proof assistant, see [50]. ZooCrypt [15] is a platform for synthesizing padding-based encryption schemes; it has been used successfully to analyze more than a million schemes, leading to the discovery of new and interesting schemes. ZooCrypt harnesses two specialized computational logics for proving chosen-plaintext and chosen-ciphertext security, and effective procedures for finding attacks. The computational logics use deducibility to trigger proof steps that apply reduction to one-wayness assumptions, and to compute the probability of bad events using a notion of symbolic entropy. However, ZooCrypt is highly specialized.

AutoG&P [22] introduce a computational logic and provide an implementation of their logic, called AutoG&P, for proving security of pairing-based cryptographic constructions. Their logic uses deducibility for ensuring that proof rules are correctly enforced. Their implementation achieves a high level of automation, thanks to a heuristics for checking deducibility, and a proof search procedure, which decides which proof rule to apply and automatically selects applications of computational assumptions. We build heavily on this work; in particular, AutoLWE is implemented as an independent branch of AutoG&P. The main differences are:

- AutoLWE supports oracle-relative assumptions and general forms of the Left-over Hash Lemma, and (semi-)decision procedures for deducibility problems, for the theories of Diffie-Hellman exponentiation, fields, non-commutative rings and matrices. In contrast, AutoG&P only support more limited assumptions and implements heuristics for the theory of Diffie-Hellman exponentiation;

- AutoG&P supports automated generation of EasyCrypt proofs, which is not supported by AutoLWE. Rather than supporting generation of proofs a posteriori, a more flexible alternative would be to integrate the features of AutoG&P and AutoLWE in EasyCrypt.

Theodorakis and Mitchell [122] develop a category-theoretical framework for game-based security proofs, and leverage their framework for transferring such proofs from the group-based or pairing-based to the lattice-based setting. Their results give an elegant proof-theoretical perspective on the relationship between cryptographic proofs. However, they are not supported by an implementation. In contrast, we implement our computational logic. Furthermore, proofs in AutoLWE have a first-class status, in the form of proof scripts. An interesting direction for future work is to implement automated compilers that transform proofs from the group- and pairing-based settings to the lattice-based settings. Such proof compilers would offer a practical realization of [122] and could also implement patches when they fail on a specific step.

CHAPTER 2

PRELIMINARIES

Notations. 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 $\boldsymbol{v}$ (row vector). We use $[n]$ to denote the set $\{1, ..., n\}$. 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. Let X and Y be two random variables taking values in Ω. Define the statistical distance, denoted as $\Delta(X, Y)$ as

$$\Delta(X, Y) := \frac{1}{2} \sum_{s \in \Omega} |\mathsf{Pr}[X = s] - \mathsf{Pr}[Y = s]|$$

Let $X(\lambda)$ and $Y(\lambda)$ be distributions of random variables. We say that X and Y are statistically close, denoted as $X \overset{s}{\approx} Y$, if $d(\lambda) := \Delta(X(\lambda), Y(\lambda))$ is a negligible function of λ. We say two distributions $X(\lambda)$ and $Y(\lambda)$ are computationally indistinguishable, denoted as $X \overset{c}{\approx} Y$ if for any ppt distinguisher D, it holds that $|\mathsf{Pr}[D(X(\lambda)) = 1] - \mathsf{Pr}[D(Y(\lambda)) = 1]| = \mathsf{negl}(\lambda)$.

2.1 Random Access Machines

We recall the definition of RAM program in [74]. A RAM computation consists of a RAM program P and a database D. The representation size of P is independent of the length of the database D. P has random access to the database D and we represent this as P^D. On input x, $P^D(x)$ outputs the answer y. In more detail, the computation proceeds as follows.

The RAM program P is represented as a step-circuit C. It takes as input internal state from the previous step, location to be read, value at that location

and it outputs the new state, location to be written into, value to be written and the next location to be read. More formally, for every $\tau \in T$, where T is the upper running time bound

$$(\mathsf{st}^\tau, \mathsf{rd}^\tau, \mathsf{wt}^\tau, \mathsf{wb}^\tau) \leftarrow C(\mathsf{st}^{\tau-1}, \mathsf{rd}^{\tau-1}, b^\tau)$$

where we have the following:

- $\mathsf{st}^{\tau-1}$ denotes the state from the $(\tau-1)$-th step and st^τ denotes the state in the τ-th step.
- $\mathsf{rd}^{\tau-1}$ denotes the location to be read from, as output by the $(\tau-1)$-th step.
- b^τ denotes the bit at the location $\mathsf{rd}^{\tau-1}$.
- rd^τ denotes the location to be read from, in the next step.
- wt^τ denotes the location to be written into.
- wb^τ denotes the value to be written at τ-th step at the location wt^τ.

At the end of the computation, denote the final state to be $\mathsf{st}_{\mathsf{end}}$. If the computation has been performed correctly, $\mathsf{st}^{\mathsf{end}} = y$. In this work, we consider a simpler case, where the RAM program P does not take additional input x and the output of P^D is in space $\{0,1\}$.

2.2 Branching Program

We recall the definition of branching program in [39, 83]. A width-w branching program BP of length L with input space $\{0,1\}^\ell$ and s states (represented by $[s]$) is a sequence of L tuples of form $(\mathsf{var}(t), \sigma_{t,0}, \sigma_{t,1})$, where

- $\sigma_{t,0}$ and $\sigma_{t,1}$ are injective functions from $[s]$ to itself.
- $\mathsf{var} : [L] \to [\ell]$ is a function that associates the t-th tuple $\sigma_{t,0}, \sigma_{t,1}$ with the input bit $x_{\mathsf{var}(t)}$.

The branching program BP on input $\boldsymbol{x} = (x_1, ..., x_\ell)$ computes its outputs as follows. At step t, we denote the state of the computation by $\eta_t \in [s]$. The initial state is set to be $\eta_0 = 1$. In general, η_t can be computed recursively as

$$\eta_t = \sigma_{t,x_{\mathsf{var}(t)}}(\eta_{t-1})$$

Finally, after L steps, the output of the computation $\mathsf{BP}(\boldsymbol{x}) = 1$ if $\eta_L = 1$ and 0 otherwise. As mentioned in [39], we represent states with bits rather than numbers to bound the noise growth. In particular, we represent the state $\eta_t \in [s]$ by a unit vector $\boldsymbol{v}_t \in \{0,1\}^s$. The idea is that $\boldsymbol{v}_t = 1$ if and only if $\sigma_{t,x_{\mathsf{var}(t)}}(\eta_{t-1}) = i$. Note that we can also write the above expression as $\boldsymbol{v}_t[i] = 1$ if and only if either:

- $\boldsymbol{v}_{t-1}[\sigma_{t,0}^{-1}(i)] = 1$ and $x_{\mathsf{var}(t)} = 0$.
- $\boldsymbol{v}_{t-1}[\sigma_{t,1}^{-1}(i)] = 1$ and $x_{\mathsf{var}(t)} = 1$.

This later form will be useful since we can rewrite the above conditions in the following formula. For $t \in [L]$ and $i \in [s]$,

$$\begin{aligned}\boldsymbol{v}_t[i] :&= \boldsymbol{v}[\sigma_{t,0}^{-1}(i)](1 - x_{\mathsf{var}(t)}) + \boldsymbol{v}_{t-1}[\sigma_{t,1}^{-1}(i)] \cdot x_{\mathsf{var}(t)} \\ &= \boldsymbol{v}_{t-1}[\gamma_{t,i,0}](1 - x_{\mathsf{var}(t)}) + \boldsymbol{v}_{t-1}[\gamma_{t,i,1}] \cdot x_{\mathsf{var}(t)}\end{aligned}$$

where we set $\gamma_{t,i,0} = \sigma_{t,i}^{-1}(i)$ and $\gamma_{t,i,1} = \sigma_{t,i}^{-1}(i)$, and $\gamma_{t,i,0}, \gamma_{t,i,1}$ can be publicly computed from the description of the branching program. Hence, $\{\mathsf{var}(t), \{\gamma_{t,i,0}, \gamma_{t,i,1}\}_{i\in[s]}\}$ is also a valid representation of a branching program BP.

For clarity of representation, we will deal with width-5 permutation branching program, which is shown to be equivalent to the circuit class $\mathcal{NC}^1$ [13]. Hence, we have $s = w = 5$, and the functions σ_0, σ_1 are permutations on [5].

2.3 Lattice Background

Learning With Errors The LWE problem was introduced by Regev [112], who showed that solving it *on average* is as hard as (quantumly) solving several standard lattice problems *in the worst case*.

Definition 2.3.1 (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 $\mathsf{LWE}_{n,q,\chi}$ *is to distinguish between the following pairs of distributions (e.g., as given by a sampling oracle $O \in \{O_s, O_\$\}$):*

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

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 $\boldsymbol{x} \leftarrow \chi^m$.

In this work we only consider the case where the modulus $q \leq 2^n$. Recall that GapSVP_γ is the (promise) problem of distinguishing, given a basis for a lattice and a parameter d, between the case where the lattice has a vector shorter than d, and the case where the lattice does not have any vector shorter than $\gamma \cdot d$.

There are known reductions between $\mathsf{LWE}_{n,q,\chi}$ and those problems, which allows us to appropriately choose the LWE parameters for our scheme. We summarize in the following corollary (which addresses the regime of sub-exponential modulus-to-noise ratio).

Theorem 2.3.2 ([112, 109, 99, 100, 36])**.** *For any function $B = B(n) \geq \tilde{O}(\sqrt{n})$ there exists a B-bounded distribution ensemble $\chi = \chi(n)$ over the integers s.t. for all $q = q(n)$,*

letting $\gamma = \tilde{O}(\sqrt{b}q/B)$, it holds that $\mathsf{LWE}_{n,q,\chi}$ is at least as hard as the quantum hardness of GapSVP$_\gamma$. Classical hardness GapSVP$_\gamma$ follows if $q(n) \geq 2n/2$ or for other values of q for $\tilde{\Omega}(\sqrt{n})$ dimensional lattices and approximation factor $q/B \cdot \mathsf{poly}(n\lceil \log q \rceil)$.

Trapdoors and Discrete Gaussians Let $n, q \in \mathbb{Z}$, and $m = n\lceil \log q \rceil$ and $\boldsymbol{g} = (1, 2, 4, \ldots, 2^{\lceil \log q \rceil - 1})$. The *gadget matrix* [100] $\mathbf{G}$ is defined as the diagonal concatenation of vector $\boldsymbol{g}$ n times. Formally, $\mathbf{G} = \boldsymbol{g} \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n \times m}$. For any $t \in \mathbb{Z}$, the function $\mathbf{G}^{-1} : \mathbb{Z}_q^{n \times t} \to \{0,1\}^{m \times t}$ expands each entry $a \in \mathbb{Z}_q$ of the input matrix into a column of size $\lceil \log q \rceil$ consisting of the bit-representation of a. For any matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times t}$, it holds that $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A} \bmod q$.

The (centered) discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter τ, denoted $\mathcal{D}_{\mathbb{Z}^m,\tau}$, is the distribution over $\mathbb{Z}^m$ where for all $\boldsymbol{x}$, $\mathsf{Pr}[\boldsymbol{x}] \propto e^{-\pi \|\boldsymbol{x}\|^2/\tau^2}$.

More frequently, we will use the generalized multi-dimensional (or m-variate) Discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}_1^m,\mathbf{Q}}$, which denotes sampling a $\mathbb{Z}_1$-valued m-vector with *covariance matrix* $\mathbf{Q} \in \mathbb{Z}_1^{m \times m}$. In order to sample from the distribution $\mathcal{D}_{\mathbb{Z}_1^m,\mathbf{Q}}$, proceed as follows:

- Sample $\boldsymbol{t}' = (t'_1, ..., t'_m) \in \mathbb{R}^m$ independently as $t'_i \leftarrow \mathcal{D}_1$ for $i \in [m]$.
- Find the Cholesky decomposition $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$.
- Output the vector $\boldsymbol{t} := \mathbf{L}\boldsymbol{t}'$ as the sample $\boldsymbol{t} \leftarrow \mathcal{D}_{\mathbb{Z}_1^m,\mathbf{Q}}$.

Recall that the Cholesky decomposition takes as input any positive-definite matrix $\mathbf{Q} \in \mathbb{R}^{m \times m}$ and outputs a lower triangular matrix $\mathbf{L}$ so that $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$. Further, we recall the fact that the sum of two m-variate Gaussians with means μ_1, μ_2 and variances $\mathbf{Q}_1, \mathbf{Q}_2$ is an m-variate Gaussian with mean $\mu_1 + \mu_2$ and variance $\mathbf{Q}_1 + \mathbf{Q}_2$.

Next we show a useful lemma that we need for our construction.

Lemma 2.3.3. *Let $\mathbf{I}_{m\times m}$ be the m-by-m identity matrix, $\mathbf{R} \in \mathbb{R}^{m\times m}$, and $\mathbf{Q} \stackrel{\text{def}}{=} a^2\mathbf{I}_{m\times m} - b^2\mathbf{R}^T\mathbf{R}$ for positive numbers a, b such that $a > b\|\mathbf{R}^T\|$. Then $\mathbf{Q}$ is positive definite.*

Proof. To show that $\mathbf{Q}$ is positive definite, we need to show that for any column vector $\boldsymbol{x}$ of dimension m, we have $\boldsymbol{x}^T \cdot \mathbf{Q} \cdot \boldsymbol{x} > 0$. We prove this by unfolding the matrix $\mathbf{Q}$:

$$\begin{aligned}
\boldsymbol{x}^T \cdot \mathbf{Q} \cdot \boldsymbol{x} &= \boldsymbol{x}^T \cdot (a^2\mathbf{I}_{m\times m} - b^2\mathbf{R}^T\mathbf{R}) \cdot \boldsymbol{x} \\
&= a^2\boldsymbol{x}^T\mathbf{I}_{m\times m}\boldsymbol{x} - b^2\boldsymbol{x}^T\mathbf{R}^T\mathbf{R}\boldsymbol{x} \\
&= a^2\|\boldsymbol{x}^T\|^2 - b^2\|\boldsymbol{x}^T\mathbf{R}^T\|^2 \\
&> b^2\|\boldsymbol{x}^T\|^2 \cdot \|\mathbf{R}^T\|^2 - b^2\|\boldsymbol{x}^T\mathbf{R}^T\|^2.
\end{aligned}$$

Since $\|\boldsymbol{x}^T\| \cdot \|\mathbf{R}^T\| \geq \|\boldsymbol{x}^T\mathbf{R}^T\|$, we can conclude $\boldsymbol{x}^T \cdot \mathbf{Q} \cdot \boldsymbol{x} > 0$. □

Definition 2.3.4 (Matrix norm)**.** *Let S^m denote the set of vectors in $\mathbb{R}^{m+1}$ whose length is 1. Then the norm of a matrix $\mathbf{R} \in \mathbb{R}^{m\times m}$ is defined to be $\sup_{\boldsymbol{x}\in S^m}\|\boldsymbol{x}^T\mathbf{R}\|$.*

We note that throughout the paper, for vector-matrix multiplication, we always multiply the vector on the left-hand side of the matrix. Then we have the following lemma, which bounds the norm for some specified distributions.

Lemma 2.3.5 ([2])**.** *Regarding the norm defined above, we have the following bounds:*

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

The following lemmas have been established in a sequence of works.

Lemma 2.3.6 (Trapdoor Generation [75, 100])**.** *Let q, n, m be positive integers with $q \geq 2$ and sufficiently large $m = \Omega(n \log q)$. There exists a* ppt *algorithm* $\mathsf{TrapGen}(1^n, q, m)$ *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 the distribution of* $\mathbf{A}$ *is statistically close to uniform distribution over* $\mathbb{Z}_q^{n \times m}$ *and* $\|\mathbf{T_A}\| \leq O(\sqrt{n \log q})$.

Lemma 2.3.7 ([75, 47, 2])**.** *Given integers $n \geq 1, q \geq 2$ there exists some $m = m(n, q) = O(n \log q)$ There are sampling algorithms as follows:*

- *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 $\mathbf{F} := [\mathbf{A} \| \mathbf{AR} + \mathbf{B}]$.*

Based on the above sampling algorithms, we have the following lemma:

Lemma 2.3.8 ([82])**.** *Given integers $n \geq 1, q \geq 2$ there exists some $m = m(n, q) = O(n \log q)$, $\beta = \beta(n, q) = O(n\sqrt{\log q})$ and $s > \|\widetilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log(m)})$ such that for all $m \geq m^*$ and all k, we have the following two distributions are statistically close*

$$(\mathbf{A}, \mathbf{T_A}, \mathbf{B}, \mathbf{U}, \mathbf{V}) \approx (\mathbf{A}, \mathbf{T_A}, \mathbf{B}, \mathbf{U}', \mathbf{V}')$$

where $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(q, n, m)$, $(\mathbf{A}', \mathbf{B}) \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ *and* $\mathbf{U} \leftarrow \mathcal{D}_{\mathbb{Z}^{2m \times k}}$, $\mathbf{V} = \mathbf{A} \cdot \mathbf{U}$, $\mathbf{V}' \xleftarrow{\$} \mathbb{Z}_q^{n \times k}$ *and* $\mathbf{U}' \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \mathbf{V}', s)$.

Lemma 2.3.9 (Noise Rerandomization [90])**.** *Let q, ℓ, m be positive integers and r a positive real satisfying $r > max\{\eta_\epsilon(\mathbb{Z}^m), \eta_\epsilon(\mathbb{Z}^\ell)\}$. Let $\boldsymbol{b} \in \mathbb{Z}_q^m$ be arbitrary and $\boldsymbol{x}$ chosen from $D_{\mathbb{Z}^m,r}$. Then for any $\mathbf{V} \in \mathbb{Z}^{m\times\ell}$ and positive real $\sigma > \mathsf{s}_1(\mathbf{V})$, there exists a PPT algorithm $\mathsf{ReRand}(\mathbf{V}, \boldsymbol{b} + \boldsymbol{x}, r, \sigma)$ that outputs $\boldsymbol{b}' = \boldsymbol{b}\mathbf{V} + \boldsymbol{x}' \in \mathbb{Z}_q^\ell$ where the statistical distance of the discrete Gaussian $D_{\mathbb{Z}^\ell,2r\sigma}$ and the distribution of $\boldsymbol{x}'$ is within 8ϵ.*

We conclude with a variant of Leftover Hash Lemma [2, 31]:

Lemma 2.3.10. *Suppose that $m > (n+1)\log q + \omega(\log n)$ and that $q > 2$ is prime. Let $\mathbf{S}$ be an $m \times k$ matrix chosen uniformly in $\{0,1\}^{m\times k}$ where $k = k(n)$ is polynomial in n. Let $\mathbf{A}$ and $\mathbf{B}$ be matrices chosen uniformly in $\mathbb{Z}_q^{n\times m}$ and $\mathbb{Z}_q^{n\times k}$ respectively. Then, for all vectors $\boldsymbol{e}$ in $\mathbb{Z}_q^m$, the distribution $(\mathbf{A}, \mathbf{AS}, \boldsymbol{e}\mathbf{S})$ is statistically close to the distribution $(\mathbf{A}, \mathbf{B}, \boldsymbol{e}\mathbf{S})$.*

Lattice Evolution The following is an abstraction of the evaluation procedure in recent LWE based FHE and ABE schemes that developed in a long sequence of works [2, 100, 76, 6, 31, 82]. We use a similar formalism as in [40, 35, 38].

Theorem 2.3.11. *There exist efficient deterministic algorithms* PubEval *and* CtEval *such that for all $n, q, \ell \in \mathbb{N}$, and for any sequence of matrices $(\mathbf{D}_1, \ldots, \mathbf{D}_\ell) \in (\mathbb{Z}_q^{n\times n\lceil\log q\rceil})^\ell$, for any depth-$d$ Boolean circuit $f : \{0,1\}^\ell \to \{0,1\}$ and for every $\boldsymbol{x} = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$, the following properties hold:*

- $\mathsf{PubEval}(f, \{\mathbf{D}_i \in \mathbb{Z}_q^{n\times n\lceil\log q\rceil}\}_{i\in[\ell]})$*: On input matrices $\{\mathbf{D}_i\}_{i\in[d]}$ and a function $f \in \mathcal{F}$, the public evaluation algorithm outputs $\mathbf{D}_f \in \mathbb{Z}_q^{n\times n\lceil\log q\rceil}$ as the result.*
- $\mathsf{TrapEval}(f, \boldsymbol{x}, \mathbf{A} \in \mathbb{Z}_q^{n\times\lceil\log q\rceil}, \{\mathbf{R}_i\}_{i\in[\ell]})$*: the trapdoor evaluation algorithm outputs $\mathbf{R}_f$, such that*

$$\mathsf{PubEval}(f, \{\mathbf{AR}_i + x_i\mathbf{G}\}_{i\in[\ell]}) = \mathbf{AR}_f + f(\boldsymbol{x})\mathbf{G}$$

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

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

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

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

CHAPTER 3

DENIABLE ATTRIBUTE BASED ENCRYPTION FOR BRANCHING PROGRAMS

3.1 New Definitions and Tools

In this section, we first describe our new notion of flexibly bi-deniable ABE, which is a natural generalization of the flexibly bi-deniable PKE of [105]. Then we define the notion of a flexibly attribute-based bi-translucent set (AB-BTS), which generalizes the idea of bi-translucent set (BTS) in the work [105]. Using a similar argument as in the work [105], we can show that an AB-BTS suffices to construct bi-deniable ABE. In the last part of this section, we define a new assumption called Extended LWE Plus, and show its hardness by giving a reduction from the standard LWE problem.

3.1.1 Flexibly Bi-Deniable ABE: Syntax and Deniability Definition

A flexibly bi-deniable key-policy attribute based encryption for a class of Boolean circuits $\mathcal{C} : \{0,1\}^{\ell} \rightarrow \{0,1\}$ consists a tuple of ppt algorithms $\Pi = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{DenSetup}, \mathsf{DenEnc}, \mathsf{SendFake}, \mathsf{RecFake})$. We describe them in detail as follows:

$\mathsf{Setup}(1^{\lambda})$: On input the security parameter λ, the setup algorithm outputs public parameters PP and master secret key MSK.

$\mathsf{Keygen}(\mathsf{MSK}, f)$: On input the master secret key MSK and a function $f \in C$, it outputs a secret key SK_f.

$\mathsf{Enc}(\mathsf{PP}, \boldsymbol{x}, \mu; r_S)$: On input the public parameter PP, an attribute/message pair $(\boldsymbol{x}, \mu)$ and randomness r_S, it outputs a ciphertext $c_{\boldsymbol{x}}$.

$\mathsf{Dec}(\mathsf{SK}_f, c_{\boldsymbol{x}})$: On input the secret key SK_f and a ciphertext $c_{\boldsymbol{x}}$, it outputs the corresponding plaintext μ if $f(\boldsymbol{x}) = 0$; otherwise, it outputs $\perp$.

$\mathsf{DenSetup}(1^\lambda)$: On input the security parameter λ, the deniable setup algorithm outputs pubic parameters PP, master secret key MSK and faking key fk.

$\mathsf{DenEnc}(\mathsf{PP}, \boldsymbol{x}, \mu; r_S)$: On input the public parameter PP, an attribute/message pair $(\boldsymbol{x}, \mu)$ and randomness r_S, it outputs a ciphertext $c_{\boldsymbol{x}}$.

$\mathsf{SendFake}(\mathsf{PP}, r_S, \mu, \mu')$: On input public parameters PP, original random coins r_S, message μ of DenEnc and desired message μ', it outputs a faked random coin r'_S.

$\mathsf{RecFake}(\mathsf{PP}, \mathsf{fk}, c_{\boldsymbol{x}}, f, \mu')$: On input public parameters PP, faking key fk, a ciphertext $c_{\boldsymbol{x}}$, a function $f \in C$, and desired message μ', the receiver faking algorithm outputs a faked secret key SK'_f.

Correctness. We say the flexibly bi-deniable ABE scheme described above is correct, if for any $(\mathsf{MSK}, \mathsf{PP}) \leftarrow \mathsf{S}(1^\lambda)$, where $\mathsf{S} \in \{\mathsf{Setup}, \mathsf{DenSetup}\}$, any message μ, function $f \in C$, and any attribute vector $\boldsymbol{x}$ where $f(\boldsymbol{x}) = 0$, we have $\mathsf{Dec}(\mathsf{SK}_f, c_{\boldsymbol{x}}) = \mu$, where $\mathsf{SK}_f \leftarrow \mathsf{Keygen}(\mathsf{MSK}, f)$ and $c_{\boldsymbol{x}} \leftarrow \mathsf{E}(\mathsf{PP}, \boldsymbol{x}, \mu; r_S)$ where $\mathsf{E} \in (\mathsf{Enc}, \mathsf{DenEnc})$.

Bi-deniability Definition. Let μ, μ' be two arbitrary messages, not necessarily different. We propose the bi-deniability definition by describing real ex-

periment $\mathsf{Expt}^{\mathsf{Real}}_{\mathcal{A},\mu,\mu'}(1^\lambda)$ and faking experiment $\mathsf{Expt}^{\mathsf{Fake}}_{\mathcal{A},\mu,\mu'}(1^\lambda)$ regarding adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ below:

1. $(\boldsymbol{x}^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\lambda)$
2. $(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$
3. $c'_{\boldsymbol{x}^*} \leftarrow \mathsf{Enc}(\mathsf{PP}, \boldsymbol{x}^*, \mu; r_S)$
4. $(f^*, \mathsf{st}_2) \leftarrow \mathcal{A}_2^{\mathsf{KG}(\mathsf{MSK}, \boldsymbol{x}^*, \cdot)}(\mathsf{PP}, \mathsf{st}_1, c_{\boldsymbol{x}^*})$
5. $\mathsf{SK}_{f^*} \leftarrow \mathsf{Keygen}(\mathsf{MSK}, f^*)$
6. $b \leftarrow \mathcal{A}_3^{\mathsf{KG}(\mathsf{MSK}, \boldsymbol{x}^*, \cdot)}(\mathsf{SK}_{f^*}, c, \mathsf{st}_2, r_S)$
7. Output $b \in \{0, 1\}$

(a) $\mathsf{Expt}^{\mathsf{Real}}_{\mathcal{A}}(1^\lambda)$

1. $(\boldsymbol{x}^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\lambda)$
2. $(\mathsf{PP}, \mathsf{MSK}, \mathsf{fk}) \leftarrow \mathsf{DenSetup}(1^\lambda)$
3. $c'_{\boldsymbol{x}^*} \leftarrow \mathsf{DenEnc}(\mathsf{PP}, \boldsymbol{x}^*, \mu'; r_S)$
4. $(f^*, \mathsf{st}_2) \leftarrow \mathcal{A}_2^{\mathsf{KG}(\mathsf{MSK}, \boldsymbol{x}^*, \cdot)}(\mathsf{PP}, \mathsf{st}_1, c'_{\boldsymbol{x}^*})$
5. $r'_S \leftarrow \mathsf{SendFake}(\mathsf{PP}, \mu, \mu', r_S)$
6. $\mathsf{SK}_{f^*} \leftarrow \mathsf{RecFake}(\mathsf{PP}, \mathsf{fk}, c'_{\boldsymbol{x}}, \boldsymbol{v}^*, \mu')$
7. $b \leftarrow \mathcal{A}_3^{\mathsf{KG}(\mathsf{MSK}, \boldsymbol{x}^*, \cdot)}(\mathsf{SK}_{f^*}, c, \mathsf{st}_2, r'_S)$
8. Output $b \in \{0, 1\}$

(b) $\mathsf{Expt}^{\mathsf{Fake}}_{\mathcal{A}}(1^\lambda)$

Figure 3.1: Security experiments for bi-deniable ABE

where $\mathsf{KG}(\mathsf{MSK}, \boldsymbol{w}^*, \cdot)$ returns a secret key $\mathsf{SK}_{\boldsymbol{v}} \leftarrow \mathsf{Keygen}(\mathsf{MSK}, \boldsymbol{v})$ if $\langle \boldsymbol{v}, \boldsymbol{w}^* \rangle \neq 0$ and $\perp$ otherwise.

Definition 3.1.1 (Flexibly Bi-Deniable ABE)**.** *An ABE scheme* Π *is bi-deniable if for any two messages* μ, μ'*, any probabilistic polynomial-time adversaries* $\mathcal{A}$ *where* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$*, there is a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\mathsf{Adv}^{\Pi}_{\mathcal{A},\mu,\mu'}(1^\lambda) = |\Pr[\mathsf{Expt}^{\mathsf{Real}}_{\mathcal{A},\mu,\mu'}(1^\lambda) = 1] - \Pr[\mathsf{Expt}^{\mathsf{Fake}}_{\mathcal{A},\mu,\mu'}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

3.1.2 Attribute Based Bitranslucent Set Scheme

In this section, we define the notion of a *Attribute Based Bitranslucent Set* (AB-BTS), which is an extension of bitranslucent sets (BTS) as defined by O'Neill et al. in [105]. Our new notion permits a more fine-grained degree of access control, where pseudorandom samples and secret keys are associated with attributes $\boldsymbol{x}$, and the testing algorithm can successfully distinguish a pseudorandom sample from a truly random one if and only if the attribute of the sample is

accepted under a given secret key's policy f – i.e., when $f(\boldsymbol{x}) = \mathbf{0}$. This concept is reminiscent of *attribute-based encryption* (ABE), and in fact, we will show in the sequel how to construct a flexibly bi-deniable ABE from an AB-BTS. This is analogous to the construction of a flexibly bi-deniable PKE from O'Neill et al.'s BTS. We present the formal definition below.

Let $\mathbb{F}$ be some family of functions. An attribute based bitranslucent set (AB-BTS) scheme for $\mathbb{F}$ consists of the following algorithms:

$\mathsf{Setup}(1^\lambda)$: On input the security parameter, the normal setup algorithm outputs a public parameter PP and master secret key MSK.

$\mathsf{DenSetup}(1^\lambda)$: On input the security parameter, the deniable setup algorithm outputs a public parameter PP, master secret key MSK and faking key fk.

$\mathsf{Keygen}(\mathsf{MSK}, f)$: On input the master secret key MSK and a function $f \in \mathbb{F}$, the key generation algorithm outputs a secret key SK_f.

P- and U-samplers $\mathsf{SampleP}(\mathsf{PP}, \boldsymbol{x}; r_S)$ and $\mathsf{SampleU}(\mathsf{PP}, \boldsymbol{x}; r_S)$ output some $\boldsymbol{c}$.

$\mathsf{TestP}(\mathsf{SK}_f, \boldsymbol{c}_{\boldsymbol{x}})$: On input a secret key SK_f and a ciphertext $\boldsymbol{c}_{\boldsymbol{x}}$, the P-tester algorithm outputs 1 (accepts) or 0 (rejects).

$\mathsf{FakeSCoins}(\mathsf{PP}, r_S)$: On input a public parameters PP and randomness r_S, the sender-faker algorithm outputs randomness r_S^*.

$\mathsf{FakeRCoins}(\mathsf{PP}, \mathsf{fk}, \boldsymbol{c}_{\boldsymbol{x}}, f)$: On input a public parameters PP, the faking key fk, a ciphertext $\boldsymbol{c}_{\boldsymbol{x}}$ and a function $f \in \mathbb{F}$, the receiver-faker algorithm outputs a faked secret key SK'_f.

Definition 3.1.2 (AB-BTS)**.** *We say a scheme* $\Pi = (\mathsf{Setup}, \mathsf{DenSetup}, \mathsf{Keygen}, \mathsf{SampleP}, \mathsf{SampleU}, \mathsf{TestP}, \mathsf{FakeSCoins}, \mathsf{FakeRCoins})$ *is an AB-BTS scheme for a function family* $\mathbb{F}$ *if it satisfies:*

1. *(Correctness.) The following experiments accept or respectively reject with overwhelming probability over the randomness.*

   - *Let* $(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$, $f \in \mathbb{F}$, $\mathsf{SK}_f \leftarrow \mathsf{Keygen}(\mathsf{MSK}, f)$. *If* $f(\boldsymbol{x}) = 0$ *and* $\boldsymbol{c_x} \leftarrow \mathsf{SampleP}(\mathsf{PP}, \boldsymbol{x}; r_S)$, *then* $\mathsf{TestP}(\mathsf{SK}_f, \boldsymbol{c_x}) = 1$; *otherwise,* $\mathsf{TestP}(\mathsf{SK}_f, \boldsymbol{c_x}) = 0$.
   - *Let* $(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$, $f \in \mathbb{F}$, $\mathsf{SK}_f \leftarrow \mathsf{Keygen}(\mathsf{MSK}, f)$, $\boldsymbol{c} \leftarrow \mathsf{SampleU}(\mathsf{PP}; r_S)$. *Then* $\mathsf{TestP}(\mathsf{SK}_f, \boldsymbol{c}) = 0$.

2. *(Indistinguishable public parameters.) The public parameters* PP *generated by the two setup algorithms* $(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $(\mathsf{PP}, \mathsf{MSK}, \mathsf{fk}) \leftarrow \mathsf{DenSetup}(1^\lambda)$ *should be indistinguishable.*

3. *(Selective bi-deniability.) Let* $\mathbb{F}$ *be a family of functions. We define the following two experiments: the real experiment* $\mathsf{Expt}^{\mathsf{Real}}_{\mathcal{A},\mathbb{F}}(1^\lambda)$ *and the faking experiment* $\mathsf{Expt}^{\mathsf{Fake}}_{\mathcal{A},\mathbb{F}}(1^\lambda)$ *regarding an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *below:*

   where $\mathsf{KG}(\mathsf{MSK}, \boldsymbol{x}^*, \cdot)$ *returns a secret key* $\mathsf{SK}_f \leftarrow \mathsf{Keygen}(\mathsf{MSK}, f)$ *if* $f \in \mathbb{F}$ *and*

(a) $\mathsf{Expt}^{\mathsf{Real}}_{\mathcal{A}}(1^\lambda)$:
- $(f^*, \boldsymbol{x}^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\lambda)$
- $(\mathsf{PP}, \mathsf{MSK}, \mathsf{fk}) \leftarrow \mathsf{DenSetup}(1^\lambda)$
- $\boldsymbol{c} \leftarrow \mathsf{SampleU}(\mathsf{PP}; r_S)$
- $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{KG}(\mathsf{MSK}, \boldsymbol{x}^*, \cdot)}(\mathsf{PP}, \mathsf{st}_1, \boldsymbol{c})$
- $\mathsf{SK}_{f^*} \leftarrow \mathsf{Keygen}(\mathsf{MSK}, f^*)$
- $b \leftarrow \mathcal{A}_3^{\mathsf{KG}(\mathsf{MSK}, \boldsymbol{x}^*, \cdot)}(\mathsf{SK}_{f^*}, c, \mathsf{st}_2, r_S)$
- Output $b \in \{0, 1\}$

(b) $\mathsf{Expt}^{\mathsf{Fake}}_{\mathcal{A}}(1^\lambda)$:
- $(f^*, \boldsymbol{x}^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\lambda)$
- $(\mathsf{PP}, \mathsf{MSK}, \mathsf{fk}) \leftarrow \mathsf{DenSetup}(1^\lambda)$
- $\boldsymbol{c} \leftarrow \mathsf{SampleP}(\mathsf{PP}, \boldsymbol{x}^*; r_S)$
- $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{KG}(\mathsf{MSK}, \boldsymbol{x}^*, \cdot)}(\mathsf{PP}, \mathsf{st}_1, \boldsymbol{c})$
- $r'_S \leftarrow \mathsf{FakeSCoins}(\mathsf{PP}, r_S)$
- $\mathsf{SK}_{f^*} \leftarrow \mathsf{FakeRCoins}(\mathsf{PP}, \mathsf{fk}, \boldsymbol{c}, f^*)$
- $b \leftarrow \mathcal{A}_3^{\mathsf{KG}(\mathsf{MSK}, \boldsymbol{x}^*, \cdot)}(\mathsf{SK}_{f^*}, c, \mathsf{st}_2, r'_S)$
- Output $b \in \{0, 1\}$

Figure 3.2: Security experiments for AB-BTS

   $f(\boldsymbol{x}^*) \neq 0$*; it returns* $\perp$ *otherwise. We also require that* $f^* \in \mathbb{F}$.

   We say the scheme is selectively bi-deniable for $\mathbb{F}$*, if for any probabilistic polynomial-*

time adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, there is a negligible function $\mathsf{negl}(\lambda)$ *such that*

$$\mathsf{Adv}^{\Pi}_{\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Expt}^{\mathsf{Real}}_{\mathcal{A},\mathbb{F}}(1^\lambda) = 1] - \Pr[\mathsf{Expt}^{\mathsf{Fake}}_{\mathcal{A},\mathbb{F}}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

Remark 3.1.3. *Correctness for the faking algorithms is implied by the bi-deniability property. In particular, with overwhelming probability over the overall randomness, the following holds: let* $(\mathsf{PP}, \mathsf{MSK}, \mathsf{fk}) \leftarrow \mathsf{DenSetup}(1^\lambda)$, $f \in \mathbb{F}$, $\mathsf{SK}_f \leftarrow \mathsf{Keygen}(\mathsf{MSK}, f)$, *$x$ be a string and* $\boldsymbol{c_x} \leftarrow \mathsf{SampleP}(\mathsf{PP}, x; r_S)$, *then*

- $\mathsf{SampleU}(\mathsf{PP}; \mathsf{FakeSCoins}(\mathsf{PP}, r_S)) = \boldsymbol{c_x}$,
- $\mathsf{TestP}(\mathsf{FakeRCoins}(\mathsf{PP}, \mathsf{fk}, \boldsymbol{c_x}, f), \boldsymbol{c_x}) = 0$
- *For any other x', let* $\boldsymbol{c}' \leftarrow \mathsf{SampleP}(\mathsf{PP}, x'; r'_S)$, *then (with overwhelming probability) we have*

$$\mathsf{TestP}\left(\mathsf{FakeRCoins}(\mathsf{PP}, \mathsf{fk}, \boldsymbol{c_x}, f), \boldsymbol{c}'\right) = \mathsf{TestP}(\mathsf{SK}_f, c').$$

It is not hard to see that if one of these does not hold, then one can easily distinguish the real experiment from the faking experiment.

Remark 3.1.4. *Canetti et al. [43] gave a simple encoding technique to construct a sender-deniable encryption scheme from a translucent set. O'Neill, Peikert, and Waters [105] used a similar method to construct a flexibly bi-deniable encryption from a bi-translucent set scheme. Here we further observe that the same method as well allows us to construct a flexibly bi-deniable ABE scheme from bi-deniable AB-BTS. We present the construction in Section 3.2.4.*

3.1.3 Extended LWE and Our New Variant

O'Neill et al. [105] introduced the Extended LWE problem, which allows a "hint" on the error vector $\boldsymbol{x}$ to leak in form of a noisy inner product. They ob-

serve a trivial "blurring" argument shows that LWE reduces to eLWE when the hint-noise βq is superpolynomially larger than the magnitude of samples from χ, and also allows for unboundedly many *independent* hint vectors $\langle \boldsymbol{z}, \boldsymbol{x}_i \rangle$ while retaining LWE-hardness.

Definition 3.1.5 (Extended LWE)**.** *For an integer $q = q(n) \geq 2$, and an error distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, the extended learning with errors problem $\mathsf{eLWE}_{n,m,q,\chi,\beta}$ is to distinguish between the following pairs of distributions:*

$$\{\mathbf{A}, \boldsymbol{b} = \mathbf{A}^T \boldsymbol{s} + \boldsymbol{e}, z, \langle z, \boldsymbol{b} - \boldsymbol{e} \rangle + e'\} \text{ and } \{\mathbf{A}, \boldsymbol{u}, z, \langle z, \boldsymbol{u} - \boldsymbol{x} \rangle + e'\}$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^m$, $\boldsymbol{e}, \boldsymbol{z} \xleftarrow{\$} \chi^m$ and $e' \xleftarrow{\$} \mathcal{D}_{\beta q}$.

Further, Alperin-Sheriff and Peikert [5] show that LWE reduces to eLWE with a polynomial modulus and no hint-noise (i.e., $\beta = 0$), even in the case of a bounded number of *independent* hints.

We introduce the following new form of extended-LWE, called eLWE^+, which considers leaking a pair of *correlated hints* on the same noise vector. Our security proof of the AB-BTS construction relies on this new assumption.

Definition 3.1.6 (Extended LWE Plus)**.** *For integer $q = q(n) \geq 2, m = m(n)$, an error distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, and a matrix $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$, the extended learning with errors problem $\mathsf{eLWE}^+_{n,m,q,\chi,\beta,\mathbf{R}}$ is to distinguish between the following pairs of distributions:*

$$\{\mathbf{A}, \boldsymbol{b} = \mathbf{A}^T \boldsymbol{s} + \boldsymbol{e}, z_0, z_1, \langle z_0, \boldsymbol{b} - \boldsymbol{e} \rangle + e, \langle \mathbf{R} z_1, \boldsymbol{b} - \boldsymbol{e} \rangle + e'\} \text{ and}$$

$$\{\mathbf{A}, \boldsymbol{u}, z_0, z_1, \langle z_0, \boldsymbol{u} - \boldsymbol{e} \rangle + e, \langle \mathbf{R} z_1, \boldsymbol{u} - \boldsymbol{e} \rangle + e'\}$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^m$, $\boldsymbol{e}, z_0, z_1 \xleftarrow{\$} \chi^m$ and $e, e' \xleftarrow{\$} \mathcal{D}_{\beta q}$.

Hardness of extended-LWE$^+$. A simple observation, following prior work, is that when χ is $\mathsf{poly}(n)$-bounded and the hint noise βq (and thus, modulus q) is superpolynomial in n, then $\mathsf{LWE}_{n,m,q,\chi}$ trivially reduces to $\mathsf{eLWE}^+_{n,m,q,\chi,\beta,\mathbf{R}}$ for every $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$ so that $\mathbf{R}\boldsymbol{z}_1$ has $\mathsf{poly}(n)$-bounded norm. This is because, for any $r = \omega(\sqrt{\log n}), c \in \mathbb{Z}$, the statistical distance between $\mathcal{D}_{\mathbb{Z},r}$ and $c + \mathcal{D}_{\mathbb{Z},r}$ is at most $O(|c|/r)$.

However, our cryptosystem will require a polynomial-size modulus q. So, we next consider the case of *prime* modulus q of $\mathsf{poly}(n)$ size and no noise on the hints (i.e., $\beta = 0$). Following [5][1], it will be convenient to swap to the "knapsack" form of LWE, which is: given $\mathbf{H} \leftarrow \mathbb{Z}_q^{(m-n)\times m}$ and $\boldsymbol{c} \in \mathbb{Z}_q^{m-n}$, where either $\mathbf{c} = \mathbf{H}\boldsymbol{e}$ for $\boldsymbol{e} \leftarrow \chi^m$ or $\mathbf{c}$ uniformly random and independent of $\mathbf{H}$, determine which is the case (with non-negligible advantage). The "extended-plus" form of the knapsack problem also reveals a pair of hints $(\boldsymbol{z}_0, \boldsymbol{z}_1, \langle \boldsymbol{z}_0, \boldsymbol{e}\rangle, \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}\rangle)$. Note the equivalence between LWE and knapsack-LWE is proven in [99] for $m \geq n + \omega(\log n)$.

Theorem 3.1.7. *For $m \geq n + \omega(\log n)$, for every prime $q = \mathsf{poly}(n)$, for every $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$, and for every $\beta \geq 0$, $\mathbf{Adv}^{\mathsf{LWE}_{n,m,q,\chi}}_{\mathcal{B}^{\mathcal{A}}}(1^\lambda) \geq (1/q^2)\mathbf{Adv}^{\mathsf{eLWE}^+_{n,m,q,\chi,\beta,\mathbf{R}}}_{\mathcal{A}}(1^\lambda)$.*

Proof. We construct an LWE to eLWE^+ reduction $\mathcal{B}$ as follows. $\mathcal{B}$ receives a knapsack-LWE instance $\mathbf{H} \in \mathbb{Z}_q^{(m-n)\times m}, \boldsymbol{c} \in \mathbb{Z}_q^{m-n}$. It samples $\boldsymbol{e}', \boldsymbol{z}_0, \boldsymbol{z}_1 \leftarrow \chi^m$ and uniform $\boldsymbol{v}_0, \boldsymbol{v}_1 \leftarrow \mathbb{Z}_q^{m-n}$. It chooses any $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$, then sets

$$\mathbf{H}' := \mathbf{H} - \boldsymbol{v}_0\boldsymbol{z}_0^T - \boldsymbol{v}_1 (\mathbf{R}\boldsymbol{z}_1)^T \in \mathbb{Z}_q^{(m-n)\times m},$$

$$\boldsymbol{c}' := \boldsymbol{c} - \boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e}'\rangle - \boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}'\rangle \in \mathbb{Z}_q^{m-n}.$$

It sends $(\mathbf{H}', \boldsymbol{c}', \mathbf{z}_0, \mathbf{z}_1, \langle \boldsymbol{z}_0, \boldsymbol{e}'\rangle, \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}'\rangle)$ to the knapsack-eLWE^+ adversary $\mathcal{A}$, and outputs what $\mathcal{A}$ outputs.

[1] We note that a higher quality reduction from LWE to eLWE is given in [36] in the case of binary secret keys. However for our cryptosystem, it will be more convenient to have secret key coordinates in $\mathbb{Z}_q$, so we extend the reduction of [5] to eLWE^+ instead.

Notice that when $\mathbf{H}, \boldsymbol{c}$ are independent and uniform, so are $\mathbf{H}', \mathbf{c}'$, in which case $\mathcal{B}$'s simulation is perfect.

Now, consider the case when $\mathbf{H}, \mathbf{c}$ are drawn from the knapsack-LWE distribution, with $\mathbf{c} = \mathbf{H}\boldsymbol{x}$ for $\boldsymbol{e} \leftarrow \chi^m$. In this case, $\mathbf{H}'$ is uniformly random over the choice of $\mathbf{H}$, and we have

$$\begin{aligned}
\boldsymbol{c}' &= \mathbf{H}\boldsymbol{x} - \boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e}' \rangle - \boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}' \rangle \\
&= \left(\mathbf{H}' + \boldsymbol{v}_0 \boldsymbol{z}_0^T + \boldsymbol{v}_1 \left(\mathbf{R}\boldsymbol{z}_1\right)^T\right)\boldsymbol{e} - \boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e}' \rangle - \boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}' \rangle \\
&= \mathbf{H}'\boldsymbol{e} + \boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e} - \boldsymbol{e}' \rangle + \boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e} - \boldsymbol{e}' \rangle.
\end{aligned}$$

Define the event $E = [E_0 \wedge E_1]$ as

$$\begin{aligned}
E_0 &\stackrel{\text{def}}{=} \left[\langle \boldsymbol{z}_0, \boldsymbol{e} \rangle = \langle \boldsymbol{z}_0, \boldsymbol{e}' \rangle\right], \\
E_1 &\stackrel{\text{def}}{=} \left[\langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e} \rangle = \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}' \rangle\right].
\end{aligned}$$

If event E occurs, then the reduction $\mathcal{B}$ perfectly simulates a pseudorandom instance of knapsack-eLWE^+ to $\mathcal{A}$, as then $\boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e} - \boldsymbol{e}' \rangle + \boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e} - \boldsymbol{e}' \rangle$ vanishes, leaving $\boldsymbol{c}' = \mathbf{H}'\boldsymbol{e}$ for $\mathbf{H}' \leftarrow \mathbb{Z}_q^{(m-n)\times m}$ and $\boldsymbol{e} \leftarrow \chi^m$ as required. Otherwise since q is prime, the reduction $\mathcal{B}$ (incorrectly) simulates an independent and uniform instance of knapsack-eLWE^+ to $\mathcal{A}$, as then either one of $\boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e} - \boldsymbol{e}' \rangle$ or $\boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e} - \boldsymbol{e}' \rangle$ does not vanish, implying that $\boldsymbol{c}'$ is uniform in $\mathbb{Z}_q^{m-n}$ over the choice of $\boldsymbol{v}_0$ (resp. $\boldsymbol{v}_1$) alone, independent of the choices of $\mathbf{H}'$ and $\boldsymbol{x}$.

It remains to analyze the probability that event E occurs. Because $\boldsymbol{e}$ and $\boldsymbol{e}'$ are i.i.d., we may define the random variable $\mathcal{Z}_0$ that takes values $\langle \boldsymbol{z}_0, \boldsymbol{e}^* \rangle \in \mathbb{Z}_q$ and the random variable $\mathcal{Z}_1$ that takes values $\langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}^* \rangle \in \mathbb{Z}_q$ jointly over choice of $\boldsymbol{e}^* \leftarrow \chi^m$, and analyze their collision probabilities independently. Since the collision probability of *any* random variable $\mathcal{Z}$ is at least $1/|\mathsf{Supp}(\mathcal{Z})|$, we have

that $\Pr[E] \geq \min CP[\mathcal{Z}_0] \cdot \min CP[\mathcal{Z}_1] = 1/q^2 = 1/\mathsf{poly}(n)$, and the theorem follows.

□

3.2 Flexibly Bi-Deniable Attribute-Based Encryption (ABE) for Branching Programs

In this section, we present our flexibly bi-deniable ABE for bounded-length Branching Program. We organize our approach into the following three steps: **(1)** first, we recall the encoding scheme proposed in the SIM-secure ABE-BP of [83]; **(2)** Then, we present our flexibly bi-deniable attribute bi-translucent set (AB-BTS) scheme, as was defined in Definition 3.1.2. Our AB-BTS construction uses the ideas of Gorbunov and Vinayagamurthy [83], with essential modifications that allow us to tightly upper and lower bound evaluated noise terms. As discussed in the Introduction, this tighter analysis plays a key role in proving bi-deniability. **(3)** Finally, we show how to obtain the desired bi-deniable ABE scheme from our AB-BTS. As pointed out by Canetti et al. [43] and O'Neill et al. [105], a bitranslucent set scheme implies flexibly bi-deniable PKE. We observe that the same idea generalizes to the case of an AB-BTS scheme and flexibly bi-deniable ABE in a straightforward manner.

3.2.1 Encoding Schemes for Branching Programs

Basic Homomorphic Encoding. Before proceeding to the public key evaluation algorithm, we first described basic homomorphic addition and multiplication over public keys and encoded ciphertexts based on the techniques in

[76, 6, 31].

Definition 3.2.1 (LWE Encoding). *For any matrix* $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$*, we define an LWE encoding of a bit* $a \in \{0,1\}$ *with respect to a public key* $\mathbf{A}$ *and randomness* $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$ *as*

$$\psi_{\mathbf{A},\boldsymbol{s},a} = \boldsymbol{s}^T(\mathbf{A} + a \cdot \mathbf{G}) + \boldsymbol{e} \in \mathbb{Z}_q^m$$

for error vector $\boldsymbol{e} \leftarrow \chi^m$ *and the gadget matrix* $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$.

In our construction, all LWE encodings will be encoded using the same LWE secret $\boldsymbol{s}$, thus for simplicity, we will simply refer to such an encoding as $\psi_{\mathbf{A},a}$.

For homomorphic addition, the addition algorithm takes as input two encodings $\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a}$, and outputs the sum of them. Let $\mathbf{A}^+ = \mathbf{A} + \mathbf{A}'$ and $a^+ = a + a'$

$$\mathsf{Add}(\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'}) = \psi_{\mathbf{A},a} + \psi_{\mathbf{A}',a'} = \psi_{\mathbf{A}^+,a^+}$$

For homomorphic multiplication, the multiplication algorithm takes as input two encodings $\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a}$, and outputs an encoding $\psi_{\mathbf{A}^\times,a^\times}$, where $\mathbf{A}^\times = -\mathbf{A}\mathbf{G}^{-1}(\mathbf{A}')$ and $a^\times = aa'$.

$$\mathsf{Mult}(\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'}) = -\psi \cdot \mathbf{G}^{-1}(\mathbf{A}') + a \cdot \psi' = \psi_{\mathbf{A}^\times,a^\times}$$

Public Key Evaluation Algorithm. Following the notation in [83], we define a public evaluation algorithm $\mathsf{Eval}_{\mathsf{PK}}$. The algorithm takes as input a description of the branching program BP, a collection of public keys $\{\mathbf{A}_i\}_{i \in [\ell]}$ (one for each attribute bit x_i), a collection of public keys $\mathbf{V}_{0,i}$ for initial state vector and an auxiliary matrix $\mathbf{A}^c$, and outputs an evaluated public key corresponding to the branching program BP.

$$\mathbf{V}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{PK}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i \in [\ell]}, \{\mathbf{V}_{0,i}\}_{i \in [5]}, \mathbf{A}^c)$$

where the auxiliary matrix $\mathbf{A}^c$ are used to encoded constant 1 for each input wire. We also define matrix $\mathbf{A}'_i = \mathbf{A}^c - \mathbf{A}_i$ as a public key used to encode $1 - x_i$. By the definition of branching programs, the output $\mathbf{V}_{\mathsf{BP}} \in \mathbb{Z}_q^{n \times m}$ is the homomorphically generated public key $\mathbf{V}_{L,1}$ at position 1 of the state vector for the L-th step of the branching program evaluation.

Recall that in the definition of branching programs, BP is represented by the tuple $\{\mathsf{var}(t), \{\gamma_{t,i,0}, \gamma_{t,i,1}\}_{i \in [5]}\}$ for $t \in [L]$, and the initial state vector is set to be $\boldsymbol{v}_0 = (1, 0, 0, 0, 0)$. Further, for $t \in [L]$, the computation is performed as $\boldsymbol{v}_t[i] = \boldsymbol{v}_{t-1}[\gamma_{t,i,0}](1 - x_{\mathsf{var}(t)}) + \boldsymbol{v}_{t-1}[\gamma_{t,i,1}] \cdot x_{\mathsf{var}(t)}$. It is important for the security proof (among other reasons) that the evaluated state vector in each step is independent of the attribute vector.

Encoding Evaluation Algorithm. We define an encoding evaluation algorithm $\mathsf{Eval}_{\mathsf{CT}}$ that takes as input the description of a branching program BP, an attribute vector $\boldsymbol{x}$, a set of encodings for the attribute $\{\mathbf{A}_i, \psi_i := \psi_{\mathbf{A}_i, x_i}\}_{i \in [\ell]}$, encodings of the initial state vector $\{\mathbf{V}_{0,i}, \psi_{0,i} := \psi_{\mathbf{V}_{0,i} v_0[i]}\}_{i \in [5]}$ and an encoding of a constant 1, i.e., $\psi^c := \psi_{\mathbf{A}^c, 1}$. The algorithm $\mathsf{Eval}_{\mathsf{CT}}$ outputs an encoding of the result $y := \mathsf{BP}(\boldsymbol{x})$ with respect to the homomorphically derived public key $\mathbf{V}_{\mathsf{BP}} := \mathbf{V}_{L,1}$

$$\psi_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{A}_i, \psi_i\}_{i \in [\ell]}, \{\mathbf{V}_{0,i}, \psi_{0,i}\}_{i \in [5]}, \{\mathbf{A}^c, \psi^c\})$$

As mentioned above, in branching program computation, for $t \in [L]$, we have for all $i \in [5]$

$$\boldsymbol{v}_t[i] = \boldsymbol{v}_{t-1}[\gamma_{t,i,0}](1 - x_{\mathsf{var}(t)}) + \boldsymbol{v}_{t-1}[\gamma_{t,i,1}] \cdot x_{\mathsf{var}(t)}$$

The evaluation algorithm proceeds inductively to update the encoding of the state vector for each step of the branching program. Next, we need to in-

stantiate this inductive computation using the homomorphic operations described above, i.e., $\mathsf{Add}, \mathsf{Mult}$. Following the notation used in [83], we define $\psi'_i := \psi_{\mathbf{A}'_i,(1-x_i)} = \boldsymbol{s}^T(\mathbf{A}'_i + (1-x_i)\mathbf{G}) + \boldsymbol{e}'_i$, where $\mathbf{A}'_i = \mathbf{A}^c - \mathbf{A}_i$, to denote the encoding of $1 - x_i$. This encoding can be computed using $\mathsf{Add}(\psi_{\mathbf{A}_i^c,1}, -\psi_{\mathbf{A}_i,x_i})$. Then assuming at time $t-1 \in [L]$ we hold encodings of the state vector $\{\psi_{\mathbf{v}_{t-1},i,\boldsymbol{v}_{t-1}[i]}\}_{i\in[5]}$. For $i \in [5]$, we compute the encodings of new state values as

$$\psi_{i,t} = \mathsf{Add}(\mathsf{Mult}(\psi'_{\mathsf{var}(t)}, \psi_{t-1,\gamma_0}), \mathsf{Mult}(\psi_{\mathsf{var}(t)}, \psi_{t-1,\gamma_1}))$$

where $\gamma_0 := \gamma_{t,i,0}$ and $\gamma_1 := \gamma_{t,i,1}$. We omit the correctness proof of the encoding here, which is presented in [83].

Simulated Public Key Evaluation Algorithm. The simulation strategy was first developed in [31], and then adapted by Gorbunov and Vinayagamurthy [83] in branching program scenario. In particular, set $\mathbf{A}_i = \mathbf{A}_i\mathbf{R}_i - x_i\mathbf{G}$ for some shared public key matrix $\mathbf{A}$ and low norm matrix $\mathbf{R}_i$. Similarly, the state public keys $\mathbf{A}_{t,i} = \mathbf{A}\mathbf{R}_{t,i} - \boldsymbol{v}_t[i]\mathbf{G}$, and matrices $\mathbf{A}^c = \mathbf{A}\mathbf{R}^c - \mathbf{G}$. The evaluation algorithm $\mathsf{Eval}_{\mathsf{Sim}}$ takes as input the description of branching program BP, the attribute vector $\boldsymbol{x}$, collections of low norm matrices $\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c$ corresponding to input public key, initial state vector and complement matrices respectively, and a shared matrix $\mathbf{A}$. It outputs a homomorphically derived low norm matrix $\mathbf{R}_{\mathsf{BP}}$:

$$\mathbf{R}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{Sim}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c, \mathbf{A})$$

In particular, let $\mathbf{R}'_i = \mathbf{R}_i^c - \mathbf{R}_i$ for $i \in [\ell]$. We derive the low-norm matrices $\mathbf{R}_{t,i}$ for $i \in [5]$ as

1. Let $\gamma_0 := \gamma_{t,i,0}$ and $\gamma_1 := \gamma_{t,i,1}$.

2. Compute

$$\begin{aligned}\mathbf{R}_{t,i} = (-\mathbf{R}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) + (1 - x_{\mathsf{var}(t)}) \cdot \mathbf{R}_{t-1,\gamma_0}) \\ + (-\mathbf{R}_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1}) + x_{\mathsf{var}(t)} \cdot \mathbf{R}_{t-1,\gamma_1})\end{aligned} \tag{3.1}$$

We let $\mathbf{R}_{L,1}$ be the matrix obtained at L-th step corresponding to state value 1 by the above algorithm. The correctness requires the norm of $\mathbf{R}_{\mathsf{BP}}$ remains small and the matrix $\mathbf{V}_{\mathsf{BP}}$ output by $\mathsf{Eval}_{\mathsf{PK}}$ satisfies $\mathbf{V}_{\mathsf{BP}} = \mathbf{A}\mathbf{R}_{\mathsf{BP}} - \mathsf{BP}(\boldsymbol{x})\mathbf{G}$. We refer to the counterpart in [83] for the detailed correctness proof.

In order to achieve correctness and deniability, it is important for us to both lower and upper bound the norm of $\|\mathbf{R}_{\mathsf{BP}}\|$. Here we apply the triangular inequality of the norm and obtain the following lemma:

Lemma 3.2.2. *Let $\mathbf{R}_{i,j}$'s be the matrices defined as above. Then for every $t \in [\ell], i \in [5]$ and every error vector $\boldsymbol{e} \in \mathbb{Z}_q^m$, we have $\|\boldsymbol{e}^T \cdot \mathbf{R}_{t-1,j}\| - \Theta(m^{1.5}) \cdot \|\boldsymbol{e}\| \leq \|\boldsymbol{e}^T \cdot \mathbf{R}_{t,i}\| \leq \|\boldsymbol{e}\| \cdot \|\mathbf{R}_{t-1,j}\| + \Theta(m^{1.5}) \cdot \|\boldsymbol{e}\|$, where $j = \gamma_{x_{\mathsf{var}(t)}}$.*

Proof. Recall the matrix $\mathbf{R}_{i,j}$ is computed as

$$\mathbf{R}_{t,i} = (-\mathbf{R}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) + (1 - x_{\mathsf{var}(t)}) \cdot \mathbf{R}_{t-1,\gamma_0}) + (-\mathbf{R}_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1}) + x_{\mathsf{var}(t)} \cdot \mathbf{R}_{t-1,\gamma_1})$$

where $x_{\mathsf{var}(t)} \in \{0, 1\}$. Without loss of generality, we assume $x_{\mathsf{var}(t)} = 1$, thus we obtain

$$\mathbf{R}_{t,i} = -\mathbf{R}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) + \mathbf{R}_{t-1,\gamma_1} - \mathbf{R}_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1})$$

Since $\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1}) \in \{0, 1\}^{m \times m}$, we know $\|\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1})\| \leq m$. Since matrices $\mathbf{R}_{\mathsf{var}(t)}, \mathbf{R}'_{\mathsf{var}(t)}$ were chosen uniformly at random in $\in \{-1, 1\}^{m \times m}$, we know that their norm is bounded by $\Theta(\sqrt{m})$ with high probability by Lemma 2.3.5. Therefore, we can bound the norm of term $\|\mathbf{R}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) + \mathbf{R}_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1})\| \leq \Theta(m^{1.5})$.

By applying the triangular inequality, it holds for every $t \in [\ell], i \in [5]$ and vector $\boldsymbol{e} \in \mathbb{Z}_q^m$,

$$\|\boldsymbol{e}^T \cdot \mathbf{R}_{t-1,j}\| - \Theta(m^{1.5})\|\boldsymbol{e}\| \leq \|\boldsymbol{e}^T \cdot \mathbf{R}_{t,i}\| \leq \|\boldsymbol{e}\| \cdot \|\mathbf{R}_{t-1,j}\| + \Theta(m^{1.5})\|\boldsymbol{e}\|$$

where $j = \gamma_{x_{\mathsf{var}(t)}}$. □

By applying the above lemma inductively on the equation (3.1) of computing matrix $\mathbf{R}_{\mathsf{BP}}$ for input length ℓ, we can obtain the following theorem:

Theorem 3.2.3. *Let* BP *be a length ℓ branching program, and $\mathbf{R}_{\mathsf{BP}}$ be the matrix as defined above. Then we have $\|\boldsymbol{e}^T \cdot \mathbf{R}_{0,j}\| - 2m^{1.5}\ell\|\boldsymbol{e}\| \leq \|\boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}}\| \leq \|\boldsymbol{e}^T\| \cdot \|\mathbf{R}_{0,j}\| + 2m^{1.5}\ell\|\boldsymbol{e}\|$ for some $j \in [5]$.*

Proof. Applying Lemma 3.2.2 inductively on input length ℓ, we have

$$\|\boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}}\| \geq \|\boldsymbol{e}^T \cdot \mathbf{R}_{\ell-1,j}\| - 2m^{1.5}\|\boldsymbol{e}\| \geq \cdots \geq \|\boldsymbol{e}^T \cdot \mathbf{R}_{0,j}\| - 2m^{1.5}\ell\|\boldsymbol{e}\|$$

We can obtain the upper bound of $\|\boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}}\|$ using similar computation. □

Lemma 3.2.4. *Let $\mathbf{R}$ is an $m \times m$ be a matrix chosen at random from $\{-1, 1\}^{m \times m}$, and $\boldsymbol{u} = (u_1, ..., u_m) \in \mathbb{R}^m$ be a vector chosen according to the m dimensional Gaussian with width α. Then we have*

$$\Pr\left[\|\boldsymbol{u}^T\mathbf{R}\|^2 \in \Theta(m^2\alpha^2)\right] > 1 - \mathsf{negl}(m).$$

Proof. We know with overwhelming probability over the choice of $\boldsymbol{u}$, all of its entries have absolute value less than $B = \alpha\omega(\log m)$. Also, we know that with overwhelming probability, we have $\|\boldsymbol{u}\|^2 = \Theta(m\alpha^2)$. We call a sample typical if it satisfies these two conditions. Note that it is without loss of generality to just consider the typical samples, from a simple union bound argument.

Then we consider a fixed typical choice of vector $\boldsymbol{u} = (u_1, ..., u_m) \in \mathbb{R}^m$. We write the inner product of $\boldsymbol{u}^T \cdot \boldsymbol{r}$ where $\boldsymbol{r} = (r_1, \ldots, r_m)$ is sampled uniformly from $\{-1, 1\}^m$. We observe that $\mathbb{E}\left[\|\boldsymbol{u}^T \cdot \boldsymbol{r}\|^2\right] = \mathbb{E}\left[\sum_{i=1}^m r_i^2 u_i^2 + \sum_{i<j\leq m} r_i r_j u_i u_j\right] = \sum_{i=1}^m u_i^2 = \|\boldsymbol{u}\|^2$. This is because each r_i, r_j are independent and have mean 0.

Now, for such a fixed $\boldsymbol{u}$ we denote random variables $X_1, \ldots, X_m$ be i.i.d. samples of $\boldsymbol{r}^T \boldsymbol{u}$. It is not hard to see that

- $\|\boldsymbol{u}^T \mathbf{R}\|^2 = X_1^2 + X_2^2 + \cdots + X_m^2$, (one can view X_i as the i-th entry of $\boldsymbol{u}^T \mathbf{R}$),
- $\mathbb{E}\left[\|\boldsymbol{u}^T \mathbf{R}\|^2\right] = m\|\boldsymbol{u}\|^2$.

Next we claim that for each i, $X_i^2 \leq mB^2\omega(\log m)$ with overwhelming probability. By Hoeffding's inequality, we have

$$\Pr\left[\left|\sum_{j\in[m]} r_j u_j\right| > t\right] < 2e^{-\frac{2t^2}{m\cdot 4B^2}}.$$

This is because each $r_j u_j \in [-B, B]$. (Recall that we consider a fixed $\boldsymbol{u}$ for the typical case). By setting $t = \sqrt{m}B\omega(\log m)$, we have $\Pr[|X_i| > t] < \mathsf{negl}(m)$. Thus $X_i^2 \leq mB^2\omega(\log m)$ with overwhelming probability. So we can consider truncated versions of X_i^2's, where we cut out the large samples. This will only induce a negligible statistical distance, and change the expectation by a negligible amount. For simplicity of presentation, we still use the notation X_i^2's in the following arguments, but the reader should keep in mind that they were truncated.

Next again we apply Hoeffding's inequality to the X_i^2's to obtain

$$\Pr\left[\left|\|\boldsymbol{u}^T \mathbf{R}\|^2 - m\|\boldsymbol{u}\|^2\right| > t'\right] < 2e^{-\frac{2t'^2}{\sum_{i=1}^m (mB^2\omega(\log m))^2}} = 2e^{-\frac{2t'^2}{m^3 B^4 \omega(\log m)}}.$$

By taking $t' = m\|\boldsymbol{u}\|^2/2$, we have

$$\Pr\left[\left|\|\boldsymbol{u}^T \mathbf{R}\|^2 - m\|\boldsymbol{u}\|^2\right| > t'\right] < 2e^{-\frac{\|\boldsymbol{u}\|^4}{2mB^4\omega(\log m)}}.$$

Since $\boldsymbol{u}$ is typical, we know that $\|\boldsymbol{u}\|^2 = \Theta(m\alpha^2)$. Also recall that $B = \alpha\omega(\log m)$. So we have

$$\Pr\left[\|\boldsymbol{u}^T\mathbf{R}\|^2 \in \Theta(m^2\alpha^2)\right] > 1 - 2e^{-\frac{m}{\omega(\log m)}} = 1 - \mathsf{negl}(m).$$

This completes the proof. □

3.2.2 Construction of Flexibly Bi-Deniable ABE for Branching Programs

In this part, we present our flexibly bi-deniable AB-BTS scheme for bounded-length Branching Programs. We use a semantically-secure public key encryption $\Pi = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ with message space $\mathcal{M}_\Pi = \mathbb{Z}_q^{m\times m}$ and ciphertext space C_Π. For a family of branching programs of length bounded by L and input space $\{0,1\}^\ell$, the description of BiDenAB-BTS = (Setup, DenSetup, Keygen, SampleP, SampleU, TestP, FakeRCoins, FakeSCoins) are as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^\ell)$: On input the security parameter λ, the length of the branching program L and length of the attribute vector ℓ,

  1. Set the LWE dimension be $n = n(\lambda)$, modulus $q = q(n, L)$. Choose Gaussian distribution parameter $s = s(n)$. Let $\mathsf{params} = (n, q, m, s)$.
  2. Sample one random matrix associated with its trapdoor as

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

  3. Choose $\ell + 6$ random matrices $\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c$ from $\mathbb{Z}_q^{n\times m}$.

4. Choose a random vector $\boldsymbol{u} \in \mathbb{Z}_q^n$.

5. Compute a public/secret key pair $(\mathsf{PK}', \mathsf{SK}')$ for a semantically secure public key encryption $(\mathsf{PK}', \mathsf{SK}') \leftarrow \mathsf{Gen}'(1^\lambda)$

6. Output the public parameter PP and master secret key MSK as

$$\mathsf{PP} = (\mathsf{params}, \mathbf{A}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c, \boldsymbol{u}, \mathsf{PK}'), \quad \mathsf{MSK} = (\mathbf{T}_\mathbf{A}, \mathsf{SK}')$$

- $\mathsf{DenSetup}(1^\lambda, 1^L, 1^\ell)$: On input the security parameter λ, the length of branching program L and length of attribute vector ℓ, the deniable setup algorithm runs the same computation as setup algorithm, and outputs

$$\mathsf{PP} = (\mathsf{params}, \mathbf{A}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c, \boldsymbol{u}, \mathsf{PK}'), \quad \mathsf{MSK} = (\mathbf{T}_\mathbf{A}, \mathsf{SK}') \quad \mathsf{fk} = (\mathbf{T}_\mathbf{A}, \mathsf{SK}')$$

- $\mathsf{Keygen}(\mathsf{MSK}, \mathsf{BP})$: On input the master secret key MSK and the description of a branching program BP, $\mathsf{BP} = (\boldsymbol{v}_0, \{\mathsf{var}(t), \{\gamma_{t,i,0}, \gamma_{t,i,1}\}_{i\in[5]}\}_{t\in[L]})$.

  1. Homomorphically compute a public matrix with respect to the branching program BP: $\mathbf{V}_\mathsf{BP} \leftarrow \mathsf{Eval}_\mathsf{PK}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$.

  2. Sample a low norm vector $\boldsymbol{r}_\mathsf{BP} \in \mathbb{Z}_q^{2m}$, using

$$\boldsymbol{r}_\mathsf{BP} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_\mathbf{A}, (\mathbf{V}_\mathsf{BP} + \mathbf{G}), \boldsymbol{u}, sq)$$

     such that $\boldsymbol{r}_\mathsf{BP}^T \cdot [\mathbf{A}|\mathbf{V}_\mathsf{BP} + \mathbf{G}] = \boldsymbol{u}$.

  3. Output the secret key SK_BP for branching program as $\mathsf{SK}_\mathsf{BP} = (\boldsymbol{r}_\mathsf{BP}, \mathsf{BP})$.

- $\mathsf{SampleP}(\mathsf{PP}, \boldsymbol{x})$: On input public parameters PP and attribute $\boldsymbol{x}$,

  1. Choose an LWE secret $\boldsymbol{s} \in \mathbb{Z}_q^n$ uniformly at random.

  2. Choose noise vector $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}_q^m, \alpha}$, and compute $\psi_0 = \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}$.

  3. Choose one random matrices $\mathbf{R}^c \leftarrow \{-1, 1\}^{m\times m}$, and let $\boldsymbol{e}^c = \boldsymbol{e}^T\mathbf{R}^c$. Compute an encoding of constant 1: $\psi^c = \boldsymbol{s}^T(\mathbf{A}^c + \mathbf{G}) + \boldsymbol{e}^c$.

4. Encode each bit $i \in [\ell]$ of the attribute vector:

   (a) Choose a random matrix $\mathbf{R}_i \leftarrow \{-1,1\}^{m\times m}$, and let $\boldsymbol{e}_i = \boldsymbol{e}^T\mathbf{R}_i$.

   (b) Compute $\psi_i = \boldsymbol{s}^T(\mathbf{A}_i + x_i\mathbf{G}) + \boldsymbol{e}_i$.

5. Encode the initial state vector $\boldsymbol{v}_0 = (1,0,0,0,0)$, for $i \in [5]$

   (a) Choose a random matrix $\mathbf{R}'_{0,i} \leftarrow \{-1,1\}^{m\times m}$, and let $\mathbf{R}_{0,i} = \eta\mathbf{R}'_{0,i}, \boldsymbol{e}_{0,i} = \boldsymbol{e}^T\mathbf{R}_{0,i}$, where the noise scaling parameter η is set in Section 3.2.3.

   (b) Compute $\psi_{0,i} = \boldsymbol{s}^T(\mathbf{A}_i + \boldsymbol{v}_0[i]\mathbf{G}) + \boldsymbol{e}_{0,i}$.

6. Compute $c = \boldsymbol{s}^T\boldsymbol{u} + e$, where $e \leftarrow \mathcal{D}_{\mathbb{Z}_q,s}$

7. Use PKE to encrypt randomly chosen matrices $\mathbf{R}^c, \{\mathbf{R}_i\}_{i\in[\ell]}$ and $\{\mathbf{R}_{0,i}\}_{i\in[5]}$:

$$\mathbf{T}_i \leftarrow \mathsf{Enc}'(\mathsf{PK}', \mathbf{R}_i), \mathbf{T}^c \leftarrow \mathsf{Enc}'(\mathsf{PK}', \mathbf{R}^c), \mathbf{T}_{0,i} \leftarrow \mathsf{Enc}'(\mathsf{PK}', \mathbf{R}_{0,i})$$

8. Output the ciphertext

$$\mathsf{CT}_{\boldsymbol{x}} = (\boldsymbol{x}, \psi_0, \{\psi_i\}_{i\in[\ell]}, \psi^c, \{\psi_{0,i}\}_{i\in[5]}, c, \{\mathbf{T}_i\}_{i\in[\ell]}, \mathbf{T}^c, \{\mathbf{T}_{0,i}\}_{i\in[5]})$$

- $\mathsf{SampleU}(\mathsf{PP}, \boldsymbol{x})$: Output a uniformly random vector $\mathsf{CT} \in \mathbb{Z}_q^m \times \mathbb{Z}_q^{\ell m} \times \mathbb{Z}_q^{\ell m} \times \mathbb{Z}_q^{5m} \times \mathbb{Z}_q \times C_\Pi^\ell \times C_\Pi \times C_\Pi^5$.

- $\mathsf{TestP}(\mathsf{SK}_{\mathsf{BP}}, \mathsf{CT}_{\boldsymbol{x}})$: On input the secret key $\mathsf{SK}_{\mathsf{BP}}$ for a branching program BP and a ciphertext associated with attribute $\boldsymbol{x}$, if $\mathsf{BP}(\boldsymbol{x}) = 0$, output $\perp$, otherwise,

  1. Homomorphically compute the evaluated ciphertext of result $\mathsf{BP}(\boldsymbol{x})$

$$\psi_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{A}_i, \psi_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \psi_{0,i}\}_{i\in[5]}, \{\mathbf{A}_i^c, \psi_i^c\}_{i\in[\ell]})$$

  2. Then compute $\phi = [\psi_0|\psi_{\mathsf{BP}}]^T \cdot \boldsymbol{r}_{\mathsf{BP}}$. Accept $\mathsf{CT}_{\boldsymbol{x}}$ as a P-sample if $|c - \phi| < 1/4$, otherwise reject.

- $\mathsf{FakeSCoins}(r_S)$: Simply output the P-sample $\mathbf{c}$ as the randomness r_S^* that would cause $\mathsf{SampleU}$ to output $\boldsymbol{c}_{\boldsymbol{x}}$.

- $\mathsf{FakeRCoins}(\mathsf{PP}, \mathsf{fk}, \mathsf{CT}_{\boldsymbol{x}}, \mathsf{BP})$: On input the public parameters PP, the faking key fk, a ciphertext $\mathsf{CT}_{\boldsymbol{x}}$ and description of a branching program BP

  1. If $\mathsf{BP}(\boldsymbol{x}) \neq 0$, then output $\mathsf{SK}_f \leftarrow \mathsf{Keygen}(\mathsf{fk}, \mathsf{BP})$.

  2. Otherwise, parse ciphertext $\mathsf{CT}_{\boldsymbol{x}}$ as

  $$\mathsf{CT}_{\boldsymbol{x}} = (\boldsymbol{x}, \psi_0, \{\psi_i\}_{i\in[\ell]}, \psi^c, \{\psi_{0,i}\}_{i\in[5]}, c, \{\mathbf{T}_i\}_{i\in[\ell]}, \mathbf{T}^c, \{\mathbf{T}_{0,i}\}_{i\in[5]})$$

  Compute $\boldsymbol{e} \leftarrow \mathsf{Invert}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \psi_0)$. Then decrypt $(\{\mathbf{T}_i\}_{i\in[\ell]}, \mathbf{T}^c, \{\mathbf{T}_{0,i}\}_{i\in[5]})$ respectively using $\mathsf{Dec}(\mathsf{SK}', \cdot)$ to obtain $\{\mathbf{R}_i\}_{i\in[\ell]}, \mathbf{R}^c, \{\mathbf{R}_{0,i}\}_{i\in[5]}$. Compute evaluated error

  $$\boldsymbol{e}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{A}_i, \boldsymbol{e}^T\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \boldsymbol{e}^T\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c, \boldsymbol{e}^T\mathbf{R}^c\})$$

  such that $\boldsymbol{e}_{\mathsf{BP}} = \boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}}$.

  3. Homomorphically compute a public matrix with respect to the branching program BP: $\mathbf{V}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{PK}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \{\mathbf{A}_i^c\}_{i\in[\ell]})$. Then sample a properly distributed secret key $\boldsymbol{r}_{\mathsf{BP}} \in \mathbb{Z}_q^{2m}$, using

  $$\boldsymbol{r}_{\mathsf{BP}} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, (\mathbf{V}_{\mathsf{BP}} + \mathbf{G}), \boldsymbol{u}, s)$$

  4. Sample correlation vector $\mathbf{y}_0 \leftarrow \mathcal{D}_{\mathbb{Z}_q^m, \beta^2 q^2 \mathbf{I}_{m\times m}}$. Then sample correlation coefficient $\mu \leftarrow \mathcal{D}_\gamma$, and set vector $\boldsymbol{y}_1 = (\mu \boldsymbol{e}_{\mathsf{BP}} + \mathcal{D}_{\mathbf{Z}^m, \mathbf{Q}})q$, where

  $$\mathbf{Q} = \beta^2 \mathbf{I}_{m\times m} - \gamma^2\alpha^2 \mathbf{R}_{\mathsf{BP}}^T \mathbf{R}_{\mathsf{BP}} \tag{3.2}$$

  5. Let $\boldsymbol{y} = (\boldsymbol{y}_0 | \boldsymbol{y}_1)$, then sample and output the faked secret key $\mathsf{SK}_{\mathsf{BP}}^* = \mathbf{r}_{\mathsf{BP}}^*$ as $\mathbf{r}_{\mathsf{BP}}^* \leftarrow \boldsymbol{y} + \mathcal{D}_{\Lambda + \boldsymbol{r}_{\mathsf{BP}} - \boldsymbol{y}, \sqrt{s^2 - \beta^2}}$, using $\mathsf{SampleD}(\mathsf{ExtBasis}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{V}_{\mathsf{BP}} + \mathbf{G}), \boldsymbol{r}_{\mathsf{BP}} - \boldsymbol{y}, \sqrt{s^2 - \beta^2})$, where $\Lambda = \Lambda^\perp([\mathbf{A}|\mathbf{V}_{\mathsf{BP}} + \mathbf{G}])$.

The SampleP algorithm is similar to the ABE ciphertexts in the work [83], except that we add another scaling factor η to the rotation matrices $\mathbf{R}_{0,i}$'s. This

allows us to both upper and lower bound the noise growth, which is essential to achieve bi-deniability. As we discussed in the introduction, the FakeRCoins embeds the evaluated noise into the secret key, so that it will change the decrypted value of the targeted ciphertext, but not others. Next we present the theorem we achieve and a high level ideas of the proof. We describe the intuition of our proof as follows.

Overview of Our Security Proof. At a high level, our security proof begins at the Fake experiment (cf. Definition 3.1.1 for a formal description), where first a ciphertext CT^* and its associated noise terms $\boldsymbol{e}^*$ are sampled, then a fake key $\boldsymbol{r}^*$ is generated that "artificially" fails to decrypt any ciphertext with noise vector (oriented close to) $\boldsymbol{e}^*$. In the end, we will arrive at the Real experiment, where an honest key $\boldsymbol{r}$ is generated that "genuinely" fails to decrypt the honestly generated, coerced ciphertext CT^*. (Multi-CT coercion security follows by a standard hybrid argument that repeatedly modifies respective $\boldsymbol{r}^*$ to $\boldsymbol{r}$ for each coerced CT^* in order.) In order to transition from Fake to Real, we move through a sequence of computationally- or statistically-indistinguishable hybrid experiments.

The first set of intermediate experiments (represented by Hyb_1 and Hyb_2 in our formal proof) embeds the attribute $\boldsymbol{x}$ of the challenge ciphertext CT^* in the public parameters, in a similar fashion to the beginning of every SIM-secure proof of lattice-based ABE. Indistinguishability follows via the Leftover Hash Lemma [58]. (Note that the additional hybrid in our proof is used to ensure that the random rotation matrices $\mathbf{R}$ employed by the LHL for public key embedding of $\boldsymbol{x}$ are the *exact same* matrices $\mathbf{R}$ as used to generate the noise terms of the coerced CT^*, and uses the security of any semantically-secure PKE for computational indistinguishability.)

The next set of intermediate experiments (given by $\mathsf{Hyb}_3, \mathsf{Hyb}_4$, and Hyb_5 in our formal proof) perform the "main, new work" of our security proof. Specifically, they "swap the order" of the generation of the PK matrices $\{\mathbf{A}\}$, the public coset $\boldsymbol{u}$ (in the public parameters and in the coerced ciphertext), and the error vector(s) $\boldsymbol{e}$ in the coerced ciphertext components. (An additional hybrid is used to toggle the order of a "correlation vector" $\boldsymbol{y}$ – a random, planted vector used to allow for a more modular analysis of these steps.) In each case, we give a *statistical* argument that the adversary's view in adjacent hybrids is indistinguishable or identical, using elementary properties of multi-dimensional Gaussians.

In the next step (given by Hyb_6), we apply the eLWE^+ assumption to (roughly) change every component of the coerced ciphertext CT^* to uniform – except for the final c^* component used to blind the message μ.

In the final step (given by Hyb_7), we transition to the Real experiment by changing the c^* component to uniform (in the presence of Dual Regev decryption under honest $\boldsymbol{z}$), using our sharper noise analysis as described above to show statistical indistinguishability of the final decryption output of $\boldsymbol{z}$ on CT^*.

Theorem 3.2.5. *Assuming the hardness of extended-*$\mathsf{LWE}_{q,\beta'}$*, the above algorithms form a secure attribute-based bitranslucent set schemem, as in Definition 3.1.2.*

Lemma 3.2.6. *For parameters set in Section 3.2.3, the AB-BTS defined above satisfies the correctness property in Definition 3.1.2.*

Proof. As we mentioned in Remark 3.1.3, the correctness of faking algorithms is implied by the bi-deniability property. Therefore, we only need to prove the correctness of normal decryption algorithm. For branching program BP and

input $\boldsymbol{x}$, such that $\mathsf{BP}(\boldsymbol{x}) = 1$, we compute $\psi_{t,i}$ for $t \in [\ell]$ as

$$\begin{aligned}
\psi_{t,i} &= \mathsf{Add}(\mathsf{Mult}(\psi'_{\mathsf{var}(t)}, \psi_{t-1,\gamma_0}), \mathsf{Mult}(\psi_{\mathsf{var}(t)}, \psi_{t-1,\gamma_1})) \\
&= \mathsf{Add}\Big([\boldsymbol{s}^T(-\mathbf{A}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) + (\boldsymbol{v}_t[\gamma_0] \cdot (1 - x_{\mathsf{var}(t)})) \cdot \mathbf{G}) + \boldsymbol{e}_1], \\
&\qquad ([\boldsymbol{s}^T(-\mathbf{A}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1}) + (\boldsymbol{v}_t[\gamma_1] \cdot x_{\mathsf{var}(t)}) \cdot \mathbf{G}) + \boldsymbol{e}_2]\Big) \\
&= \boldsymbol{s}^T\Big[\underbrace{(-\mathbf{A}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) - \mathbf{A}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1}))}_{\mathbf{V}_{t,i}} \\
&\qquad + \underbrace{(\boldsymbol{v}_t[\gamma_0] \cdot (1 - x_{\mathsf{var}(t)}) + \boldsymbol{v}_t[\gamma_1] \cdot x_{\mathsf{var}(t)})}_{\boldsymbol{v}_t[i]} \cdot \mathbf{G}\Big] + \boldsymbol{e}_{t,i}
\end{aligned}$$

At the end of the ciphertext evaluation, since $\mathsf{BP}(\boldsymbol{x}) = 1$, we can obtain $\psi_{\mathsf{BP}} = \boldsymbol{s}^T(\mathbf{V}_{\mathsf{BP}}+\mathbf{G})+\boldsymbol{e}_{\mathsf{BP}}$, where $\boldsymbol{e}_{\mathsf{BP}} = \boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}}$. Recall that the secret key $\mathsf{SK} = \boldsymbol{r}_{\mathsf{BP}}$ satisfying $[\mathbf{A}|\mathbf{V}_{\mathsf{BP}} + \mathbf{G}] \cdot \boldsymbol{r}_{\mathsf{BP}} = \boldsymbol{u}$. Then for $c - [\psi_0|\psi_{\mathsf{BP}}] \cdot \boldsymbol{r}_{\mathsf{BP}}$, it holds that

$$c - [\psi_0|\psi_{\mathsf{BP}}]^T \cdot \boldsymbol{r}_{\mathsf{BP}} = e - \boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}} \cdot \boldsymbol{r}_{\mathsf{BP}}$$

Now we need to compute a bound for the final noise term. By applying Theorem 3.2.3, we obtain that

$$\|\boldsymbol{e}^T\| \cdot \|\mathbf{R}_{\mathsf{BP}}\| + 2m^{1.5}\ell\|e\| \leq (2m^{1.5}\ell + \eta\sqrt{m})\|\boldsymbol{e}\| \leq \alpha\sqrt{m}(2m^{1.5}\ell + \eta\sqrt{m}) \cdot sq\sqrt{m} \leq \frac{1}{4}$$

So by setting the parameters appropriately, as in Section 3.2.3, we have that

$$|c - [\psi_0|\psi_{\mathsf{BP}}]^T \cdot \boldsymbol{r}_{\mathsf{BP}}| \leq 1/4$$

and the lemma follows. □

Lemma 3.2.7. *Assuming the hardness of extended-*$\mathsf{LWE}_{q,\beta'}$*, the AB-BTS scheme described above is bi-deniable as defined in Definition 3.1.2.*

Proof. First, we notice that because $\mathsf{SampleU}$ simply outputs its random coins as a uniformly random CT, we can use CT itself as the coins.

We prove the bi-deniability property by a sequence of hybrids Hyb_i with details as follows:

Hybrid Hyb_0: Hybrid Hyb_0 is the same as the view of adversary $\mathcal{A}$ in the right-hand faking experiment in the definition of bi-deniability. We use the fact that algorithm Invert successfully recovers $\boldsymbol{e}$ from CT with overwhelming probability over all randomness in the experiment.

Hybrid Hyb_1: In hybrid Hyb_2, we switch the encryptions of matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c)$ in the ciphertext to encryptions of zero.

Recall that in hybrid Hyb_0, we encrypt the randomness matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c)$ using semantically secure PKE Π, i.e.,

$$\mathbf{T}_i \leftarrow \mathsf{Enc}'(\mathsf{PK}', \mathbf{R}_i), \quad \mathbf{T}^c \leftarrow \mathsf{Enc}'(\mathsf{PK}', \mathbf{R}^c), \quad \mathbf{T}_{0,i} \leftarrow \mathsf{Enc}'(\mathsf{PK}', \mathbf{R}_{0,i})$$

In hybrid Hyb_1, we just set

$$\mathbf{T}_i \leftarrow \mathsf{Enc}'(\mathsf{PK}', \mathbf{0}), \quad \mathbf{T}^c \leftarrow \mathsf{Enc}'(\mathsf{PK}', \mathbf{0}), \quad \mathbf{T}_{0,i} \leftarrow \mathsf{Enc}'(\mathsf{PK}', \mathbf{0})$$

to be encryptions of $\mathbf{0} \in \mathbb{Z}^{m\times m}$ to replace encryptions of matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c)$.

Hybrid Hyb_2: In hybrid Hyb_2, we embed random matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c)$ and challenge attribute $\boldsymbol{x}^*$ in the public parameters PP.

Recall that in hybrid Hyb_1 the matrices $(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$ are sampled at random. In hybrid Hyb_2, we slightly change how these matrices are generated. Let $\boldsymbol{x}^* = (x_1^*, ..., x_\ell^*)$ be the challenge attribute that the adversary $\mathcal{A}$ intends to attack. We sample matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}'_{0,i}\}_{i\in[5]}, \mathbf{R}^c)$ uniformly random from $\{-1, 1\}^{m\times m}$ and set $\mathbf{R}_{0,i} = \eta\mathbf{R}'_{0,i}$, which would be used both in the generation of public parameters and challenge ciphertext. We set $(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$ respectively

as

$$\mathbf{A}_i = \mathbf{A}\mathbf{R}_i - x_i^*\mathbf{G}, \quad \mathbf{V}_{0,i} = \mathbf{A}\mathbf{R}_{0,i} - \boldsymbol{v}_0[i]\mathbf{G}, \quad \mathbf{A}^c = \mathbf{A}\mathbf{R}^c - \mathbf{G}$$

where $\boldsymbol{v}_0 = [1, 0, 0, 0, 0]$. The rest of the hybrid remains unchanged.

Hybrid Hyb_3: In hybrid Hyb_3, we change the generation of matrix $\mathbf{A}$ and vector $\boldsymbol{u}$ in public parameters PP.

Let $\mathbf{A}$ be a random matrix in $\mathbb{Z}_q^{n\times m}$. The construction of matrices $(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$ remains the same, as in hybrid Hyb_2. Sample error vectors $\boldsymbol{e}$ that would be used in algorithm $\mathsf{SampleP}$ later. Then compute the error vector

$$\boldsymbol{e}_{\mathsf{BP}^*} \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{A}_i, \boldsymbol{e}^T\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \boldsymbol{e}^T\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c, \boldsymbol{e}^T\mathbf{R}^c\})$$

and choose a correlation coefficient $\mu \leftarrow \mathcal{D}_\gamma$, and set vector $\boldsymbol{y}_1 = (\mu\boldsymbol{e}_{\mathsf{BP}^*} + \mathcal{D}_{\mathbb{Z}^m,\mathbf{Q}})q$, where

$$\mathbf{Q} = \beta^2\mathbf{I}_{m\times m} - \gamma^2\alpha^2\mathbf{R}_{\mathsf{BP}^*}^T\mathbf{R}_{\mathsf{BP}^*}$$

Then let $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1)$, where $\boldsymbol{y}_0 \leftarrow \mathcal{D}_{\mathbf{Z}_q^m,\beta^2q^2\mathbf{I}_{m\times m}}$. Sample vector $\mathbf{r}_{\mathsf{BP}^*} \leftarrow \boldsymbol{y} + \mathcal{D}_{\mathbb{Z}^{2m}-\boldsymbol{y},(s^2-\beta^2)q^2\mathbf{I}_{2m\times 2m}}$, and compute matrix

$$\mathbf{V}_{\mathsf{BP}^*} \leftarrow \mathsf{Eval}_{\mathsf{PK}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$$

Set vector $\boldsymbol{u}$ in public parameters PP as $\mathbf{u} = [\mathbf{A}|\mathbf{V}_{\mathsf{BP}^*}] \cdot \mathbf{r}_{\mathsf{BP}^*}$. Since $\mathbf{A}$ is a random matrix without trapdoor $\mathbf{T}_\mathbf{A}$ to answer key queries, we will use trapdoor $\mathbf{T}_\mathbf{G}$ to answer queries as follows. Consider a secret key query for branching program BP such that $\mathsf{BP}(\boldsymbol{x}^*) = 0$. To respond, we do the following computations:

1. First, we compute

$$\mathbf{R}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{Sim}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c, \mathbf{A})$$

to obtain a low-norm matrix $\mathbf{R}_{\mathsf{BP}} \in \mathbb{Z}_q^{m\times m}$ satisfying $\mathbf{A}\mathbf{R}_{\mathsf{BP}} - \mathsf{BP}(\boldsymbol{x}^*)\mathbf{G} = \mathbf{V}_{\mathsf{BP}}$.

2. Then, we sample $\boldsymbol{r}_{\mathsf{BP}}$ using

$$\boldsymbol{r}_{\mathsf{BP}} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}_{\mathsf{BP}}, \mathbf{T}_{\mathbf{G}}, \boldsymbol{u}, sq)$$

such that

$$\boldsymbol{r}_{\mathsf{BP}}^T \cdot [\mathbf{A}|\mathbf{V}_{\mathsf{BP}} + \mathbf{G}] = \boldsymbol{u}$$

By Lemma 2.3.7, vector $\boldsymbol{r}_{\mathsf{BP}}$ is distributed as required.

The computation of answering P-sampler query, $\mathsf{SampleP}$ is the same as hybrid Hyb_1 with error vectors $\boldsymbol{e}$, For faking receiver coins, $\mathsf{FakeRCoins}$, simply output the vector $\boldsymbol{r}_{\mathsf{BP}^*}$ pre-sampled in the generation of vector $\boldsymbol{u}$ before.

Hybrid Hyb_4: In hybrid Hyb_4, we change the generation order of vector $\boldsymbol{y}$ and error vector $\boldsymbol{e}$.

First sample vector $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1) \leftarrow \mathcal{D}_{\mathbb{Z}^{2m},\beta^2q^2\mathbf{I}_{2m\times 2m}}$ and compute $\boldsymbol{r}_{\mathsf{BP}^*}$ from $\boldsymbol{y}$ as in previous hybrid. Next, we compute error term $\boldsymbol{e}$ as $\boldsymbol{e} = \nu \boldsymbol{y}_1^T \mathbf{R}_{\mathsf{BP}^*}/q + \mathcal{D}_{\mathbb{Z}^m,\mathbf{Q}'}$, where $\nu \leftarrow \mathcal{D}_\tau, \tau = \gamma\alpha^2/\beta^2$, and $\mathcal{D}_{\mathbb{Z}^m,\mathbf{Q}'}$ is sampled as $\mathbf{L}'\mathcal{D}_{\mathbf{Z}_1^m,\mathbf{I}_{m\times m}}$ for

$$\mathbf{Q}' = \mathbf{L}'\mathbf{L}'^T = \alpha^2\mathbf{I} - \tau^2\beta^2\mathbf{R}_{\mathsf{BP}^*}^T\mathbf{R}_{\mathsf{BP}^*} \tag{3.3}$$

Additionally, we modify the challenge ciphertext to be

$$\psi_0^* = \boldsymbol{s}^T\mathbf{A}/q + \boldsymbol{e}, \quad \psi_i^* = \psi_0^{*T}\mathbf{R}_i/q, \quad \psi_{0,i}^* = \psi_0^{*T}\mathbf{R}_{0,i}/q, \quad \psi^{*c} = \psi_0^{*T}\mathbf{R}^c/q$$

and $c^* = \boldsymbol{s}^T\boldsymbol{u} + \mathcal{D}_{\mathbf{Z}^m,\alpha\mathbf{I}_{m\times m}}$.

Hybrid Hyb_5: In hybrid Hyb_5, we change the generation order of secret key $\boldsymbol{r}_{\mathsf{BP}^*}$ and vector $\boldsymbol{y}$.

We first sample matrix $\boldsymbol{r}_{\mathsf{BP}^*}$ from discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}^{2m},s^2q^2\mathbf{I}_{2m\times 2m}}$, and set vector $\boldsymbol{u}$ in public parameters PP to be $\boldsymbol{u} = [\mathbf{A}|\mathbf{V}_{\mathsf{BP}^*}] \cdot \boldsymbol{r}_{\mathsf{BP}^*}$, where

$$\mathbf{V}_{\mathsf{BP}^*} \leftarrow \mathsf{Eval}_{\mathsf{PK}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \{\mathbf{A}_i^c\}_{i\in[\ell]})$$

Then set $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1) = \boldsymbol{r}_{\mathsf{BP}^*}/2 + \mathcal{D}_{\mathbb{Z}^{2m},(\beta^2-s^2/4)q^2\mathbf{I}_{2m\times 2m}}$. The remainder of the hybrid remains roughly the same. In particular, the challenge ciphertext CT^* is generated in the same manner as Hybrid Hyb_4. We break the noise term $\boldsymbol{e}$ into two terms $\boldsymbol{e} = \boldsymbol{e}_0^{(1)} + \boldsymbol{e}_0^{(2)} + \nu \boldsymbol{y}_1^T \mathbf{R}_{\mathsf{BP}^*}/q$, where $\boldsymbol{e}_0^{(1)} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\beta'\mathbf{I}_{m\times m}}, \boldsymbol{e}_0^{(2)} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\mathbf{Q}'-\beta'^2\mathbf{I}_{m\times m}}$ and $\beta' = \alpha/2$.

Hybrid Hyb_6: In hybrid Hyb_6, we change how the challenge ciphertext is generated by using the Extended-LWE$^+$ instance.

First sample uniformly random vector $\boldsymbol{b} \in \mathbb{Z}^m$ and set the challenge ciphertext as

$$\psi_0^* = \boldsymbol{b}/q + \boldsymbol{e}_0^{(2)}, \quad \psi_i^* = \psi_0^{*T}\mathbf{R}_i/q, \quad \psi_{0,i}^* = \psi_0^{*T}\mathbf{R}_{0,i}/q, \quad \psi^{*c} = \psi_0^{*T}\mathbf{R}^c/q$$

and $c^* = \boldsymbol{r}_{\mathsf{BP}^*}^T[\mathbf{I}_{m\times m}|\mathbf{R}_{\mathsf{BP}^*}](\boldsymbol{b}/q - \boldsymbol{e}_0^{(1)}) + \mathcal{D}_{\mathbb{Z}^m,\alpha\mathbf{I}_{m\times m}}$.

Hybrid Hyb_7: In hybrid Hyb_7, we change the challenge ciphertext to be uniformly random.

In algorithm $\mathsf{SampleP}$, sample uniformly random vectors $\mathsf{CT} \in \mathbb{Z}_q^m \times \mathbb{Z}_q^{\ell m} \times \mathbb{Z}_q^m \times \mathbb{Z}_q^{5m} \times \mathbb{Z}_q$ and outputs CT.

Claim 3.2.8. *Assuming the semantic security of PKE* $\Pi = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$*, hybrid* Hyb_0 *and* Hyb_1 *are computationally indistinguishable.*

Proof. Observe there is only one difference between hybrids Hyb_0 and Hyb_1 occurs in the challenge ciphertext, i.e., the encryption (under PKE Π) of the random matrices $\mathbf{S}_i$ are replaced by encryption of 0. If a ppt adversary $\mathcal{A}$ distinguishes between the Hyb_0-encryptions of $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{R}^c\})$ and the Hyb_1-encryptions of $\mathbf{0}$ with non-negligible probability, then we can construct an efficient reduction $\mathcal{B}$ that uses $\mathcal{A}$ to break the semantic security of PKE Π with similar probability. □

Claim 3.2.9. *Hybrids* Hyb_1 *and* Hyb_2 *are statistically indistinguishable.*

Proof. Observe the only difference between hybrids Hyb_1 and Hyb_2 is the generation of matrices

$$(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \{\mathbf{A}_i^c\}_{i\in[\ell]})$$

The random matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{R}_i^c\}_{i\in[\ell]})$ are used in the generation of public parameters PP:

$$\mathbf{A}_i = \mathbf{A}\mathbf{R}_i - x_i^*\mathbf{G}, \quad \mathbf{V}_{0,i} = \mathbf{A}\mathbf{R}_{0,i} - \boldsymbol{v}_0[i]\mathbf{G}, \quad \mathbf{A}^c = \mathbf{A}\mathbf{R}^c - \mathbf{G}$$

and the construction of errors in challenge ciphertext

$$\boldsymbol{e}_i = \boldsymbol{e}^T\mathbf{R}_i, \quad \boldsymbol{e}^c = \boldsymbol{e}^T\mathbf{R}^c, \quad \boldsymbol{e}_{0,i} = \boldsymbol{e}^T\mathbf{R}_{0,i}$$

Then by Leftover Hash Lemma 5.2.2, the following two distributions are statistically indistinguishable

$$(\mathbf{A}, \{\mathbf{A}\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{A}\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{A}\mathbf{R}^c\}, \widetilde{\boldsymbol{e}}) \approx (\mathbf{A}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c\}, \widetilde{\boldsymbol{e}})$$

where $\widetilde{\boldsymbol{e}} = (\{\boldsymbol{e}_i\}_{i\in[\ell]}, \{\boldsymbol{e}_{0,i}\}_{i\in[5]}, \{\boldsymbol{e}^c\})$. Hence, hybrid Hyb_0 and Hyb_1 are statistically indistinguishable. □

Claim 3.2.10. *Hybrids* Hyb_2 *and* Hyb_3 *are statistically indistinguishable.*

Proof. Observe there are three differences between hybrid Hyb_2 and Hyb_3: The generation of matrix $\mathbf{A}$ and vector $\boldsymbol{u}$ in PP, challenge secret key $\mathsf{SK}_{\mathsf{BP}^*}$ and the computation methods to answer secret key queries. By the property of algorithm $\mathsf{TrapGen}(q, n, m)$ in Lemma 2.3.6, the distribution of matrix $\mathbf{A}$ in hybrid Hyb_2 is statistically close to uniform distribution, from which matrix $\mathbf{A}$ in hybrid Hyb_3 is sampled.

For secret key queries regarding branching program BP, in hybrid Hyb_2, we sample vector $\boldsymbol{r}_{\mathsf{BP}}$, using

$$\boldsymbol{r}_{\mathsf{BP}} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, (\mathbf{V}_{\mathsf{BP}} + \mathbf{G}), \boldsymbol{u}, s)$$

While in hybrid Hyb_3, we sample vector $\boldsymbol{r}_{\mathsf{BP}}$, using

$$\boldsymbol{r}_{\mathsf{BP}} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}_{\mathsf{BP}}, \mathbf{T}_{\mathbf{G}}, \boldsymbol{u}, sq)$$

By setting the parameters appropriately as specified in Section 3.2.3, and the properties of algorithms $\mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ in Lemma 2.3.7, the answers to secret key queries are statistically close.

By Leftover Hash Lemma 5.2.2, the distribution $([\mathbf{A}|\mathbf{V}_{\mathsf{BP}^*}], [\mathbf{A}|\mathbf{V}_{\mathsf{BP}^*}] \cdot \boldsymbol{r}_{\mathsf{BP}^*})$ and $([\mathbf{A}|\mathbf{V}_{\mathsf{BP}^*}], \boldsymbol{u})$ are statistically close. Hence, hybrid Hyb_2 and Hyb_3 are statistically indistinguishable. □

Claim 3.2.11. *Hybrids* Hyb_3 *and* Hyb_4 *are statistically indistinguishable.*

Proof. The only difference between the two experiments is in the choice of $\boldsymbol{y}$ and $\boldsymbol{e}$, specifically, the choice of the $\boldsymbol{y}_1$ component of $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1)$. We will show that the joint distribution of $(\boldsymbol{e}, \boldsymbol{y}_1)$ is identically distributed in these two hybrids:

In hybrid Hyb_3, $\boldsymbol{y}_1$ is set as $\boldsymbol{y}_1 = (\mu \boldsymbol{e}_{\mathsf{BP}^*} + \mathcal{D}_{\mathbb{Z}^m, \mathbf{Q}})q$, where $\mathbf{Q} = \beta^2 \mathbf{I}_{m\times m} - \gamma^2\alpha^2 \mathbf{R}_{\mathsf{BP}^*}^T \mathbf{R}_{\mathsf{BP}^*}$ with $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m, \alpha^2 \mathbf{I}_{m\times m}}$ and

$$\boldsymbol{e}_{\mathsf{BP}^*} \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{A}_i, \boldsymbol{e}^T\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \boldsymbol{e}^T\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c, \boldsymbol{e}^T\mathbf{R}^c\})$$

Therefore, in hybrid Hyb_3, we may write the joint distribution of $(\boldsymbol{e}, \boldsymbol{y}_1)$ as $\mathbf{T}_1 \cdot \mathcal{D}_{\mathbb{Z}^{2m}, \mathbf{I}_{2m\times 2m}}$, where $\mathbf{T}_1 \stackrel{\text{def}}{=} \begin{pmatrix} \alpha \mathbf{I}_{m\times m} & \mathbf{0}_{m\times m} \\ \gamma\alpha q \mathbf{R}_{\mathsf{BP}^*}^T & q\mathbf{L} \end{pmatrix}$ for $\mathbf{Q} = \mathbf{L}\mathbf{L}^T \in \mathbb{Z}^{m\times m}$ via the Cholesky decomposition due to Lemma 2.3.3.

In hybrid Hyb_4, vector $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1)$ is sampled as $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1) \leftarrow \mathcal{D}_{\mathbb{Z}^{2m},\beta^2q^2\mathbf{I}_{2m\times 2m}}$. Then $\boldsymbol{e}$ is computed as $\boldsymbol{e} = \nu\boldsymbol{y}_1^T\mathbf{R}_{\mathsf{BP}^*}/q + \mathcal{D}_{\mathbb{Z}^m,\mathbf{Q}'}$, where $\nu \leftarrow \mathcal{D}_\tau, \tau = \gamma\alpha^2/\beta^2$, and $\mathbf{Q}' = \alpha^2\mathbf{I} - \tau^2\beta^2\mathbf{R}_{\mathsf{BP}^*}^T\mathbf{R}_{\mathsf{BP}^*}$. Then in hybrid Hyb_4, we may write the joint distribution of $(\boldsymbol{e}, \boldsymbol{y}_1)$ as $\mathbf{T}_2 \cdot \mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}}$, where $\mathbf{T}_2 \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{L}' & \tau\beta\mathbf{R}_{\mathsf{BP}^*} \\ \mathbf{0}_{m\times m} & \beta q\mathbf{I}_{m\times m} \end{pmatrix}$ for $\mathbf{Q}' = \mathbf{L}'\mathbf{L}'^T \in \mathbb{Z}^{m\times m}$ via the Cholesky decomposition due to Lemma 2.3.3.

We claim equality of the following systems of equations:

$$\begin{aligned}\mathbf{T}_1\mathbf{T}_1^T &= \begin{pmatrix} \alpha^2\mathbf{I}_{m\times m} & \gamma\alpha^2 q\mathbf{R}_{\mathsf{BP}^*} \\ \gamma\alpha^2 q\mathbf{R}_{\mathsf{BP}^*}^T & \gamma^2\alpha^2q^2\mathbf{R}_{\mathsf{BP}^*}^T\mathbf{R}_{\mathsf{BP}^*} + q^2\mathbf{L}\mathbf{L}^T \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{L}'\mathbf{L}'^T + \tau^2\beta^2\mathbf{R}_{\mathsf{BP}^*}\mathbf{R}_{\mathsf{BP}^*}^T & \tau\beta^2 q\mathbf{R}_{\mathsf{BP}^*} \\ \tau\beta^2 q\mathbf{R}_{\mathsf{BP}^*}^T & \beta^2q^2\mathbf{I}_{m\times m} \end{pmatrix} = \mathbf{T}_2\mathbf{T}_2^T.\end{aligned}$$

This fact may be seen quadrant-wise by our choice of $\tau = \gamma\alpha^2/\beta^2$ and the settings of $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$ and $\mathbf{Q}' = \mathbf{L}'\mathbf{L}'^T$ in Equations (3.2) and (3.3). It then follows that $(\mathbf{T}_2^{-1}\mathbf{T}_1)(\mathbf{T}_2^{-1}\mathbf{T}_1)^T = \mathbf{I}_{2m\times 2m}$, implying $\mathbf{T}_1 = \mathbf{T}_2\mathbf{Q}^*$ for some orthogonal matrix $\mathbf{Q}^*$. Because the spherical Gaussian $\mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}}$ is invariant under rigid transformations, we have $\mathbf{T}_1 \cdot \mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}} = \mathbf{T}_2\mathbf{Q}^* \cdot \mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}} = \mathbf{T}_2 \cdot \mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}}$, and the claim follows. □

Claim 3.2.12. *Hybrids* Hyb_4 *and* Hyb_5 *are statistically indistinguishable.*

Proof. Observe the main difference between hybrids Hyb_4 and Hyb_5 is the order of generation of vectors $\boldsymbol{y}$ and $\boldsymbol{r}_{\mathsf{BP}^*}$: In hybrid Hyb_4, we first sample $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1) \leftarrow \mathcal{D}_{\mathbb{Z}^{2m},\beta^2q^2\mathbf{I}_{2m\times 2m}}$ and set $\boldsymbol{r}_{\mathsf{BP}^*} \leftarrow \boldsymbol{y} + \mathcal{D}_{\mathbb{Z}^{2m}-\boldsymbol{y},q^2(s^2-\beta^2)\mathbf{I}_{2m\times 2m}}$, while in hybrid Hyb_5, we first sample $\boldsymbol{r}_{\mathsf{BP}^*} \leftarrow \mathcal{D}_{\mathbb{Z}^{2m},s^2q^2\mathbf{I}_{2m\times 2m}}$ and set $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1) \leftarrow \boldsymbol{r}_{\mathsf{BP}^*}/2 + \mathcal{D}_{\mathbb{Z}^{2m},(\beta^2-s^2/4)q^2\mathbf{I}_{2m\times 2m}}$. By setting parameters appropriately as in Section 3.2.3, these two distributions are statistically close. □

Claim 3.2.13. *Assuming the hardness of extended-*$\mathsf{LWE}^{+}_{n,m,q,D_{\mathbb{Z}^m,\beta'},\mathbf{R}}$ *for any adversarially chosen distribution over matrices* $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$*, then hybrids* Hyb_5 *and* Hyb_6 *are computationally indistinguishable.*

Proof. Suppose $\mathcal{A}$ has non-negligible advantage in distinguishing hybrid Hyb_5 and Hyb_6, then we use $\mathcal{A}$ to construct an extended-LWE^{+} algorithm $\mathcal{B}$ as follows:

Invocation. $\mathcal{B}$ invokes adversary $\mathcal{A}$ to commit to a challenge attribute vector $\boldsymbol{x}^* = (x_1^*, ..., x_\ell^*)$ and challenge branching program BP^*. Then $\mathcal{B}$ generates $\mathbf{R}_{\mathsf{BP}^*}$ by first sampling $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{R}^c\})$ as in the hybrid, and computes

$$\mathbf{R}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{Sim}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{R}^c\}, \mathbf{A})$$

Then it receives an extended-LWE^{+} instance for the matrix $\mathbf{R} = \mathbf{R}_{\mathsf{BP}^*}$ as follows:

$$\{\mathbf{A}, \boldsymbol{b} = \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}, z_0, z_1, \langle z_0, \boldsymbol{b} - \boldsymbol{e}\rangle + e, \langle z_1^T\mathbf{R}, \boldsymbol{b} - \boldsymbol{e}\rangle + e'\}$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^m$, $\boldsymbol{e}, z_0, z_1 \xleftarrow{\$} \chi^n$ and $e, e' \xleftarrow{\$} \chi$. Algorithm $\mathcal{B}$ aims to leverage adversary $\mathcal{A}$'s output to solve the extended-LWE^{+} assumption.

Setup. $\mathcal{B}$ generates matrices $(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c\})$ as specified in hybrid Hyb_1. Then, $\mathcal{B}$ sets challenge secret key $\mathsf{SK}_{\mathsf{BP}^*} = \boldsymbol{r}_{\mathsf{BP}^*} = (\boldsymbol{r}_0^*|\boldsymbol{r}_1^*) = (z_0|z_1)$ from extended-LWE^{+} instance and computes vector $\boldsymbol{u}$ as in hybrid Hyb_5.

Secret key queries. $\mathcal{B}$ answers adversary $\mathcal{A}$'s secret key queries as in hybrid Hyb_2.

Challenge ciphertext. $\mathcal{B}$ answers adversary $\mathcal{A}$'s P-sample query by setting

$$\psi_0^* = \boldsymbol{b}/q + \boldsymbol{e}_0^{(2)} + \nu\boldsymbol{y}_1^T\mathbf{R}_{\mathsf{BP}^*}/q, \quad \psi_i^* = \psi_0^{*T}\mathbf{R}_i/q, \quad \psi_{0,i}^* = \psi_0^{*T}\mathbf{R}_{0,i}/q, \quad \psi^{*c} = \psi_0^{*T}\mathbf{R}^c/q$$

and $c^* = \boldsymbol{r}_{\mathsf{BP}^*}^T[\mathbf{I}_{m\times m}|\mathbf{R}_{\mathsf{BP}^*}](\boldsymbol{b}/q - \boldsymbol{e}^{(1)}) + \mathcal{D}_{\mathbb{Z}^m,\alpha\mathbf{I}_{m\times m}}$.

Faking receiver coin query. $\mathcal{B}$ answers adversary $\mathcal{A}$'s faking receiver coin query by outputting the extended-LWE instance's vector $\mathsf{SK}_{\mathsf{BP}^*} = \boldsymbol{r}_{\mathsf{BP}^*}$.

Output. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

We can rewrite the expression of $c^{*'}$ to be

$$\begin{aligned} c^{*'} &= ([\mathbf{A}^*|\mathbf{A}^*\mathbf{R}_{\mathsf{BP}^*}]\binom{z_0}{z_1})^T \boldsymbol{s}/q + \mathcal{D}_{\mathbb{Z}_1,\alpha} \\ &= ((z_0|z_1)\binom{\mathbf{A}^{*T}}{\mathbf{R}_{\mathsf{BP}^*}^T\mathbf{A}^{*T}})\boldsymbol{s}/q + D_{\mathbb{Z}_1,\alpha} = z_0\mathbf{A}^{*T}\boldsymbol{s}/q + z_1\mathbf{R}_{\mathsf{BP}^*}^T\mathbf{A}^{*T}\boldsymbol{s}/q + \mathcal{D}_{\mathbb{Z}_1,\alpha} \\ &= \langle z_0, \boldsymbol{b}/q - \boldsymbol{e}^{(1)}\rangle + \langle z_1^T\mathbf{R}_{\mathsf{BP}^*}, \boldsymbol{b}/q - \boldsymbol{e}^{(1)}\rangle + \mathcal{D}_{\mathbb{Z}_1,\alpha} \end{aligned}$$

We can see that if the eLWE^+ instance's vector $\boldsymbol{b}$ is pseudorandom, then the distribution simulated by $\mathcal{B}$ is exactly the same as H_5. If $\boldsymbol{b}$ is truly random and independent, then the distribution simulated by $\mathcal{B}$ is exactly the same as H_6. Therefore, if $\mathcal{A}$ can distinguish H_5 from H_6 with non-negligible probability, then $\mathcal{B}$ can break the $\mathsf{eLWE}^+_{n,m,q,\mathcal{D}_{(\alpha/2)q},\alpha',\mathbf{S}_{f^*}}$ problem for some $\alpha' \geq 0$ with non-negligible probability. □

Claim 3.2.14. *Hybrids* Hyb_6 *and* Hyb_7 *are statistically indistinguishable.*

Proof. Recall the only difference between hybrids Hyb_6 and Hyb_7 is the generation of challenge ciphertext. In hybrid Hyb_7, we observe if ψ_0^* is chosen from uniform distribution, then by Leftover Hash Lemma 5.2.2, it holds

$$\psi_i^* = \psi_0^{*T}\mathbf{R}_i/q, \quad \psi_{0,i}^* = \psi_0^{*T}\mathbf{R}_{0,i}/q, \quad \psi^{*c} = \psi_0^{*T}\mathbf{R}^c/q$$

is also uniformly random (in their marginal distribution). Therefore, it remains to show that c^* is still uniformly random even conditioned on fixed samples of $(\psi_0^*, \{\psi_i^*\}_i, \{\psi_{0,i}^*\}_i, \{\psi^c\})$.

As calculated above, we can unfold the expression of c^* as

$$c^* = \langle z_0, \boldsymbol{b}/q - \boldsymbol{x}^{(1)}\rangle + \langle z_1^T\mathbf{R}_{\mathsf{BP}^*}, \boldsymbol{b}/q - \boldsymbol{x}^{(1)}\rangle + \mathcal{D}_{\mathbb{Z}_1,\alpha}$$

We note that $\boldsymbol{b}/q - \boldsymbol{x}^{(1)} = \psi_0^* - \boldsymbol{x}^{(1)} - \boldsymbol{x}^{(2)} - \nu \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{y}_1/q$, thus if we show that

$$\langle \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{z}_1, \nu \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{y}_1/q \rangle$$

is close to uniform distribution (modulo 1), then c^* will also be close to the uniform distribution (modulo 1), as c^* is masked by this uniformly random number. Recall in hybrids, we set $\boldsymbol{y}_1 = \boldsymbol{z}_1/2 +$ (shift), so it is sufficient to analyze

$$\langle \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{z}_1, \nu \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{y}_1/q \rangle = \nu \langle \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{z}_1, \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{z}_1/q \rangle = \nu \|\mathbf{R}^*_{\mathsf{BP}^*} \boldsymbol{z}_1\|^2/q$$

By applying Lemma 3.2.2 inductively on matrix $\mathbf{R}_{\mathsf{BP}^*}$, we can obtain that

$$\|\mathbf{R}^*_{\mathsf{BP}^*} \boldsymbol{z}_1\|^2/q \geq \frac{(\|\mathbf{R}_{0,j} \boldsymbol{z}_1\| - \Theta(m^{1.5})\ell \|\boldsymbol{z}_1\|)^2}{q}$$

where $\mathbf{R}_{0,j} \in \{-1, 1\}^{m \times m}$. Since vector $\boldsymbol{z}_1$ is sampled from Gaussian with width sq, so its two-norm is at least $\sqrt{m}(sq)$ with overwhelming probability. Then by Lemma 3.2.4, the distribution $\nu \|\mathbf{R}^*_{\mathsf{BP}^*} \boldsymbol{z}_1\|^2/q$ is a Gaussian distribution with width at least

$$d = \tau \frac{(\eta sqm - \Theta(m^2\ell)sq)^2}{q} = \frac{\gamma\alpha^2(\eta sqm - \Theta(m^2\ell)sq)^2}{\beta^2 q}$$

We recall again that ν was sampled from a Gaussian with parameter $\tau = \gamma\alpha^2/\beta^2$. By our setting of parameters, we have $d/\omega(\log(n)) \geq 1$. A Gaussian with such width is statistically close to uniform in the domain $\mathbb{Z}_1$. This completes the proof. □

This completes the proof of Lemma 3.2.7. Further, Theorem 3.2.5 follows from Lemmas 3.2.6 and 3.2.7. A (flexibly) bi-deniable ABE from LWE then follows. □

3.2.3 Parameter Setting

The parameters in Table 3.1 are selected in order to satisfy the following constraints:

Parameters	Description	Setting
n, m	lattice dimension	$n = \lambda, m = n^2 \log n$
ℓ	length of input to branching program	$\ell = n$
q	modulus (resp. bit-precision)	prime $\geq n^{1.5} m^{2.5} \omega(\log n)$
α	sampling error terms $\boldsymbol{e}, e$	$\frac{1}{n^{2.5} \log^3 n}$
β	sampling correlation vector $\boldsymbol{y}$	$\alpha/2$
γ	sampling correlation coefficient μ	$\frac{1}{n \log^{1.5} n}$
s	sampling secret key $\boldsymbol{r}$	$3\beta/2$
η	scaling parameter for $\mathbf{R}_{0,j}$	$\Theta(m\ell)$

Table 3.1: Parameter Description and Simple Example Setting

- To ensure correctness in Lemma 3.2.6, we have $\alpha s q m(\eta \sqrt{m} + 2m^{1.5}\ell) \leq 1/4$.
- To ensure deniability in Hybrid Hyb_7, we have $d/\omega(\log(n)) > \frac{\gamma\alpha^2(\eta sqm - \Theta(m^2 \ell sq))^2}{\beta^2 q \omega(\log(n))} > 1$.
- To ensure large enough LWE noise, we need $\alpha \geq (\sqrt{n} \log^{1+\delta} n)/q$.
- To apply the leftover hash lemma, we need $m \geq 2n \log(q)$.
- To ensure that the matrix $\mathbf{Q}$ in $\mathsf{FakeRCoins}$ is positive definite, we have $\beta \geq \alpha\gamma \sqrt{\eta \sqrt{m} + 2m^{1.5}\ell}$; To ensure that the matrix $\mathbf{Q}'$ in the security proof is positive definite, we have $\alpha \geq \tau\beta \sqrt{\eta \sqrt{m} + 2m^{1.5}\ell}$. This constraint will also imply that in the security proof, both $\mathbf{Q}'$ and $\mathbf{Q}' - \beta' \mathbf{I}_{m \times m}$ are positive definite (note $\beta' = \alpha/2$).
- To ensure hybrids Hyb_3 and Hyb_5 are well-defined, we have $s > \beta$ and $\beta > s/2$. Let $s := (3/2)\beta$.

Regev [112] showed that for $q > \sqrt{m}/\beta'$, an efficient algorithm for $\mathsf{LWE}_{n,m,q,\chi}$ for $\chi = \mathcal{D}_{\beta' q}$ (and $\beta' q \geq \sqrt{n}\omega(\log(n))$) implies an efficient quantum algorithm

for approximating the SIVP and GapSVP problems, to within $\tilde{O}(n/\beta')$ approximation factors in the worst case. Our example parameter setting yields a bi-deniable AB-BTS based on the (quantum) hardness of solving $\mathsf{SIVP}_{\widetilde{O}(n^{9.5})}$, respectively $\mathsf{GapSVP}_{\widetilde{O}(n^{9.5})}$. (We write this term to additionally absorb the $(1/q^2)$ loss from our LWE to eLWE^+ reduction.) We leave further optimizing the lattice problem approximation factor to future work.

3.2.4 From AB-BTS to Flexible Bi-Deniable ABE

We present the instantiation of a flexible bi-deniable ABE using our AB-BTS scheme described above. We let $\Sigma' = (\mathsf{Setup}', \mathsf{DenSetup}', \mathsf{Keygen}', \mathsf{SampleP}', \mathsf{SampleU}', \mathsf{TestP}', \mathsf{FakeRCoins}', \mathsf{FakeSCoins}')$ be an AB-BTS scheme. Then the flexible bi-deniable ABE $\Sigma = (\mathsf{Setup}, \mathsf{DenSetup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{DenEnc}, \mathsf{Dec}, \mathsf{SendFake}, \mathsf{RecFake})$ is:

- $\mathsf{Setup}(1^\lambda)$: Run algorithm $(\mathsf{PP}', \mathsf{MSK}') \leftarrow \mathsf{Setup}'(1^\lambda)$ in AB-BTS and set $\mathsf{PP} = \mathsf{PP}', \mathsf{MSK} = \mathsf{MSK}'$.
- $\mathsf{DenSetup}(1^\lambda)$: Run algorithm $(\mathsf{PP}', \mathsf{MSK}', \mathsf{fk}') \leftarrow \mathsf{DenSetup}'(1^\lambda)$ in AB-BTS and set $\mathsf{PP} = \mathsf{PP}', \mathsf{MSK} = \mathsf{MSK}', \mathsf{fk} = (\mathsf{fk}', \mathsf{MSK}')$.
- $\mathsf{Keygen}(\mathsf{MSK}, f)$: Run algorithm $\mathsf{SK}'_f \leftarrow \mathsf{Keygen}'(\mathsf{MSK}, f)$ in AB-BTS and set $\mathsf{SK}_f = \mathsf{SK}'_f$.
- $\mathsf{Enc}(\mathsf{PP}, \boldsymbol{x}, \mu; (r_S^{(1)}, r_S^{(2)}))$: On input the message $\mu \in \{0,1\}$, if $\mu = 0$, then run $c_i \leftarrow \mathsf{SampleU}'(\mathsf{PP}, \boldsymbol{x}; r_S^{(i)})$ for $i = 1, 2$, otherwise, $\mu = 1$, run $c_1 \leftarrow \mathsf{SampleU}'(\mathsf{PP}, \boldsymbol{x}; r_S^{(1)})$ and $c_2 \leftarrow \mathsf{SampleP}'(\mathsf{PP}, \boldsymbol{x}; r_S^{(2)})$. Output $\mathsf{CT}_{\boldsymbol{x}} = (c_1, c_2)$.
- $\mathsf{DenEnc}(\mathsf{PP}, \boldsymbol{x}, \mu; (r_S^{(1)}, r_S^{(2)}))$: On input the message $\mu \in \{0,1\}$, then run $c_i \leftarrow$

$\mathsf{SampleP}'(\mathsf{PP}, \boldsymbol{x}; r_S^{(i)})$ for $i = 1, 2$, otherwise, $\mu = 1$, run $c_1 \leftarrow \mathsf{SampleU}'(\mathsf{PP}, \boldsymbol{x}; r_S^{(1)})$ and $c_2 \leftarrow \mathsf{SampleP}'(\mathsf{PP}, \boldsymbol{x}; r_S^{(2)})$. Output $\mathsf{CT}_{\boldsymbol{x}} = (c_1, c_2)$.

- $\mathsf{Dec}(\mathsf{CT}_{\boldsymbol{x}}, \mathsf{SK}_f)$: If $f(\boldsymbol{x}) \neq 0$, then output $\perp$. Otherwise, parse $\mathsf{CT}_{\boldsymbol{x}} = (c_1, c_2)$ and run $b_i \leftarrow \mathsf{TestP}'(\mathsf{SK}_f, c_i)$ for $i = 1, 2$. Output 0 if the $b_1 = b_2$ and 1 if $b_1 \neq b_2$.
- $\mathsf{SendFake}(\mathsf{PP}, r_S, \mu, \mu')$: If $\mu = \mu'$, return r_S. If $(\mu, \mu') = (0, 1)$, then run $r_S^{*(2)} \leftarrow \mathsf{FakeSCoins}'(\mathsf{PP}, r_S^{(2)})$ and return $(r_S^{(1)}, r_S^{*(2)})$. Else if $(\mu, \mu') = (1, 0)$, run $r_S^{*(1)} \leftarrow \mathsf{FakeSCoins}'(\mathsf{PP}, r_S^{(1)})$ and return $(r_S^{*(1)}, r_S^{(2)})$.
- $\mathsf{RecFake}(\mathsf{PP}, \mathsf{fk}, \mathsf{CT}_{\boldsymbol{x}}, f, \mu')$: Parse $\mathsf{CT}_{\boldsymbol{x}} = (c_1, c_2)$ and use fk to decrypt the ciphertext $\mathsf{CT}_{\boldsymbol{x}}$ then obtain the plaintext μ. If $\mu = \mu'$, then run the honest key generation of the BTS scheme, i.e., $\mathsf{SK}'_f \leftarrow \mathsf{Keygen}'(\mathsf{MSK}', f)$. Otherwise, run $\mathsf{SK}'_f \leftarrow \mathsf{FakeRCoins}'(\mathsf{PP}, \mathsf{fk}, c_{\mu+1}, f)$. Return SK'_f.

Similar to the work by Canetti et al. [43] and O'Neil et al. [105], the following, desired theorem can be proven in a straightforward manner.

Theorem 3.2.15. *Assume that Σ' is a flexible bi-deniable AB-BTS, as in Definition 3.1.2. Then Σ is a flexibly bi-deniable ABE, as in Definition 3.1.1.*

CHAPTER 4

ATTRIBUTE-BASED ENCRYPTION FOR RAMS

4.1 Technical Overview

We give an overview of the techniques employed in our main construction. We later reuse some of the techniques used in our main construction to obtain a construction in the dual setting as well.

Starting Point: Garbled RAMs. A natural idea to build ABE for RAMs is to use garbled RAMs [74, 71, 72]. A garbled RAM allows for separately encoding a RAM program[1]-database pair (P, D) and encoding an input x such that the encodings only leak the output $P^D(x)$; compute both the encodings requires a private key not revealed to the adversary. Notice that a garbled RAM scheme implies a *one-time, secret key* ABE for RAM scheme; meaning that the adversary only gets to make a single ciphertext query and a single attribute key query. Indeed, it is unclear how to remove the one-time restriction while simultaneously achieve a public-key ABE for RAMs scheme by generically using garbled RAMs. Hence, we circumvent this conundrum by diving into the innards of the existing garbled RAMs schemes. The hope would be to adopt some of the techniques used in constructing garbled RAMs to build an ABE for RAMs scheme.

Most of the current known constructions of garbled RAMs have the following blueprint: to garble a RAM program P (associated with a step circuit C), database D, generate T garbled circuits, where T is an upper bound on the running time of P. The i^{th} garbled circuit performs the execution of the i^{th} time step

[1]The formal definition of a RAM program can be found in the preliminaries.

of P. Also, every entry of the database D is suitably encoded using an appropriate encoding scheme (for instance, in [74], an IBE (identity-based encryption) key is associated with every entry of the database). The garbling of P consists of all the T garbled circuits and the encoding of the database D. The encoding of input x consists of wire labels of the first garbled circuit corresponding to the input x.

To evaluate a garbling of P on an encoded database D and wire labels of x, perform the following operations for $i = 1, \ldots, T-1$:

- If $i = 1$, evaluate the first garbled circuit on wire labels of x.
- If $i > 1$, evaluate the i^{th} garbled circuit to obtain output encodings of the i^{th} step of execution of P^D on x.
- Next, we compute the *recoding step* that converts the output encodings of the i^{th} step into the input encodings of the $(i+1)^{th}$ step. These input encodings will be fed to the $(i+1)^{th}$ garbled circuit.

The output of the T^{th} garbled circuit determines the output of execution of $P^D(x)$.

From Garbled RAMs to ABE for RAMs: Challenges Toward realizing our hope of using garbled RAMs techniques to build an ABE for RAMs scheme, we encounter the following fundamental issues:

- The garbling and encoding operations in a garbled RAM scheme are inherently secret-key operations; they require a shared secret-key to compute garbled program and database encodings respectively. Since our goal is

to construct public-key ABE for RAMs, the encryptor can perform neither the garbling nor the encoding procedures.

- Garbling schemes typically do not offer any reusability property[2]; they are useful only when a single computation needs to be hidden. It is unclear how to use garbled circuits, an integral part of current garbled RAM constructions, in the ABE setting, where multiple attribute keys need to be issued.
- Tied to the issue of using garbled circuits is also the issue of implementing the recoding step. We need to implement a recoding step that can be reused across different computations.

Our Solution in a Nutshell. The main technical contribution of this paper is to identify a template to solve this problem and instantiate this template using a novel combination of existing lattice-based techniques.

We describe our template of ABE for RAMs. This will be an oversimplifcation of our actual scheme and is intended to help the reader toward understanding our final construction. For now, focus on the setting when the keys are only associated with read-only RAMs (i.e., they only read from the memory and never write back to the memory). This template can be easily adapted to the setting where the program can also write to the memory.

- A key for a program P and a database D will consist of two parts: the first part, denoted by sk_D, is associated with the database and the second part,

[2]An exception is the reusable garbling scheme of Goldwasser et al. [78], however their scheme only offers one-sided reusability: that is, their scheme only allows the adversary to get a single garbled circuit which can be reused across multiple input encodings. This is not useful in our setting since the adversary gets to query multiple keys. Moreover, just like any garbling scheme, even reusable garbled circuits require secret-key to perform the encoding operations.

denoted by $(\mathsf{StepKey}_1, \ldots, \mathsf{StepKey}_T)$, consists of T sets of **recoding** keys with T being the maximum running time of $P^D(\cdot)$.

- A ciphertext for an input x and a secret message μ consists of two parts $(\mathsf{CT}_x^{(1)}, \mathsf{CT}_x^{(2)})$ and an encryption of μ, namely CT_μ (we will soon see that we will not be able to use any encryption scheme but rather a scheme that satisfies some specific properties): the first part $\mathsf{CT}_x^{(1)}$ serves as encoding of the initial input to the step circuit of the RAM program. We describe the role of the second part $\mathsf{CT}_x^{(2)}$ when we describe the decryption operation below.

- The decryption of a ciphertext of (x, μ) using a key of (P, D) proceeds in the following steps:

  - **Translation Step**: First, using the second part of the ciphertext, i.e., using $\mathsf{CT}_x^{(2)}$, and using the key associated with the database sk_D in the attribute key, obtain a probabilistic encoding of D.

  - The following operations are executed for time steps $t = 1, \ldots, T$:

    * **Evaluation Step**: Homomorphically evaluate on the input encodings of the t^{th} step to obtain the output encodings of the t^{th} step. This is akin to the evaluation of the t^{th} garbled circuit in the garbled RAM constructions.

    * **Recoding Step**: Recode the output encodings of the t^{th} step to obtain the input encodings of the $(t+1)^{th}$ step. This is akin to the recoding step of the garbled RAM constructions.

    The t^{th} evaluation and the recoding steps are performed using the key $\mathsf{StepKey}_t$. Moreover, they interact with the probabilistic encoding of D produced in the translation step.

If the output of the final T^{th} step is an encoding of 0 then this is used to decrypt the encryption of μ, given as part of the ciphertext, to obtain the result μ.

We now show how to implement the above template using lattice-based techniques. The starting point to our construction is the work of [31].

Implementation of Our Template: Read-only RAMs. We implement our template using lattice-based techniques; as before, we first consider the read-only setting. We first start with the high level description of the encryption procedure: let $(\mathsf{CT}_x^{(1)}, \mathsf{CT}_x^{(2)}, \mathsf{CT}_\mu)$ be the ciphertext associated with the input x and secret message μ. The first part $\mathsf{CT}_x^{(1)}$ consists of lattice-based encodings of x, initial state, initial read address and the initial read value of the RAM program. A lattice-based encoding of a bit b is computed using $\mathbf{s} \cdot (\mathbf{A} + b \cdot \mathbf{G}) + e$, where $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{1\times n}, \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and $e \in \mathbb{Z}_q^{1\times m}$ is drawn from a suitable error distribution; such lattice-based encodings has been studied by many works in the past [80, 31]. We generate CT_μ to be $\langle \mathbf{s}, \mathbf{u} \rangle + \mu\lceil q/2 \rceil + e^*$, where $\mathbf{u}, \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{1\times n}$ and $e^* \in \mathbb{Z}_q$ is drawn from a suitable error distribution.

We will postpone the discussion on the generation of $\mathsf{CT}_x^{(2)}$ and the attribute keys. Instead, we first mention the main ideas incorporated in the translation, evaluation and the recoding steps; this will then guide us toward identifying the attribute keys and also $\mathsf{CT}_x^{(2)}$ that will let us execute these steps.

- **Implementing the translation step**: The goal of this step is to obtain a lattice-based encoding of the database D; in particular, this encoding should be computed with respect to the same secret $\mathbf{s}$ used in the ci-

phertext. To do this, we generate $\mathsf{CT}_x^{(2)}$ and sk_D (belonging to the attribute key) such that evaluating sk_D on $\mathsf{CT}^{(2)}$ yields encodings of the form $(\{\mathbf{s}\cdot(\mathbf{A}_i^* + D[i]\mathbf{G})e_i\})$ (see footnote [3]). In more detail, $\mathsf{CT}_x^{(2)}$ contains auxiliary encodings of many $0/1$ matrices such that given any matrix, using these auxiliary encodings, we can compute an encoding of this specific matrix. That is, $\mathsf{CT}_x^{(2)}$ will consist of encodings of the form $\mathbf{s}\cdot(\mathbf{B}_{jk\ell} + 2^\ell \mathbf{M}_{jk}) + \boldsymbol{e}_{jk\ell}$ and $\mathbf{s}\cdot\mathbf{B}'_{jk\ell} + \boldsymbol{e}'_{jk\ell}$, where $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{1\times n}, \mathbf{B}_{jk\ell} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ for $j \in [n], k \in [m], \ell \in [\log(q)]$ and $(\boldsymbol{e}_{jk\ell}, \boldsymbol{e}'_{jk\ell}) \in \mathbb{Z}_q^{1\times m}$ is drawn from a suitable error distribution. Here, $\mathbf{M}_{jk}$ is a matrix with 1 in the $(j,k)^{th}$ entry and zeroes everywhere else. Now, observe that using the additive homomorphic properties, we can compute an encoding that is approximately $\mathbf{s}\cdot\left(\underbrace{\sum_{jk\ell} a_{jk\ell}\mathbf{B}_{jk\ell} + (1-a_{jk\ell})\mathbf{B}'_{jk\ell} + \mathbf{A}'_i)}_{\mathbf{A}_i^*} + D[i]\cdot\mathbf{G}\right)$, where $a_{jk\ell}$ denotes the ℓ^{th} bit in the bit decomposition of the $(j,k)^{th}$ entry in the matrix $\mathbf{A}'_i + D[i]\cdot\mathbf{G}$, with $\mathbf{A}'_i + D[i]\cdot\mathbf{G}$ being part of sk_D. Finally, we note that $\mathsf{CT}_x^{(2)}$ is independent of the size of the database D; this is necessary since we require that the ciphertext should be of size independent of the database length. We note that this technique of transforming encodings of bit decomposition of matrices into encodings of matrices have been studied in the past albeit for different reasons (see [38] for example).

An astute reader would notice that the translation step takes time proportional to the database size and thus, would violate the sub-linear decryption property! We avoid this problem by *only* translating only those database entries that are going to read during the evaluation of $P^D(x)$; note that P, D and x are public and hence, the entries that are going to be read can be correctly identified.

[3] $\mathbf{A}_i^* + D[i]\mathbf{G}$ will be denoted by $\mathbf{E_i}$ in the technical sections.

- **Implementing the evaluation step**: This step would be a direct adaptation of the lattice-based evaluation procedure of [31]. Given approximate encodings $(\{\mathbf{s} \cdot (\mathbf{A}_i + b_i\mathbf{G})\})$, for bits $b_1, \ldots, b_n$, and for any circuit C with a single-bit output, the evaluation procedure of [31] (we will use the notation later for this procedure as CtEval) allows for obtaining $(\{\mathbf{s} \cdot (\mathbf{A}_C + C(b_1, \ldots, b_n)\mathbf{G})e_i\})$. The matrix $\mathbf{A}_C$ is obtained by homomorphically evaluating the matrices $(\mathbf{A}_1, \ldots, \mathbf{A}_n)$ using the circuit C (later, we will refer to this procedure as $\mathsf{PubEval}$). We use the procedure of [31] to homomorphically evaluate the step circuit.

- **Implementing the recoding step**: We use lattice trapdoors [75] to convert output encodings of one time step into input encodings of the next time step. To give a flavor of how the lattice trapdoors are generated, we will take a simple case: suppose we need to translate an encoding of the read address $i \in [N]$ output by the τ^{th} evaluation step, we first sample a matrix $\mathbf{A}^{\mathsf{val},\tau}$ and then generate $\mathbf{T}_i^{\mathsf{rd},\tau}$ such that the following holds:

$$\left[\mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G} \| \mathbf{A}_i^* + D[i] \cdot \mathbf{G}\right] \begin{pmatrix} \mathbf{T}_i^{\mathsf{rd},\tau} \\ \mathbf{I} \end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + D[i] \cdot \mathbf{G} \tag{4.1}$$

  where $\mathbf{A}^{\mathsf{rd},\tau}$ is the matrix computed during the τ^{th} evaluation step. Recall that $\mathbf{A}_i^* + D_i \cdot \mathbf{G}$ is output by Moreover, $\mathbf{A}^{\mathsf{val},\tau} + b\mathbf{G}$ will serve as the matrix that is used to encode the read value for the $(\tau+1)^{th}$ step. (Later we will see that in order to make the security proof work, we also additionally need an anchor matrix $\mathbf{A}$ and this will be taken into account when we generate the trapdoor matrices; see the technical sections for more details). All the lattice trapdoors generated during the t^{th} step will be part of $\mathsf{StepKey}_t$.

Implementation of Our Template: Handling Write Operations. To handle RAM programs that also write to the memory, we do the following: first, we view the database as an append-only data structure, with initial size to be N. That is, every time the program wishes to write to some memory location i, it instead appends this value to the end of the database, say at the $(N+\tau)^{th}$ location. However, this procedure introduces an additional issue. Before we describe the issue, we point out that the current lattice-based techniques disallow us from rewriting to the same location twice[4] and thus, our only other option is to use the append-only data structure.

Suppose the i^{th} location is written during the τ^{th} step. This means that the $(N + \tau)^{th}$ location would now encode the latest value corresponding to the i^{th} location. If at a later point in time, i.e., in time step $\gg \tau$, the i^{th} location needs to be read, there is no mechanism in place that prevents an adversarial evaluator to use the old encoding of the i^{th} memory location to perform an illegal evaluation.

To solve this problem, we introduce an auxiliary circuit C^{up} which keeps track of all the writes done so far and thus, for any given location i, can correctly identify the latest encoding to be used. In particular, the evaluation step from the read-only setting needs to be revised to also take into account the circuit C^{up}. That is, first the step circuit is homomorphically evaluated to obtain the location i to be read next and then the circuit C^{up} is executed to correctly identify the $(N + \tau)^{th}$ encoding that contains the value associated with location i, where τ is the time step where the i^{th} memory location was last written to. The translation and the recoding steps will be defined along the same lines as that of the read-only setting; we defer the details to the technical sections.

[4]This would tantamount to obtaining two approximate encodings of the form $\mathbf{s}(\mathbf{A}_i + b_i \cdot \mathbf{G})$ and $\mathbf{s}(\mathbf{A}_i + b'_i \cdot \mathbf{G})$, where b_i is the old value and b'_i is the newly written value; assuming $b'_i \neq b_i$, having these two encodings is sufficient to break LWE.

Careful readers may notice that the run-time of circuit C^{up} is $O(T)$, which implies the decryption time depends quadratically on T. However, we can resolve this issue by first compiling a RAM into a last-write-aware RAM. Given a RAM P, we can compile it into another machine denoted RAM P' where the next-instruction circuit is replaced with a "next-instruction RAM" that not only emits the next address to access, but also when the next address was last written. We show such a compilation algorithm that incurs only logarithmic overhead. The idea is to maintain a balanced search tree (e.g., a 2-3 tree) that records for each logical address, when the last write was. Moreover, in this balanced search tree, each parent also keeps track of the last written times of its children. Now, when the next-instruction circuit of RAM P decides to access some logical address addr, P' would search for addr in this search tree to find out when addr was last written. Note that every search-tree operation touches constant number of tree-paths, and since the parent knows the last-written times of the children, during the search-tree operation, every memory access always knows its last-written time.

The construction for the dual setting, where the database is part of the ciphertext as against the attribute key, is obtained by a simple modification of the above template. In particular, the translation step is not necessary for the dual setting and hence, will be removed. The other steps, evaluation and recoding steps, will be defined along the same lines as the above template.

4.2 ABE for RAMs: Definitions

We state the syntax and security definition of (key-policy) public-key attribute-based encryption (ABE) for RAMs. It consists of a tuple of ppt algorithms $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with details as follows:

- **Setup,** $\mathsf{Setup}(1^\lambda, 1^T)$: On input security parameter λ and upper time bound T, setup algorithm outputs public parameters PP and master secret key MSK.
- **Key Generation,** $\mathsf{KeyGen}(\mathsf{MSK}, P, D)$: On input a master secret key MSK, a RAM program P and database D, it outputs a secret key $\mathsf{SK}_{P,D}$.
- **Encryption,** $\mathsf{Enc}(\mathsf{PP}, x, \mu)$: On input public parameters PP, an input x and a message μ, it outputs a ciphertext CT_x.
- **Decryption,** $\mathsf{Dec}(\mathsf{SK}_{P,D}, \mathsf{CT}_x)$: This is modeled as a RAM program. In particular, this algorithm will have random access to the binary representations of the key $\mathsf{SK}_{P,D}$ and the ciphertext CT_x. It outputs the corresponding plaintext μ if $P^D(x) = 0$; otherwise, it outputs $\perp$.

We define the correctness, efficiency and security properties below.

Correctness.

Definition 4.2.1 (Correctness)**.** *We say that the ABE for RAMs scheme described above is correct, if for any message μ, any RAM program P, any database D and any input x where $P^D(x) = 0$, we have* $\mathsf{Dec}(\mathsf{SK}_{P,D}, \mathsf{CT}_x) = \mu$, *where* $(\mathsf{MSK}, \mathsf{PP}) \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$, $\mathsf{SK}_{P,D} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, P, D)$ *and* $\mathsf{CT}_x \leftarrow \mathsf{Enc}(\mathsf{PP}, x, \mu)$.

Efficiency. We define two efficiency properties associated with a ABE for RAMs scheme: namely sub-linear decryption and input-specific runtime property. The latter property implies the former.

SUB-LINEAR DECRYPTION: This property states that the complexity of decryption is $p(\lambda, T)$ for some fixed polynomial p, where T is the maximum runtime bound specified as part of the setup. We call this sub-linear decryption for the following reason: suppose T is *sufficiently* sublinear in $|D|$ (for instance, poly-logarithmic in $|D|$) then the decryption time is sub-linear in $|D|$. More specifically, suppose $p(\lambda, T) = \lambda^{c'} \cdot T^c$ and if $T << |D|^{\frac{1}{c}}$, for some constants $c', c \in \mathbb{N}$, then the decryption complexity is sub-linear in $|D|$.

Definition 4.2.2 (Sublinear Decryption)**.** *An ABE for RAMs scheme* ABE *is said to satisfy sublinear decryption property if the following holds: for any database* D*, message* μ*, program* P*, input* x*, (i)* $(\mathsf{MSK}, \mathsf{PP}) \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$*, (ii)* $\mathsf{SK}_{P,D} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, P, D)$*, (iii)* $\mathsf{CT}_x \leftarrow \mathsf{Enc}(\mathsf{PP}, x, \mu)$ *and, (iv) the decryption* Dec *of the functional key* $\mathsf{SK}_{P,D}$ *on input the ciphertext* CT_x *takes time poly*(T, λ)*, where* T *is the running time of* $P^D(x)$*.*

INPUT-SPECIFIC RUNTIME: This property states that the time to decrypt a ciphertext CT of (D, μ) using an attribute key of SK_P is $p(\lambda, t)$ for some fixed polynomial p, where t is the execution time of P on input database D. Note that t could be much smaller than T, where T is the maximum bound on the running time of the P.

Definition 4.2.3 (Input-specific Runtime)**.** *An ABE for RAMs scheme* ABE *is said to satisfy input-specific runtime property if the following holds: for any database* D*, message* μ*, program* P*, input* x*, (i)* $(\mathsf{MSK}, \mathsf{PP}) \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$*, (ii)* $\mathsf{SK}_{P,D} \leftarrow$

$\mathsf{KeyGen}(\mathsf{MSK}, P, D)$, *(iii)* $\mathsf{CT}_x \leftarrow \mathsf{Enc}(\mathsf{PP}, x)$ *and, (iv) the decryption* Dec *of the functional key* $\mathsf{SK}_{P,D}$ *on input the ciphertext* CT_x *takes time poly*(t, λ)*, where* t *is the running time of* $P^D(x)$.

Remark 4.2.4. *While the above properties focus on the decryption complexity, we can also correspondingly define efficiency measures for setup, key generation and encryption. Since the focus of this work is on decryption complexity, we postpone the discussion of these properties to future works.*

Security. Our definition of security for ABE for RAMs will be simulation-based and in the selective setting; along the same lines as that of ABE for circuits. Informally speaking, the adversary is allowed to make multiple RAM program and database queries and submit an input query x^* such that for ever program/database (P, D) queried, we have $P^D(x^*) \neq 0$. The adversary is also allowed to submit the challenge message μ. We require that the adversary cannot distinguish the two worlds: (i) when the attribute keys and ciphertext are computed as per the scheme, (ii) when the attribute keys and ciphertext can be simulated even without given μ.

Definition 4.2.5. *An ABE scheme* Π *for RAMs is simulation-based selectively secure if there exists* ppt *simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ *such that for any* ppt *admissible adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the two distributions* $\{\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A}}(1^\lambda)\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \{\mathsf{Expt}^{\mathsf{ideal}}_{\mathcal{S}}(1^\lambda)\}_{\lambda\in\mathbb{N}}$ *are computationally indistinguishable*

1. $x^* \leftarrow \mathcal{A}_1(1^\lambda)$
2. $(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$
3. $\mu \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{MSK},\cdot,\cdot)}(\mathsf{PP})$
4. $\mathsf{CT}_{x^*} \leftarrow \mathsf{Enc}(\mathsf{PP}, x^*, \mu)$
5. $\alpha \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{MSK},\cdot,\cdot)}(\mathsf{PP}, \mathsf{CT}_{x^*})$
6. Output $(\mathsf{PP}, \mu, \alpha)$

(a) $\mathsf{Expt}^{\mathsf{real}}_{\mathcal{A}}(1^\lambda)$

1. $x^* \leftarrow \mathcal{A}_1(1^\lambda)$
2. $\mathsf{PP} \leftarrow \mathcal{S}_1(1^\lambda, 1^T, x^*)$
3. $\mu \leftarrow \mathcal{A}_2^{\mathcal{S}_3(x^*,\cdot,\cdot)}(\mathsf{PP})$
4. $\mathsf{CT}_{x^*} \leftarrow \mathcal{S}_2(\mathsf{PP}, x^*, 1^{|\mu|})$
5. $\alpha \leftarrow \mathcal{A}_2^{\mathcal{S}_3(x^*,\cdot,\cdot)}(\mathsf{PP}, \mathsf{CT}_{x^*})$
6. Output $(\mathsf{PP}, \mu, \alpha)$

(b) $\mathsf{Expt}^{\mathsf{ideal}}_{\mathcal{S}}(1^\lambda)$

We call adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ admissible*, if the query* (P_i, D_i) *made by* $\mathcal{A}_2$ *satisfies* $P_i^{D_i}(x^*) \neq 0$. In the ideal experiment $\mathsf{Expt}^{\mathsf{ideal}}_{\mathcal{S}}(1^\lambda)$: $\mathcal{S}_1$ is used to generate simulated public paramters, $\mathcal{S}_2$ generates challenge ciphertext, and $\mathcal{S}_3$ answers secret key queries.

Dual Setting. We also consider the dual setting of the syntax described above, where the database is associated with ciphertext. We term this notion as *dual ABE for RAMs*. As in the above definition, the dual scheme consists of algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. The algorithms Setup and Dec are defined the same way as above. We define KeyGen and Enc as follows.

- $\mathsf{KeyGen}(\mathsf{MSK}, P)$: On input a master secret key MSK, a RAM program P, it outputs a secret key SK_P.
- $\mathsf{Enc}(\mathsf{PP}, D, x, \mu)$: On input public parameters PP, a database D, an input x and a message μ, it outputs a ciphertext $\mathsf{CT}_{D,x}$.

We omit the descriptions of the correctness, efficiency and security properties of dual ABE for RAMs as they are defined analogously.

4.3 ABE for RAMs: Read-Only Case

In this part, we describe our ABE construction for read-only RAMs. A RAM program P, with random access to database D and input x, is said to be read-only if it only reads from D and never writes to it. The step circuit for read-only RAM will be defined as follows:

$$(\mathsf{st}^\tau, \mathsf{rd}^\tau) \leftarrow C(\mathsf{st}^{\tau-1}, \mathsf{rd}^{\tau-1}, b^\tau)$$

where st^τ denotes the state information at τ-th step, rd^τ denotes the read address at τ-th step and b^τ is the read value.

Parameters of the Scheme. In the description below, the parameters we use are specified in Table 4.1.

Parameters	Description	Setting
N	maximum database length	$\mathsf{poly}(\lambda)$
T	maximum running time	$\mathsf{poly}(\lambda)$
L_{st}	state bit-length	$\mathsf{poly}(\lambda)$
L_{rd}	address bit-length	$\log N$

Table 4.1: Read-only ABE Parameters

We use notation $\{\mathsf{rd}_i^\tau\}_{i\in[L_{\mathsf{rd}}]}$ to denote the bit representation of read address $\mathsf{rd}^\tau \in [N]$.

4.3.1 Subroutines TranslatePK, StepEvalPK and StepEvalCT

Before proceeding to our ABE construction, we first describe the syntax of three following subroutines that are used in the construction:

- $\mathsf{ListMxDB} \leftarrow \mathsf{TranslatePK}\,(\mathsf{MxPK}_{\mathsf{aux}},\ D)$: On input auxiliary encoding public key $\mathsf{MxPK}_{\mathsf{aux}}$ and database $D = \{D_i\}_{i\in[N]}$, the translation algorithm outputs encoding matrices $\mathsf{ListMxDB}$ for the database.
- $(\mathsf{StepKey}_\tau, \mathsf{ListMxPK}_\tau) \leftarrow \mathsf{StepEvalPK}\,(C, \tau, \mathsf{ListMxPK}_{\tau-1}, \mathsf{MSK}, D)$: On input the step circuit C, step index τ, matrices $\mathsf{ListMxPK}_{\tau-1}$ for the $(\tau-1)$-th step and master secret key MSK, the key evaluation outputs the τ-th step key $\mathsf{StepKey}_\tau$ and encoding matrices $\mathsf{ListMxPK}_\tau$ for the τ-th step.
- $\mathsf{ListVecCT}_\tau \leftarrow \mathsf{StepEvalCT}\,(C, \tau, \mathsf{ListVecCT}_{\tau-1},\ \mathsf{StepKey}_\tau, D)$: On input the step circuit C, step index τ, ciphertext $\mathsf{ListVecCT}_{\tau-1}$ of the $(\tau-1)$-th step, τ-th attribute key and databae D, the ciphertext evaluation outputs the ciphertext $\mathsf{ListVecCT}_\tau$ of the τ-th step.

In the following description, we set function $f : \{0,1\}^{L_{\mathsf{st}}} \to \mathbb{Z}$ to be $f\left(\{x_i\}_{i=1}^{L_{\mathsf{st}}}\right) = \sum x_i \cdot 2^i$. The construction of $\mathsf{StepEvalPK}$ and $\mathsf{StepEvalCT}$ with respect to step circuit C are as follows:

$\underline{\mathsf{TranslatePK}\,(\mathsf{MxPK}_{\mathsf{aux}}, D)}$: the translation algorithm does the following:

- Parse $\mathsf{MxPK}_{\mathsf{aux}}$ as $\left\{\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right\}_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}$.
- Sample N random matrices $\{\mathbf{A}'_i\}_{i\in[N]}$ from uniform distribution over $\mathbb{Z}_q^{n\times m}$.
- For $i \in [N]$, set $\mathbf{A}_i = \mathbf{A}'_i + D[i]\mathbf{G}$.
- For $i \in [N]$, compute the encoding of i-th entry $\mathbf{E}_i$ as

$$\mathbf{E}_i = \sum_{j,k,\ell}\left(a_{jk\ell}\left(\mathbf{B}_{jk\ell} + 2^\ell \mathbf{M}_{j,k}\right) + \bar{a}_{jk\ell}\mathbf{B}'_{jk\ell}\right) = \mathbf{A}_i + \sum_{j,k,\ell}\left(a_{jk\ell}\mathbf{B}_{jk\ell} + \bar{a}_{jk\ell}\mathbf{B}'_{jk\ell}\right)$$

  where $\mathbf{M}_{j,k} \in \{0,1\}^{n\times m}$ is matrix with 1 on the (j,k)-th element and 0 elsewhere, $d_{jk\ell}$ is ℓ-th bit of the bit-decomposition of (j,k)-th element a_{jk} in

matrix $\mathbf{A}_i$, and $\bar{a}_{jk\ell}$ is its complement. For ease of notation, we set $\mathbf{B}_i = \sum_{j,k,\ell} \left(a_{jk\ell} \mathbf{B}_{jk\ell} + \bar{a}_{jk\ell} \mathbf{B}'_{jk\ell} \right)$.

Output matrices $\mathsf{ListMxDB} = \left\{ \left(\mathbf{A}'_i, \mathbf{B}_i, \mathbf{E}_i \right) \right\}_{i \in [N]}$.

$\underline{\mathsf{StepEvalPK}\,(C, \tau, \mathsf{ListMxPK}_{\tau-1}, \mathsf{MSK} = \mathbf{T}_{\mathbf{A}}, D)}$: the key evaluation algorithm does the following:

- Parse the encoding matrices $\mathsf{ListMxPK}_{\tau-1}$ as

$$\left(\mathbf{A}, \mathsf{ListMxPK}, \left\{ \mathbf{A}_i^{\mathsf{st},\tau-1} \right\}_{i \in [L_{\mathsf{st}}]}, \left\{ \mathbf{A}_i^{\mathsf{rd},\tau-1} \right\}_{i \in [L_{\mathsf{rd}}]}, \mathbf{A}^{\mathsf{val},\tau-1} \right)$$

- Compute $\left(\left\{ \mathbf{A}_i^{\mathsf{st},\tau} \right\}_{i \in [L_{\mathsf{st}}]}, \left\{ \mathbf{A}_i^{\mathsf{rd},\tau} \right\}_{i \in [L_{\mathsf{rd}}]} \right) = \mathsf{PubEval}\,(\mathsf{ListMxPK}_{\tau-1}, C)$, where algorithm $\mathsf{PubEval}$ is defined in Theorem 2.3.11.

- Sample $\mathbf{A}^{\mathsf{val},\tau} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$. For $i \in [N]$, compute $\mathbf{T}_i^{\mathsf{rd},\tau}$ as

$$\mathbf{T}_i^{\mathsf{rd},\tau} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G}, \mathbf{A}^{\mathsf{val},\tau} - \mathbf{A}'_i - \mathbf{B}_i, s)$$

  where $\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(f, \left\{ \mathbf{A}_i^{\mathsf{rd},\tau} \right\}_{i \in [L_{\mathsf{rd}}]} \right)$, and $\mathsf{ListMxPK} = \{\mathbf{A}'_i, \mathbf{B}_i, \mathbf{E}_i\}_{i \in [N]}$ is computed from algorithm $\mathsf{TranslatePK}\,(\mathsf{MxPK}_{\mathsf{aux}},\ D)$. We hvae that

$$\left[\mathbf{A} \| \mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G} \| \mathbf{A}'_i + \mathbf{B}_i + D[i]\mathbf{G} \right] \begin{pmatrix} \mathbf{T}_i^{\mathsf{rd},\tau} \\ \mathbf{I} \end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + D[i]\mathbf{G}$$

- Set $\mathsf{StepKey}_\tau = \left\{ \mathbf{T}_i^{\mathsf{rd},\tau} \right\}_{i \in [N]}$ and

$$\mathsf{ListMxPK}_\tau = \left(\mathbf{A}, \mathsf{ListMxPK}, \left\{ \mathbf{A}_i^{\mathsf{st},\tau} \right\}_{i \in [L_{\mathsf{st}}]}, \left\{ \mathbf{A}_i^{\mathsf{rd},\tau} \right\}_{i \in [L_{\mathsf{rd}}]}, \mathbf{A}^{\mathsf{val},\tau} \right)$$

Output $(\mathsf{StepKey}_\tau, \mathsf{ListMxPK}_\tau)$.

$\underline{\mathsf{StepEvalCT}}\,(C, \tau, \mathsf{ListVecCT}_{\tau-1},\ \mathsf{StepKey}_\tau, D)$: the ciphertext evaluation algorithm does the following:

- Parse the ciphertext $\mathsf{ListVecCT}_{\tau-1}$ as

$$\left(\{\mathsf{CT}_{ijk}, \mathsf{CT}'_{ijk}\}_{\substack{i\in[n],\, j\in[m],,\\ k\in[\log q]}}, \left\{\mathsf{CT}_i^{\mathsf{st},\tau-1}\right\}_{i\in[L_{\mathsf{st}}]},\ \left\{\mathsf{CT}_i^{\mathsf{rd},\tau-1}\right\}_{i\in[L_{\mathsf{rd}}]},\ \mathsf{CT}^{\mathsf{val},\tau-1}\right)$$

  along with its associated value $\mathsf{ListST}_{\tau-1} = \left(\{\mathsf{st}^{\tau-1}\}_{i\in[L_{\mathsf{st}}]}, \{\mathsf{rd}^{\tau-1}\}_{i\in[L_{\mathsf{rd}}]}, \mathsf{val}^{\tau-1}\right)$.

- **Ciphertext evaluation**: Compute $\left(\left\{\mathsf{CT}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]},\ \left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right) = \mathsf{CtEval}\,(\mathsf{ListMxPK}_{\tau-1}, \mathsf{ListST}_{\tau-1}, C)$, where algorithm CtEval is defined in Theorem 2.3.11.

- **Ciphertext translation and recoding steps**: Compute

$$\mathsf{CT}^{\mathsf{val},\tau} = \left(\widehat{\mathsf{CT}}, \mathsf{CT}^{\mathsf{rd},\tau}, \mathsf{CT}_{\mathsf{rd}^\tau}\right)\begin{pmatrix}\mathbf{T}_{\mathsf{rd}^\tau}^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix}$$

  where $\mathsf{CT}^{\mathsf{rd},\tau} = \mathsf{CtEval}\left(\left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \{\mathsf{rd}_i^\tau\}_{i\in[L_{\mathsf{rd}}]}, f\right)$ and

$$\mathsf{CT}_{\mathsf{rd}^\tau} = \sum_{j,k,\ell}\left(a_{jk\ell}\mathsf{CT}_{jk\ell} + \bar{a}_{jk\ell}\mathsf{CT}'_{jk\ell}\right)$$

  $a_{jk\ell}$ is ℓ-th bit of the bit-decomposition of (j,k)-th element a_{jk} in matrix $\mathbf{A}_{\mathsf{rd}^\tau} = \mathbf{A}'_{\mathsf{rd}^\tau} + D[\mathsf{rd}^\tau]\mathbf{G}$.

Output $\mathsf{ListVecCT}_\tau = \left(\{\mathsf{CT}_{ijk}, \mathsf{CT}'_{ijk}\}_{\substack{i\in[n],\, j\in[m],,\\ k\in[\log q]}}, \left\{\mathsf{CT}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]},\ \left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]},\ \mathsf{CT}^{\mathsf{val},\tau}\right)$.

We note that $\mathsf{StepEvalCT}$ *incorporates the translation, evaluation and the recoding steps described in the technical overview.*

4.3.2 Our Construction

In our construction below, we assume the initial states are all 1, the initial read address is always the first index of database. We also assume that if the first bit of state is 0, then the RAM program terminates.

Our read-only ABE for RAMs construction $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ can be described as follows:

Setup, $\mathsf{Setup}(1^\lambda, T)$: On input security parameter λ and time bound T, the setup algorithm computes:

- $(\mathbf{A}, \mathbf{T}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(1^n, 1^q, m)$, the anchor matrix and its associated trapdoor.
- $\forall i \in [L_{\mathsf{st}}]$, sample $\mathbf{A}_i^{\mathsf{st},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the initial state.
- $\forall i \in [L_{\mathsf{rd}}]$, sample $\mathbf{A}_i^{\mathsf{rd},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the initial read address.
- $\forall j \in [n], k \in [m], \ell \in \lceil \log q \rceil$, sample $\left(\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right) \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the database.
- For $i \in [\lambda]$, sample $\mathbf{A}_i^{\mathsf{val},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the initial read value.
- Sample $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$, encoding vector for the plaintext.

Set $\mathsf{MxPK}_{\mathsf{aux}} = \left\{\left(\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right)\right\}_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}$. Output $\mathsf{MSK} = (\mathsf{PP}, \mathbf{T}_\mathbf{A})$ and

$$\mathsf{PP} = \left(\mathbf{A}, \mathsf{MxPK}_{\mathsf{aux}}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}, \boldsymbol{u}\right)$$

Key Generation, $\mathsf{KeyGen}(\mathsf{MSK}, P, D)$: On input master secret key MSK, RAM program P with step circuit C and database D, it does the following:

- First compute the translation algorithm

$$\mathsf{ListMxDB} \leftarrow \mathsf{TranslatePK}\left(\mathsf{MxPK}_{\mathsf{aux}},\ D\right)$$

where $\mathsf{ListMxDB} = \left\{\left(\mathbf{A}'_i, \mathbf{B}_i, \mathbf{E}_i\right)\right\}_{i\in[N]}$. Set

$$\mathsf{ListMxPK}_0 = \left(\mathbf{A}, \mathsf{ListMxDB}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}\right)$$

- For $\tau \in [T]$, compute

$$(\mathsf{ListMxPK}_\tau, \mathsf{StepKey}_\tau) \leftarrow \mathsf{StepEvalPK}\left(C, \tau, \mathsf{ListMxPK}_{\tau-1}, \mathbf{T}_{\mathbf{A}}, D\right)$$

- Compute $\boldsymbol{t}^{\mathsf{st},T}$ as

$$\boldsymbol{t}^{\mathsf{st},T} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{A}_1^{\mathsf{st},T}, \boldsymbol{u}, s)$$

such that

$$\left[\mathbf{A}||\mathbf{A}_1^{\mathsf{st},T}\right] \cdot \boldsymbol{t}^{\mathsf{st},T} = \boldsymbol{u}$$

Output $\mathsf{SK}_{P,D} = \left(P, D, \mathsf{ListMxDB}, \{\mathsf{StepKey}_\tau\}_{\tau\in[T]}, \boldsymbol{t}^{\mathsf{st},T}\right)$.

Encryption, $\mathsf{Enc}(\mathsf{PP}, x, \mu)$: On input public parameters PP, input $\boldsymbol{x} \in \{0,1\}^\lambda$, message μ, the encryption algorithm does the following:

- Sample vector $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$ and error vectors $\widehat{\mathbf{e}}, \mathbf{e}^*$ from Gaussian distribution $\mathcal{D}_{\mathbb{Z}^m}$.
- $\forall i \in [L_{\mathsf{st}}]$, compute $\mathsf{CT}_i^{\mathsf{st},0} = \boldsymbol{s}\left(\mathbf{A}_i^{\mathsf{st},0} + \mathbf{G}\right) + \widehat{\mathbf{e}}\mathbf{R}_i^{\mathsf{st},0}$, encoding of the initial state, where $\mathbf{R}_i^{\mathsf{st},0} \leftarrow \{0,1\}^{m\times m}$.
- $\forall i \in [L_{\mathsf{rd}}]$, compute $\mathsf{CT}_i^{\mathsf{rd},0} = \boldsymbol{s}\left(\mathbf{A}_i^{\mathsf{rd},0} + \mathsf{rd}_i^0\mathbf{G}\right) + \widehat{\mathbf{e}}\mathbf{R}_i^{\mathsf{rd},0}$, encoding of the initial read address, where $\mathbf{R}_i^{\mathsf{rd},0} \leftarrow \{0,1\}^{m\times m}$ and $\{\mathsf{rd}_i^0\}_{i\in[L_{\mathsf{rd}}]}$ is the bit representation of 1.
- For $i \in [\lambda]$, compute $\mathsf{CT}_i^{\mathsf{val},0} = \boldsymbol{s}\left(\mathbf{A}_i^{\mathsf{val},0} + \boldsymbol{x}[i]\mathbf{G}\right) + \widehat{\mathbf{e}}\mathbf{R}_i^{\mathsf{val},0}$, encoding of the initial read value, where $\mathbf{R}_i^{\mathsf{val},0} \leftarrow \{0,1\}^{m\times m}$.

- $\forall j \in [n], k \in [m], \ell \in \lceil \log q \rceil$, compute

$$\mathsf{CT}_{jk\ell} = \boldsymbol{s}\left(\mathbf{B}_{jk\ell} + 2^{\ell}\mathbf{M}_{j,k}\right) + \widehat{\mathbf{e}}\mathbf{R}_{jkl}, \quad \mathsf{CT}'_{jk\ell} = \boldsymbol{s}\mathbf{B}'_{jk\ell} + \widehat{\mathbf{e}}\mathbf{R}'_{jkl}$$

  auxiliary encodings, where $\mathbf{R}_{jk\ell}, \mathbf{R}'_{jk\ell} \leftarrow \{0,1\}^{m\times m}$.

- Compute $\widehat{\mathsf{CT}} = \boldsymbol{s}\mathbf{A} + \widehat{\mathbf{e}}$ and $\mathsf{CT}^* = \boldsymbol{s}\boldsymbol{u}^{\mathsf{T}} + \mu\lceil q/2 \rceil + e^*$.

- Set

$$\mathsf{ListVecCT}_0 = \left(\left\{\mathsf{CT}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{CT}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathsf{CT}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]} \{\mathsf{CT}_{ijk}, \mathsf{CT}'_{ijk}\}_{\substack{i\in[n],j\in[m],\\ k\in[\log q]}}\right)$$

Output ciphertext $\mathsf{CT}_{\boldsymbol{x}} = \left(\widehat{\mathsf{CT}}, \mathsf{CT}^*, \mathsf{ListVecCT}_0, \boldsymbol{x}\right)$.

Decryption, $\mathsf{Dec}(\mathsf{SK}_{P,D}, \mathsf{CT}_{\boldsymbol{x}})$: On input secret key $\mathsf{SK}_{P,D}$, ciphertext $\mathsf{CT}_{\boldsymbol{x}}$, the decryption algorithm does the following:

- Output $\perp$ if $P^D(\boldsymbol{x}) \neq 0$.

- For $\tau \in [T]$, compute,

$$\mathsf{ListVecCT}_{\tau} \leftarrow \mathsf{StepEvalCT}\left(C, \tau, \mathsf{ListVecCT}_{\tau-1}, \mathsf{StepKey}_{\tau}, D\right)$$

Check if $\left\|\left([\widehat{\mathsf{CT}}\|\mathsf{CT}_1^{\mathsf{st},T}] \cdot (\boldsymbol{t}^{\mathsf{st},T})^{\mathsf{T}}\right) - \mathsf{CT}^*\right\|_{\infty} < q/4$ and if so, output 0, otherwise output 1.

4.3.3 Analysis of Correctness, Efficiency and Parameters

In this part, we show that the ABE construction described above is correct (c.f. Definition 4.2.1), then analysis decryption time and set lattice parameters afterwards.

Lemma 4.3.1. *The ABE construction for read-only RAMs satisfies correctness as defined in Definition 4.2.1.*

Proof. Let the ciphertext be CT_x and secret key be $\mathsf{SK}_{P,D}$, such that $P^D(x) = 0$. At the τ-th step, by evaluating the ciphertext using algorithm $\mathsf{StepEvalCT}$ with respect to the step circuit, we have $\left\{\mathsf{CT}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}$, $\left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}$ are encryption of state and read address at the τ-th step respectively. Unfolding ciphertext $\mathsf{CT}^{\mathsf{val},\tau}$ (ignoring the error terms), we obtain

$$\begin{aligned}
\mathsf{CT}^{\mathsf{val},\tau} &= \left(\widehat{\mathsf{CT}}, \mathsf{CT}^{\mathsf{rd},\tau}, \mathsf{CT}_{\mathsf{rd}^\tau}\right)\begin{pmatrix}\mathbf{T}_k^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix}\\
&\approx \boldsymbol{s}\left[\mathbf{A}\|\mathbf{A}^{\mathsf{rd},\tau} + \mathsf{rd}^\tau\mathbf{G}\|\mathbf{E}_{\mathsf{rd}^\tau}\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix}\\
&\approx \mathbf{A}^{\mathsf{val},\tau} + D[\mathsf{rd}^\tau]\mathbf{G}
\end{aligned}$$

Thus, ciphertext $\mathsf{CT}^{\mathsf{val},\tau}$ encodes the read value of database at rd^τ index, which can be used in the next step evaluation.

Suppose at step t, where $t \le T$, we have $P^D = 0$, then $\mathsf{CT}_1^{\mathsf{st},t}$ encrypts state value 0. Thus,

$$\begin{aligned}
\left([\widehat{\mathsf{CT}}\|\mathsf{CT}_1^{\mathsf{st},t}]\cdot\boldsymbol{t}^{\mathsf{st},t}\right) - \mathsf{CT}^* &= \boldsymbol{s}\left[\mathbf{A}\|\mathbf{A}_1^{\mathsf{st},\tau}\right]\cdot(\boldsymbol{t}^{\mathsf{st},t})^{\mathsf{T}} + \mathbf{e}^t - \mathsf{CT}^*\\
&= \mathbf{e}^t - \mu\lceil q/2\rceil - \mathbf{e}^*
\end{aligned}$$

By setting parameters appropriately as below, our ABE scheme is correct. □

Parameters Setting. If the step circuit being evaluated has length d, then the noise in ciphertext grows in the worst case by a factor of $O(m^d)$. Thus, to support a RAM program with maximum running time T (the unit of time corresponds to one step), we set (n, m, q) as

- Lattice dimension n is an integer such that $n \geq (Td \log n)^{1/\epsilon}$, for some fixed $0 < \epsilon < 1/2$.
- Modulus q is set to be $q = 2^{n^\epsilon}$, since the noise in the ciphertexts grows by a factor of $O(m^{Td})$. Hence, we need q to be on the order of $\Omega(Bm^{Td})$, where $B = O(n)$ is the maximum magnitude of noise (from discrete Gaussian distribution) added during encryption. To ensure correctness of decryption and hardness of LWE, we set $q = 2^{n^\epsilon}$.
- Lattice column parameter m is set to be $m = \Theta(n \log q)$ to make the leftover hash lemma hold.

The parameter s used in algorithms SampleLeft and SampleRight are set as $s > \sqrt{n \log q} \cdot \omega(\sqrt{\log m})$, as required by Lemma 2.3.7.

For security we rely on the hardness of the LWE problem, which requires that the ratio q/B is not too large, where $B = O(n)$ is the maximum magnitude of noise (from discrete Gaussian distribution) added during encryption. In particular, the underlying problem is believed to be hard even when q/B is 2^{n^ϵ}.

Efficiency Analysis. The (space/time) complexity of our construction can be analyzed by the following aspects. The polynomial $n(\cdot, \cdot)$ denotes the lattice dimension.

- The public parameters contain $(L_{\mathsf{st}} + L_{\mathsf{rd}} + nmT)$ random $n \times m$ matrices in $\mathbb{Z}_q$, which is $\tilde{O}(n(\lambda, T)^2 \cdot n^2 T^2)$ in bit complexity. The master secret key is one $m \times m$ matrix.
- The secret key for program and database pair (P, D) contains $T(N+1)$ small $m \times m$ matrices, which is $\tilde{O}(n(\lambda, T)^2 \cdot NT)$ in bit complexity.

- The ciphertext for input $\boldsymbol{x}$ contains $(L_{\mathsf{st}} + L_{\mathsf{rd}} + nmT + \lambda)$ dimension-m vectors in $\mathbb{Z}_q$, which is $\tilde{O}(n(\lambda, T) \cdot \lambda n^2 T^2)$ in bit complexity.
- Decryption involves matrix-vector multiplication. The time complexity of decryption is $\tilde{O}(T)$.

Next, we would like to show the following: if a program P^D on input $\boldsymbol{x}$ takes time at most T then correspondingly, the decryption of secret key for P^D on input an encryption of message μ associated with attribute input x takes time $p(\lambda, T)$, for a fixed polynomial p.

We analyze the time to decrypt an encryption of database x associated with message μ using a key of RAM program/database with runtime bounded by T. The essential algorithm StepEvalCT, which may be computed T times, in decryption algorithm can be divided into two steps, as analyzed below

- **Step circuit**: The runtime of CtEval with respect to step circuit C is a polynomial in $(\lambda, L_{\mathsf{st}}, L_{\mathsf{rd}})$. Observe that L_{st} is the length of the state, which is independent of the input length, and $L_{\mathsf{rd}} = \log N$. Thus, the runtime of CtEval is upper bounded by a polynomial in $(\lambda, L_{\mathsf{st}})$.
- **Recoding part**: In this step, we compute CtEval with respect to the gadget circuit f, then the translation part, and last multiplication. This part is upper bounded by a polynomial in $(\lambda, L_{\mathsf{rd}})$.

From the above observations, it follows that the runtime of the decryption algorithm is a polynomial in (λ, T), where the polynomial is independent of the length of the database. In particular, notice that if T is polylogarithmic in the input length then the decryption time is sub-linear in the input length.

4.3.4 Security Proof

In this part, we show the security of our ABE for read-only RAM construction, assuming the hardness of LWE assumption. We first describe algorithms (Sim.Setup, Sim.Enc, Sim.StepEvalPK) in the following:

- Sim.Setup produces "programmed" public parameters. That is, every pubic matrix produced as part of algorithm Sim.Setup has hardwired in it, a bit of the challenge ciphertext, initial state, read address, etc.
- Sim.Enc produces a simulated encryption of the message.
- Sim.StepEvalPK takes as input the $(\tau - 1)$-th layer of simulated public keys $\mathsf{Sim.ListMxPK}_{\tau-1}$ and produces the τ-th layer of simulated public keys $\mathsf{Sim.ListMxPK}_{\tau}$ and τ-th layer of step keys $\mathsf{Sim.StepKey}_{\tau}$.

These simulated algorithms can be constructed as follows:

$\underline{\mathsf{Sim.Setup}}(1^\lambda, \boldsymbol{x}^*)$: On input the challenge input $\boldsymbol{x}^*$, the simulated setup algorithm does:

- Compute $(\mathbf{A}, \mathbf{T}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(1^n, 1^q, m)$ and sample $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$.
- $\forall i \in [L_{\mathsf{st}}]$, set $\mathbf{A}_i^{\mathsf{st},0} = \mathbf{A}\mathbf{R}_i^{\mathsf{st},0} - \mathbf{G}$, where $\mathbf{R}_i^{\mathsf{st},0} \leftarrow \{0,1\}^{m\times m}$.
- $\forall i \in [L_{\mathsf{rd}}]$, set $\mathbf{A}_i^{\mathsf{rd},0} = \mathbf{A}\mathbf{R}_i^{\mathsf{rd},0} - \mathbf{G}$, where $\mathbf{R}_i^{\mathsf{rd},0} \leftarrow \{0,1\}^{m\times m}$.
- $\forall j \in [n], k \in [m], \ell \in \lceil \log q \rceil$, set

$$\mathbf{B}_{jk\ell} = \mathbf{A}\mathbf{R}_{jk\ell} - 2^\ell \mathbf{M}_{j,k}, \quad \mathbf{B}'_{jk\ell} = \mathbf{A}\mathbf{R}'_{jk\ell}$$

  where $(\mathbf{R}_{jk\ell}, \mathbf{R}'_{jk\ell}) \leftarrow \{0,1\}^{m\times m}$.

- $\forall i \in [\lambda]$, set $\mathbf{A}^{\mathsf{val},0} = \mathbf{A}\mathbf{R}_i^{\mathsf{val},0} - \boldsymbol{x}^*[i]\mathbf{G}$, where $\mathbf{R}_i^{\mathsf{val},0} \leftarrow \{0,1\}^{m\times m}$.

Let $\mathsf{Sim.MxPK}_{\mathsf{aux}} = \left(\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right)_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}$, and denote trapdoor matrix for initial step as

$$\mathsf{ListMxTD}_0 = \left(\left\{\mathbf{R}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{R}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathbf{R}^{\mathsf{val},0}\right)$$

Output $\mathsf{MSK} = (\mathsf{PP}, \mathbf{T}_{\mathbf{A}})$ and

$$\mathsf{Sim.PP} = \left(\mathbf{A}, \mathsf{Sim.MxPK}_{\mathsf{aux}}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}, \boldsymbol{u}\right)$$

$\underline{\mathsf{Sim.Enc}(\mathsf{Sim.PP}, \boldsymbol{x}^*, 1^{|\mu|}, (\mathbf{A}, \boldsymbol{u}), (\boldsymbol{b}, b'))}$: On input simulated public parameters $\mathsf{Sim.PP}$, challenge input $\boldsymbol{x}^*$ and message length $|\mu|$ and LWE instance $((\mathbf{A}, \boldsymbol{u}), (\boldsymbol{b}, b'))$, the simulated encryption algorithm does

- $\forall i \in [L_{\mathsf{st},0}]$, compute $\mathsf{CT}_i^{\mathsf{st},0} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{st},0}$, where $\mathbf{R}_i^{\mathsf{st},0}$ is generated in $\mathsf{Sim.Setup}$.
- $\forall i \in [L_{\mathsf{rd},0}]$, compute $\mathsf{CT}_i^{\mathsf{rd},0} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{rd},0}$, where $\mathbf{R}_i^{\mathsf{rd},0}$ is generated in $\mathsf{Sim.Setup}$.
- $\forall i \in [\lambda]$, compute $\mathsf{CT}_i^{\mathsf{val},0} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{val},0}$, where $\mathbf{R}_i^{\mathsf{val},0}$ is generated in $\mathsf{Sim.Setup}$.
- $\forall j \in [n], k \in [m], \ell \in \lceil\log q\rceil$, compute $\mathsf{CT}_{jk\ell} = \boldsymbol{b}\mathbf{R}_{jk\ell}$, $\mathsf{CT}'_{jk\ell} = \boldsymbol{b}\mathbf{R}'_{jk\ell}$ where $\left(\mathbf{R}_{jk\ell}, \mathbf{R}'_{jk\ell}\right)$ is generated in $\mathsf{Sim.Setup}$.
- Set $\widehat{\mathsf{CT}} = \boldsymbol{b}$ and $\mathsf{CT}^* = b'$.
- Define $\mathsf{ListVecCT}_0$ in the same way as the real scheme.

Output challenge ciphertext $\mathsf{CT}_{x^*} = \left(\widehat{\mathsf{CT}}, \mathsf{CT}^*, \mathsf{ListVecCT}_0, x^*\right)$.

$\underline{\mathsf{Sim.StepEvalPK}(C, \tau, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP}, D)}$: On input the step circuit C of program P satisfying $P^D(\boldsymbol{x}^*) = 1$, step index τ, simulated $(\tau-1)$-th layer of simulated public keys $\mathsf{Sim.ListMxPK}_{\tau-1}$, simulated public parameters $\mathsf{Sim.PP}$ and

database query D, if $\tau = 1$, compute the translation algorithm

$$\mathsf{ListMxDB} \leftarrow \mathsf{TranslatePK}\left(\mathsf{MxPK}_{\mathsf{aux}},\ D\right)$$

where $\mathsf{ListMxDB} = \left\{\left(\mathbf{A}'_i, \mathbf{B}_i, \mathbf{E}_i\right)\right\}_{i\in[N]}$. Set

$$\mathsf{ListMxPK}_0 = \left(\mathbf{A}, \mathsf{ListMxDB}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}\right)$$

Otherwise, it does:

- Compute $\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right) = \mathsf{PubEval}\left(\mathsf{Sim.ListMxPK}_{\tau-1}, C\right)$, and then $\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(f, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right)$, where $\mathbf{A}^{\mathsf{rd},\tau}$ encodes the actual read address rd^τ of $P^D(x^*)$ at τ-th step.
- Sample $\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}^\tau} = \left(\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}^\tau,0}, \mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}_\tau,1}\right) \leftarrow \mathcal{D}_{Z^{m\times m}}$ and set $\mathbf{A}^{\mathsf{val},\tau} = \mathbf{A}\left(\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}^\tau,0} + \mathbf{R}^{\mathsf{rd},\tau}\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}_\tau,1} + \mathbf{R}_i\right)$, where $\mathbf{R}^{\mathsf{rd},\tau} = \mathsf{TrapEval}\left(f \circ C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{ListMxTD}_{\tau-1}\right)$ and $\mathbf{R}_i = \sum_{jk\ell}\left(d_{jk\ell}\mathbf{R}_{jk\ell} + \bar{d}_{jk\ell}\mathbf{R}'_{jk\ell}\right)$ and algorithm $\mathsf{TrapEval}$ is defined in Theorem 2.3.11.
- For $i \in [N] - \{\mathsf{rd}^\tau\}$, compute $\mathbf{T}_i^{\mathsf{rd},\tau}$ as

  $$\mathbf{T}_i^{\mathsf{rd},\tau} \leftarrow \mathsf{SampleRight}\left(\mathbf{A}, (i-\mathsf{rd}^\tau)\mathbf{G}, \mathbf{R}^{\mathsf{rd},\tau}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^{\mathsf{val},\tau} - \mathbf{A}'_i - \mathbf{B}_i, s\right)$$

  such that

  $$\left[\mathbf{A}||\mathbf{A}\mathbf{R}^{\mathsf{rd},\tau} + (i-\mathsf{rd}^\tau)\mathbf{G}||\mathbf{E}_i\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + D[i]\mathbf{G}$$

  where $\{\mathbf{A}'_i, \mathbf{B}_i, \mathbf{E}_i\}$ is computed by algorithm $\mathsf{TranslatePK}$.
- As $P^D(\boldsymbol{x}^*) = 1$, so $\mathbf{A}_1^{\mathsf{st},T}$ is the encoding of 1, i.e., $\mathbf{A}_1^{\mathsf{st},T} = \mathbf{A}\mathbf{R}_1^{\mathsf{st},T} - \mathbf{G}$. Compute $\boldsymbol{t}^{\mathsf{st},T}$ as

  $$\boldsymbol{t}^{\mathsf{st},T} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}_1^{\mathsf{st},T}, \mathbf{T}_{\mathbf{G}}, \boldsymbol{u}, s)$$

  such that

  $$\left[\mathbf{A}||\mathbf{A}\mathbf{R}_1^{\mathsf{st},T} - \mathbf{G}\right]\cdot\boldsymbol{t}^{\mathsf{st},T} = \boldsymbol{u}$$

Set $\mathsf{Sim.StepKey}_\tau = \left\{\mathbf{T}_i^{\mathsf{rd},\tau}\right\}_{i\in[N]}$. Output $(\mathsf{Sim.StepKey}_\tau, \mathsf{Sim.ListMxPK}_\tau)$.

Theorem 4.3.2. *Assuming the hardness of LWE assumption (with parameters as specified above), our ABE construction is secure (c.f. Definition 4.2.5).*

Proof. Let Q be the number of key queries made by the adversary. We first describe a sequence of hybrids in the following:

Hybrid Hyb_1: This corresponds to the real experiment:

- $\mathcal{A}$ specifies challenge attribute input $\boldsymbol{x}^*$ and message μ.
- Challenger computes $\mathsf{Setup}(1^\lambda)$ to obtain the public parameters PP and secret key MSK. Then challenger generates the challenge ciphertext $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{PP}, \boldsymbol{x}^*, \mu)$. It sends CT^* and PP to $\mathcal{A}$.
- For $\gamma \in [Q]$, adversary $\mathcal{A}$ specifies the programs/database (P_γ, D_γ) such that $P_\gamma^{D_\gamma}(\boldsymbol{x}^*) = 1$. Challenger generates the attribute keys for (P_γ, D_γ), for $\gamma \in [Q]$, $\mathsf{SK}_{P_\gamma,D_\gamma} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, P_\gamma, D_\gamma)$.
- Let b be the output of adversary. Output b.

Hybrid Hyb_2: Hyb_2 is the same as Hyb_1 except that it uses $\mathsf{Sim.Setup}(1^\lambda, \boldsymbol{x}^*)$ to generate $\mathsf{Sim.PP}$.

Hybrid $\{\mathsf{Hyb}_{3,i,j}\}_{i\in[Q],j\in[T]}$: Simply put, in hybrid $\mathsf{Hyb}_{3,i,j}$, for $\gamma < i$, the secret key for query (P_γ, D_γ) is simulated. For query (P_i, D_i), upto the j-th step, the step keys are simulated. For $\tau > j$, the step keys are normally generated. For query (P_γ, D_γ), where $\gamma > i$, the secret key is normally generated. We describe it in details below:

- Adversary specifies challenge attribute input $\boldsymbol{x}^*$ and message μ.

- Challenger computes $\mathsf{Sim.Setup}(1^\lambda)$ to obtain the public parameters PP and secret key MSK. Then challenger generates the challenge ciphertext $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{PP}, x^*, \mu)$. It sends CT^* and PP to $\mathcal{A}$.
- For $\gamma \in [Q]$, adversary $\mathcal{A}$ specifies the program/database (P_γ, D_γ) such that $P_\gamma^{D_\gamma}(x^*) = 1$. Challenger generates the secret key $\mathsf{SK}_{P_\gamma, D_\gamma}$ as

$$\mathsf{SK}_{P_\gamma, D_\gamma} = (P_\gamma, D_\gamma, \{\mathsf{StepKey}_\tau\}_{\tau \in [T]}, \boldsymbol{t}^{\mathsf{st},T})$$

  - For $\gamma < i$, answer secret key query P_γ as
    1. For every $\tau \in [T]$, compute

$$(\mathsf{Sim.ListMxPK}_\tau, \mathsf{Sim.StepKey}_\tau) \leftarrow \mathsf{Sim.StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})$$

    2. Set $\mathsf{SK}_\gamma = (\{\mathsf{Sim.StepKey}_\tau\}_{\tau \in [T]})$.
  - For $\gamma = i$, answer secret key query (P_i, D_i) as
    1. For $\tau < j$, generate

$$(\mathsf{Sim.ListMxPK}_\tau, \mathsf{Sim.StepKey}_\tau) \leftarrow \mathsf{Sim.StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})$$

    2. For $\tau \geq j$, generate

$$(\mathsf{ListMxPK}_\tau, \mathsf{StepKey}_\tau) \leftarrow \mathsf{StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})$$

    Set $\mathsf{SK}_i = (\{\mathsf{Sim.StepKey}_\tau\}_{\tau<i}, \{\mathsf{StepKey}_\tau\}_{\tau \geq i})$.
  - For $\gamma > i$, answer secret key query (P_γ, D_γ) as
    1. For every $\tau \in [T]$, generate

$$(\mathsf{ListMxPK}_\tau, \mathsf{StepKey}_\tau) \leftarrow \mathsf{StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})$$

2. Set $\mathsf{SK}_\gamma = (\{\mathsf{StepKey}_\tau\}_{\tau\in[T]})$.

Hybrid Hyb_4: Hyb_4 is the same as $\mathsf{Hyb}_{3,Q,T}$ except that the anchor public key $\mathbf{A}$ is sampled randomly from $\mathbb{Z}_q^{n\times m}$. In $\mathsf{Hyb}_{3,Q,T}$ the secret keys for all queries are simulated without using $\mathsf{MSK} = \mathbf{T}_\mathbf{A}$.

Hybrid Hyb_5: Hyb_5 is the same as Hyb_4 except that it uses algorithm $\mathsf{Sim.Enc}$ to generate the challenge ciphertext.

Indistinguishability of Hybrids.

Lemma 4.3.3. *By Leftover Hash Lemma 5.2.2, the output distributions of hybrids* Hyb_1 *and* Hyb_2 *are statistically close.*

Proof. The only difference between hybrid Hyb_1 and Hyb_2 is in hybrid Hyb_2, the public parameters are generated using $\mathsf{Sim.Setup}(1^\lambda, x^*)$, where they are set like this: $\mathbf{AR}_i - \mathsf{val}\cdot\mathbf{G}$ for $\mathbf{R}_i \xleftarrow{\$} \{0,1\}^{m\times m}$ and $\mathsf{val}\in\{0,1\}$. We point out that one has to be careful when applying Leftover Hash Lemma here, as the anchor matrix $\mathbf{A}$ is only statistically close to uniform, and it is generated together with $\mathbf{T}_\mathbf{A}$. However, by Markov chain property, we can first sample $\mathbf{A}$, and then sample $\mathbf{T}_\mathbf{A}$ and $\mathbf{AR}_i$ independently from the marginals. Therefore, since $(\mathbf{A}, \mathbf{AR}_i)$ is statistically close to uniform when $\mathbf{A}$ is uniform, it also holds true when $\mathbf{A}$ is only statistically close to uniform and when $\mathbf{T}_\mathbf{A}$ is known as well. Thus we have that the output distributions of hybrids Hyb_1 and Hyb_2 are statistically close. □

Lemma 4.3.4. *The output distributions of hybrids* Hyb_2 *and* $\mathsf{Hyb}_{3,1,0}$ *are identical.*

Proof. As described above, hybrid $\mathsf{Hyb}_{3,1,0}$ is obtained by using $\mathsf{Sim.Setup}$ to simulate public parameters, containing $\mathsf{Sim.ListMxPK}_0$, and generating all se-

cret keys normally. Therefore, we have that the output distributions of hybrids Hyb_2 and $\mathsf{Hyb}_{3,1,0}$ are identical. □

Lemma 4.3.5. *By properties of sampling algorithms (Lemma 2.3.7 and 2.3.8) and Left-over Hash Lemma 5.2.2, the output distributions of hybrids* $\mathsf{Hyb}_{3,i,j}$ *and* $\mathsf{Hyb}_{3,i,j+1}$ *are statistically close.*

Proof. The main difference between hybrid $\mathsf{Hyb}_{3,i,j}$ and $\mathsf{Hyb}_{3,i,j+1}$ is the generation of temporary matrices $\mathsf{ListMxPK}_{j+1}$ and $\mathsf{StepKey}_j$, which is in the SK_i. In $\mathsf{Hyb}_{3,i,j+1}$, we use algorithm $\mathsf{Sim.StepEvalPK}$ to generate $\mathsf{Sim.ListMxPK}_{j+1}, \mathsf{Sim.StepKey}_j$.

In $\mathsf{Sim.StepKey}_j$, matrices $\{\mathbf{T}_i^{\mathsf{rd},\tau}\}_{i\in[N]}$ are generated either using algorithm $\mathsf{SampleRight}$ or sampling from Gaussian distribution $\mathcal{D}$. By Lemma 2.3.7, we obtain that the distribution of $\mathsf{Sim.StepKey}_j$ is statistically close to $\mathsf{StepKey}_j$.

Next, we discuss the temporary public matrices. The only difference between $\mathsf{ListMxPK}_{j+1}$ and $\mathsf{Sim.ListMxPK}_{j+1}$ is the generation of $\mathbf{A}^{\mathsf{val},j}$. Instead of sampling $\mathbf{A}^{\mathsf{val},\tau}$ from random, in $\mathsf{Sim.ListMxPK}_{j+1}$, we set $\mathbf{A}^{\mathsf{val},j} = \mathbf{A}\left(\mathbf{T}_{\mathsf{rd}_\tau,0}^{\mathsf{rd},j} + \mathbf{R}^{\mathsf{rd},j} + \mathbf{R}_i\right)$, where $\mathbf{T}_{\mathsf{rd}_\tau,0}^{\mathsf{rd},j}, \mathbf{R}^{\mathsf{rd},j}, \mathbf{R}_i$ comes from discrete Gaussian distribution. Thus by lemma 2.3.8, we have that the distribution of $\mathsf{Sim.ListMxPK}_{j+1}$ is statistically close to $\mathsf{ListMxPK}_{j+1}$.

Combing the two components, we conclude that output distributions of hybrids $\mathsf{Hyb}_{3,i,j}$ and $\mathsf{Hyb}_{3,i,j+1}$ are statistically close. □

Lemma 4.3.6. *The output distributions of hybrids* $\mathsf{Hyb}_{3,i,T}$ *and* $\mathsf{Hyb}_{3,i+1,1}$ *are identical.*

Proof. As described above, the only difference between hybrid $\mathsf{Hyb}_{3,i,T}$ and $\mathsf{Hyb}_{3,i+1,1}$ is that in $\mathsf{Hyb}_{3,i+1,1}$ the public matrices for initial step, $\mathsf{Sim.ListMxPK}_0$,

is simulated. As $\mathsf{Sim.ListMxPK}_0$ does not exist in SK_{i+1}, we have the output distributions of hybrids $\mathsf{Hyb}_{3,i,T}$ and $\mathsf{Hyb}_{3,i+1,1}$ are identical. □

Lemma 4.3.7. *By Lemma 2.3.6, the output distributions of hybrids* $\mathsf{Hyb}_{3,Q,T}$ *and* Hyb_4 *are statistically close.*

Proof. The only difference between hybrid $\mathsf{Hyb}_{3,Q,T}$ and Hyb_4 is that in Hyb_4, the anchor public matrix $\mathbf{A}$ is sampled from uniform distribution over $\mathbb{Z}_q^{n\times m}$ instead of using algorithm $\mathsf{TrapGen}$. By Lemma 2.3.6, since $\mathsf{TrapGen}$ sampled $\mathbf{A}$ which is statistically close to uniform distribution, we conclude that the output distributions of hybrids $\mathsf{Hyb}_{3,Q,T}$ and Hyb_4 are statistically close. □

Lemma 4.3.8. *Assuming the hardness of LWE assumption, the output distributions of hybrids* Hyb_4 *and* Hyb_5 *are computationally close.*

Proof. The only difference between hybrid Hyb_4 and Hyb_5 is the generation of challenge ciphertext. In Hyb_5, the challenge ciphertext is generated using algorithm $\mathsf{Sim.Enc}$. The simulated ciphertext is simulated using LWE instance. By unfolding one component of ciphertext,

$$\mathsf{CT}_i^{\mathsf{st},0,} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{st},0} = (\boldsymbol{s}\mathbf{A} + \boldsymbol{e})\mathbf{R}_i^{\mathsf{st},0}$$

which is of the same form of normally generated ciphertext. By the hardnss of LWE assumption, we have that the distribution of$(\widehat{\mathsf{CT}}, \mathsf{CT}^*)$ in simulated ciphertext is computationally close to it normally generated counterpart. Also by Leftover Hash Lemma 5.2.2, the distribution of $\left(\mathsf{PP}, \mathsf{CT}^{D^*} - \{\widehat{\mathsf{CT}}, \mathsf{CT}^*\}\right)$ is statistically close to $\left(\mathsf{Sim.PP}, \mathsf{Sim.CT}_{D^*} - \{\widehat{\mathsf{CT}}, \mathsf{CT}^*\}\right)$.

There we conclude that the output distributions of hybrids Hyb_4 and Hyb_5 are computationally close. □

Combining the hybrids and lemmas proved above, we prove that our ABE construction for read-only RAMs is secure, as defined in Definition 4.2.5. □

4.4 ABE for RAMs: Handling Write Operations

In this part, we show how to construct ABE for a full-fledged RAM. The step circuit C here is defined as

$$(\mathsf{st}_i, \mathsf{addr}_i^{\mathsf{wt}}, b_i^{\mathsf{wt}}, \mathsf{addr}_i^{\mathsf{rd}}) \leftarrow C(\mathsf{st}_{i-1}, \mathsf{addr}_{i-1}^{\mathsf{rd}}, b_{i-1}^{\mathsf{rd}})$$

One additional parameter we use here, $L_{\mathsf{wt}} = \log N$ for write address bit-length. Then we define an array for write operations upto step τ as $\mathsf{trace}^\tau = \{(i, \mathsf{wt}^i)\}_{i=1}^{\tau-1}$, namely it records all the write operations from the first step to the $(\tau-1)$-th step. We also define an auxiliary circuit $C^{\mathsf{up}}(\tau, \mathsf{trace}^\tau, \mathsf{rd}^\tau)$ as

- $C^{\mathsf{up}}(\tau, \mathsf{trace}^\tau, \mathsf{rd}^\tau)$: On input the step index τ, the write trace trace^τ and read address $\mathsf{rd}^\tau \in [N]$, the update circuit outputs the correct address $\widehat{\mathsf{rd}}^\tau \in [N + \tau - 1]$ to read.

The intuition of handling write operations is: The database here is an append-only data structure. Instead of updating the database after each write, we write the newly computed value to a new location. For instance, at the τ-th step, the value is writed at the end of the append-only data structure, i.e., the $(N + \tau)$-th location. For each read operation, we first compute the update circuit $C^{\mathsf{up}}(\tau, \mathsf{trace}_\tau, \mathsf{rd}_\tau)$ to obtain the correct address to read.

Description of StepEvalPK **and** StepEvalCT**.** The translation process in the write case is the same as read-only one. We revise the construction of algo-

rithms StepEvalPK and StepEvalCT as follows:

StepEvalPK $(C, \tau, \mathsf{ListMxPK}_{\tau-1}, \mathbf{T}_{\mathbf{A}}, D)$: The key evaluation algorithm does:

- Parse the encoding matrices $\mathsf{ListMxPK}_{\tau-1}$ as

$$\left(\mathbf{A}, \mathsf{ListMxDB}, \left\{\mathbf{A}_i^{\mathsf{st},\tau-1}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau-1}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathbf{A}^{\mathsf{val},\tau-1}\right)$$

- Compute the step circuit as

$$\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{wt},\tau}\right\}_{i\in[L_{\mathsf{wt}}]}, \mathbf{A}^{\mathsf{wb},\tau}\right) = \mathsf{PubEval}\left(\mathsf{ListMxPK}_{\tau-1}, C\right)$$

  Set $\mathbf{A}'_{N+\tau} + \mathbf{B}_{N+\tau} = \mathbf{A}^{\mathsf{wb},\tau}$, where $(\mathbf{A}'_{N+\tau}, \mathbf{B}_{n+\tau})$ are newly initiated in ListMxDB.

- Compute the update circuit as

$$\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(\{i, \mathbf{A}^{\mathsf{wt},i}\}_{i=1}^{\tau-1}, \{\mathbf{A}_i^{\mathsf{rd},\tau}\}_{i\in[L_{\mathsf{rd}}]}, C^{\mathsf{up}}\right)$$

- Sample $\mathbf{A}^{\mathsf{val},\tau} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$. For $i \in [N+\tau-1]$, compute $\mathbf{T}_i^{\mathsf{rd},\tau}$ as

$$\mathbf{T}_i^{\mathsf{rd},\tau} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G}, \mathbf{A}^{\mathsf{val},\tau} - \mathbf{A}'_i - \mathbf{B}_i, s)$$

  where $\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(f, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right)$, such that

$$\left[\mathbf{A}||\mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G}||\mathbf{A}'_i + \mathbf{B}_i + b\mathbf{G}\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + b\mathbf{G}$$

  where matrices $(\mathbf{A}'_i, \mathbf{B}_i)$ (in ListMxDB) are computed by algorithm TranslatePK as before.

Set $\mathsf{StepKey}_\tau = \left\{\mathbf{T}_i^{\mathsf{rd},\tau}\right\}_{i=1}^{N+\tau-1}$. Output $(\mathsf{StepKey}_\tau, \mathsf{ListMxPK}_\tau)$.

StepEvalCT $(C, \tau, \mathsf{ListVecCT}_{\tau-1}, \mathsf{StepKey}_\tau, D)$: the ciphertext evaluation algorithm does the following:

- Parse the ciphertext $\mathsf{ListVecCT}_{\tau-1}$ as

$$\left(\{\mathsf{CT}_{ijk}, \mathsf{CT}'_{ijk}\}_{\substack{i\in[n],j\in[m],\\ k\in[\log q]}}, \left\{\mathsf{CT}_i^{\mathsf{st},\tau-1}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{CT}_i^{\mathsf{rd},\tau-1}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathsf{CT}^{\mathsf{val},\tau-1}\right)$$

along with its associated value $\mathsf{ListST}_{\tau-1} = \left(\{\mathsf{st}^{\tau-1}\}_{i\in[L_{\mathsf{st}}]}, \{\mathsf{rd}^{\tau-1}\}_{i\in[L_{\mathsf{rd}}]}, \mathsf{val}^{\tau-1}\right)$.

- Compute

$$\left(\left\{\mathsf{CT}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathsf{CT}_i^{\mathsf{wt},\tau}\right\}_{i\in[L_{\mathsf{wt}}]}, \mathsf{CT}^{\mathsf{wb},\tau}\right) = \mathsf{CtEval}\left(\mathsf{ListMxPK}_{\tau-1}, \mathsf{ListST}_{\tau-1}, C\right)$$

- Compute

$$\mathsf{CT}^{\mathsf{rd},\tau} = \mathsf{CtEval}\left(\{i, \mathsf{CT}^{\mathsf{wt},i}\}_{i=0}^{\tau-1}, \left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, C^{\mathsf{up}}\right)$$

along with the actual read address $\mathsf{rd}^\tau \in [N+\tau-1]$.

- Choose the corresponding $\mathbf{T}_{\mathsf{rd}^\tau}^{\mathsf{rd},\tau}$ and compute

$$\mathsf{CT}^{\mathsf{val},\tau} = \left(\widehat{\mathsf{CT}}, \mathsf{CT}^{\mathsf{rd},\tau}, \mathsf{CT}_{\mathsf{rd}^\tau}\right)\begin{pmatrix}\mathbf{T}_{\mathsf{rd}^\tau}^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix}$$

where $\mathsf{CT}^{\mathsf{rd},\tau} = \mathsf{CtEval}\left(\left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \{\mathsf{rd}_i^\tau\}_{i\in[L_{\mathsf{rd}}]}, f\right)$ and

$$\mathsf{CT}_{\mathsf{rd}^\tau} = \sum_{j,k,\ell}\left(d_{jk\ell}\mathsf{CT}_{jk\ell} + \bar{d}_{jk\ell}\mathsf{CT}'_{jk\ell}\right), \text{if } \mathsf{rd}^\tau \in [N]$$

$d_{jk\ell}$ is ℓ-th bit of the bit-decomposition of (j,k)-th element d_{jk} in matrix $\mathbf{A}_{\mathsf{rd}^\tau}$, or ($\mathsf{rd}^\tau \in [N+1, N+\tau-1]$) $\mathsf{CT}_{\mathsf{rd}^\tau}$ is from previous write operations.

Output $\mathsf{ListVecCT}_\tau = \left(\{\mathsf{CT}_{ijk}, \mathsf{CT}'_{ijk}\}_{\substack{i\in[n],j\in[m],\\ k\in[\log q]}}, \left\{\mathsf{CT}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathsf{CT}^{\mathsf{val},\tau}\right)$.

Construction, Efficiency Analysis and Security Proof. The construction of ABE for full-fledged RAMs based on $\mathsf{StepEvalPK}$ and $\mathsf{StepEvalCT}$ is similar to the counterpart of read-only RAMs. We sketch the construction below:

- $\mathsf{Setup}(1^\lambda, T)$: On input security parameter λ and time bound T, the setup algorithm generates $\left(\mathbf{A}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}, \{\left(\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right)\}, \left\{\mathbf{A}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}, \boldsymbol{u}\right)$ in the same way as the read-only case. Additionally, it also samples random matrix $\mathbf{A}^{\mathsf{wt},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, the encoding matrix for initial write address. Let $\mathsf{MxPK}_{\mathsf{aux}} = \left\{\left(\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right)\right\}_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}$. Output $\mathsf{MSK} = (\mathsf{PP}, \mathbf{T}_{\mathbf{A}})$ and

$$\mathsf{PP} = \left(\mathbf{A}, \mathsf{ListMxDB}, \mathbf{A}^{\mathsf{wt},0}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}, \left\{\mathbf{A}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}, \boldsymbol{u}\right)$$

- $\mathsf{KeyGen}(\mathsf{MSK}, P, D)$: The key generation algorithm is the same as the read-only construction.
- $\mathsf{Enc}(\mathsf{PP}, \boldsymbol{x}, \mu)$: On input public parameters PP, input $\boldsymbol{x}$, message μ, the encryption algorithm generates $\left(\{\mathsf{CT}_i^{\mathsf{st},0}\}, \{\mathsf{CT}_i^{\mathsf{rd},0}\}, \{\mathsf{CT}_i^{\mathsf{val},0}\}, \{\mathsf{CT}_{ijk}, \mathsf{CT}'_{ijk}\}_{\substack{i\in[n],j\in[m],\\ k\in[\log q]}}, \widehat{\mathsf{CT}}, \mathsf{CT}^*\right)$ in the same way as the read-only construction. Additionally, it also computes $\mathsf{CT}^{\mathsf{wt},0} = \boldsymbol{s}\mathbf{A}^{\mathsf{wt},0} + \widehat{\mathbf{e}}\mathbf{R}^{\mathsf{wt},0}$, encoding of the initial write address, where $\mathbf{R}^{\mathsf{wt},0} \leftarrow \{0,1\}^{m\times m}$.

  Let $\mathsf{ListVecCT}_0 = \left(\{\mathsf{CT}_{ijk}, \mathsf{CT}'_{ijk}\}_{\substack{i\in[n],j\in[m],\\ k\in[\log q]}}, \left\{\mathsf{CT}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{CT}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathsf{CT}_i^{\mathsf{val},0}\right\}_{i\in[\lambda]}\right)$. Output ciphertext

$$\mathsf{CT}_{\boldsymbol{x}} = \left(\widehat{\mathsf{CT}}, \mathsf{CT}^*, \mathsf{CT}^{\mathsf{wt},0}, \mathsf{ListVecCT}_0, \boldsymbol{x}\right)$$

- $\mathsf{Dec}(\mathsf{SK}_{P,D}, \mathsf{CT}_{\boldsymbol{x}})$: The decryption algorithm is the same as the read-only construction.

The correctness proof and parameters setting are similar to the counterpart of the read-only construction, thus we omit the details here.

Efficiency Analysis. The (space/time) complexity of our construction can be analyzed by the following aspects. The polynomial $n(\cdot,\cdot)$ denotes the lattice dimension.

- The public parameters contain $(L_{\mathsf{st}} + L_{\mathsf{rd}} + nmT + 1)$ random $n \times m$ matrices in $\mathbb{Z}_q$, which is $\tilde{O}(n(\lambda, T)^2 \cdot n^2 T^2)$ in bit complexity. The master secret key is one $m \times m$ matrix.
- The secret key for RAM program P and database D contains $T(N+T)$ small $m \times m$ matrices, which is $\tilde{O}(n(\lambda, T)^2 \cdot T(N+T))$ in bit complexity.
- The ciphertext for input $\boldsymbol{x}$ contains $(L_{\mathsf{st}} + L_{\mathsf{rd}} + nmT + \lambda)$ dimension-m vectors in $\mathbb{Z}_q$, which is $\tilde{O}(n(\lambda, T) \cdot \lambda n^2 T^2)$ in bit complexity.
- Decryption involves matrix-vector multiplication. The time complexity of decryption is $\tilde{O}(T)$.

Next, we show the decryption time in our ABE for full-fledged RAMs is independent of the length of database. Similarly, the essential algorithm StepEvalCT, which may be computed T times, in decryption algorithm can be divided into two steps, as analyzed below

- **Step circuit**: The runtime of CtEval with respect to step circuit C is a polynomial in $(\lambda, L_{\mathsf{st}}, L_{\mathsf{rd}})$. Observe that L_{st} is the length of the state, which is independent of the input length, and $L_{\mathsf{rd}} = \log N$. Thus, the runtime of CtEval is upper bounded by a polynomial in $(\lambda, L_{\mathsf{st}})$.
- **Recoding part**: In this step, we first compute CtEval with respect to the update circuit C^{up} and then do the recoding multiplication. This part is upper bounded by a polynomial in $(\lambda, T, L_{\mathsf{rd}})$.

From the above observation, it follows that that the runtime of the decryption algorithm is a polynomial in (λ, T), where the polynomial is independent of the length of the database.

Security Proof. The strategy of proving the security of this construction closely follows the proof of the prior one, i.e., using a sequence of hybrids:

- **Hybrid** Hyb_1: This corresponds to the real experiment.
- **Hybrid** Hyb_2: Hyb_2 is the same as Hyb_1 except that it uses $\mathsf{Sim.Setup}(1^\lambda, \boldsymbol{x}^*)$ to generate $\mathsf{Sim.PP}$.
- **Hybrid** $\mathsf{Hyb}_{3,i,j}$, for $\gamma < i$, the secret key for query (P_γ, D_γ) is simulated. For query (P_i, D_i), upto the j-th step, the step keys are simulated. For $\tau > j$, the step keys are normally generated. For query (P_γ, D_γ), where $\gamma > i$, the secret key is normally generated.
- **Hybrid** Hyb_4: Hyb_4 is the same as $\mathsf{Hyb}_{3,Q,T}$ except that the anchor public key $\mathbf{A}$ is sampled randomly from $\mathbb{Z}_q^{n\times m}$. In $\mathsf{Hyb}_{3,Q,T}$ the secret keys for all queries are simulated without using $\mathsf{MSK} = \mathbf{T}_\mathbf{A}$.
- **Hybrid** Hyb_5: Hyb_5 is the same as Hyb_4 except that it uses algorithm $\mathsf{Sim.Enc}$ to generate the challenge ciphertext.

However, the algorithms $(\mathsf{Sim.Setup}, \mathsf{Sim.Enc}, \mathsf{Sim.StepEvalPK})$ slightly differs from those used in the read-only setting. In the following, we mainly discuss the difference in these algorithms:

$\underline{\mathsf{Sim.Setup}(1^\lambda, \boldsymbol{x}^*)}$: The simulated setup algorithm generates

$$\left(\mathbf{A}, \left(\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right)_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}, \mathsf{Sim.ListMxPK}_0, \boldsymbol{u}\right)$$

in the same way as the read-only case. Additionally, it sets $\mathbf{A}^{\mathsf{wt},0} = \mathbf{A}\mathbf{R}^{\mathsf{wt},0}$, where $\mathbf{R}^{\mathsf{wt},0} \leftarrow \{0,1\}^{m\times m}$. Output $\mathsf{MSK} = (\mathsf{PP}, \mathbf{T}_\mathbf{A})$ and

$$\mathsf{Sim.PP} = \left(\mathbf{A}, \left(\mathbf{B}_{jk\ell}, \mathbf{B}'_{jk\ell}\right)_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}, \mathsf{Sim.ListMxPK}_0, \boldsymbol{u}, \mathbf{A}^{\mathsf{wt},0}\right)$$

$\underline{\mathsf{Sim.Enc}(\mathsf{Sim.PP}, \boldsymbol{x}^*, 1^{|\mu|}, (\mathbf{A}, \boldsymbol{u}), (\boldsymbol{b}, b'))}$: The simulated encryption algorithm generates

$$\left(\widehat{\mathsf{CT}},\ \mathsf{CT}^*,\ \mathsf{ListVecCT}_0, \left\{(\mathsf{CT}_{jk\ell}, \mathsf{CT}'_{jk\ell})\right\}_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}, \boldsymbol{x}^*\right)$$

in the same way as the read-only case. Additionally, it computes $\mathsf{CT}^{\mathsf{wt},0} = \boldsymbol{b}\mathbf{R}^{\mathsf{wt},0}$, where $\mathbf{R}^{\mathsf{wt},0}$ is generated in $\mathsf{Sim.Setup}$. Output challenge ciphertext

$$\mathsf{CT}_{\boldsymbol{x}^*} = \left(\widehat{\mathsf{CT}},\ \mathsf{CT}^*, \mathsf{CT}^{\mathsf{wt},0}\ \mathsf{ListVecCT}_0, \left\{(\mathsf{CT}_{jk\ell}, \mathsf{CT}'_{jk\ell})\right\}_{j\in[n],k\in[m],\ell\in\lceil\log q\rceil}, \boldsymbol{x}^*\right)$$

$\underline{\mathsf{Sim.StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})}$: The simulated algorithm does the following:

- Compute the step circuit as

  $$\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L`_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{wt},\tau}\right\}_{i\in[L_{\mathsf{wt}}]}, \mathbf{A}^{\mathsf{wb},\tau}\right) = \mathsf{PubEval}\left(\mathsf{Sim.ListMxPK}_{\tau-1}, C\right)$$

  Let $\mathbf{A}'_{N+\tau} + \mathbf{B}_{N+\tau} = \mathbf{A}^{\mathsf{wb},\tau}$.

- Compute the update circuit as

  $$\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(\{i, \mathbf{A}^{\mathsf{wt},i}\}_{i=1}^{\tau-1}, \{\mathbf{A}_i^{\mathsf{rd},\tau}\}_{i\in[L_{\mathsf{rd}}]}, C^{\mathsf{up}}\right)$$

  where $\mathbf{A}^{\mathsf{rd},\tau}$ encodes the actual read address rd^τ of execution $P^D(x^*)$ at the τ-th step.

- Sample $\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}^\tau} = \left(\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}^\tau,0}, \mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}_\tau,1}\right) \leftarrow \mathcal{D}_{Z^{m\times m}}$ and set $\mathbf{A}^{\mathsf{val},\tau} = \mathbf{A}\left(\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}^\tau,0} + \mathbf{R}^{\mathsf{rd},\tau}\mathbf{T}^{\mathsf{rd},\tau}_{\mathsf{rd}_\tau,1} + \mathbf{R}_i\right)$, where $\mathbf{R}^{\mathsf{rd},\tau} = \mathsf{TrapEval}\left(f \circ C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{ListMxTD}_{\tau-1}\right)$

- For $i \in [N + \tau - 1] - \{\mathsf{rd}^\tau\}$, compute $\mathbf{T}_i^{\mathsf{rd},\tau}$ as

  $$\mathbf{T}_i^{\mathsf{rd},\tau} \leftarrow \mathsf{SampleRight}\left(\mathbf{A}, (i - \mathsf{rd}_\tau)\mathbf{G}, \mathbf{R}^{\mathsf{rd},\tau}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^{\mathsf{val},\tau} - \mathbf{A}'_i - \mathbf{B}_i, s\right)$$

  such that

  $$\left[\mathbf{A}||\mathbf{A}\mathbf{R}^{\mathsf{rd},\tau} + (i - \mathsf{rd}_\tau)\mathbf{G}||\mathbf{A}'_i + \mathbf{B}_i + b\mathbf{G}\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + b\mathbf{G}$$

Set $\mathsf{Sim.StepKey}_\tau = \left\{\mathbf{T}_i^{\mathsf{rd},\tau}\right\}_{i=1}^{N+\tau-1}$. Output $(\mathsf{Sim.StepKey}_\tau, \mathsf{Sim.ListMxPK}_\tau)$.

The indistinguiability proofs between two adjacent hybrids are the same as those in the read-only setting, thus we omit the proofs.

4.5 Our Construction for the Dual Setting

For completeness, we first state the syntax of (key-policy) public-key attribute-based encryption for RAMs in the dual setting. It consists of a tuple of ppt algorithms $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with details as follows:

- **Setup,** $\mathsf{Setup}(1^\lambda, 1^T)$: On input security parameter λ and upper time bound T, setup algorithm outputs public parameters PP and master secret key MSK.
- **Key Generation,** $\mathsf{KeyGen}(\mathsf{MSK}, P)$: On input a master secret key MSK and a RAM program P, it outputs a secret key SK_P.
- **Encryption,** $\mathsf{Enc}(\mathsf{PP}, D, \mu)$: On input public parameters PP, a database D and a message μ, it outputs a ciphertext CT_D.
- **Decryption,** $\mathsf{Dec}(\mathsf{SK}_P, \mathsf{CT}_D)$: This is modeled as a RAM program. In particular, this algorithm will have random access to the binary representations of the key SK_P and the ciphertext CT_D. It outputs the corresponding plaintext μ if the decryption is successful; otherwise, it outputs $\perp$.

The correctness and security properties can be defined similarly as before, so we omit the definitions. Before presenting our construciton, we first provide a

detailed compasison with previous approaches addressing the dual setting ABE for RAMs.

4.5.1 Read-Only Case

In this part, we describe our ABE construction for read-only RAMs. The description below follows the same paradigm. The step circuit for read-only RAM is defined as

$$(\mathsf{st}^\tau, \mathsf{rd}^\tau) \leftarrow C(\mathsf{st}^{\tau-1}, \mathsf{rd}^{\tau-1}, b^\tau)$$

where st^τ denotes the state information at τ-th step, rd^τ denotes the read address at τ-th step and b^τ is the read value.

4.5.2 Description of StepEvalPK and StepEvalCT

Before proceeding to our ABE construction, we first describe the syntax of two following subroutines that are used in the construction. We use similar notation in the description below.

- $(\mathsf{StepKey}_\tau, \mathsf{ListMxPK}_\tau) \leftarrow \mathsf{StepEvalPK}(C, \mathsf{ListMxPK}_{\tau-1}, \mathsf{MSK}, \{\mathbf{A}_i\}_{i\in[N]}, \boldsymbol{u})$: On input the step circuit C, encoding matrices $\mathsf{ListMxPK}_{\tau-1}$ for the $(\tau-1)$-th state, read address, read value, master secret key and the encoding matrices for the database $\{\mathbf{A}_i\}_{i\in[N]}$, the key evaluation outputs the τ-th step key $\mathsf{StepKey}_\tau$ and encoding matrices $\mathsf{ListMxPK}_\tau$ for the τ-th step.
- $\mathsf{ListVecCT}_\tau \leftarrow \mathsf{StepEvalCT}(C, \mathsf{ListVecCT}_{\tau-1}, \{\mathsf{CT}_i\}_{i\in[N]}, \mathsf{StepKey}_\tau)$: On input the step circuit C, encoding $\mathsf{ListVecCT}_{\tau-1}$ of the $(\tau-1)$-th state, read address,

read value, database and the τ-th attribute key, the ciphertext evaluation outputs the encoding $\mathsf{ListVecCT}_\tau$ of the τ-th state, read address and read value.

In the following description, we set function $f : \{0,1\}^{L_{\mathsf{st}}} \to \mathbb{Z}$ to be $f\left(\{x_i\}_{i=1}^{L_{\mathsf{st}}}\right) = \sum x_i \cdot 2^i$. The construction of $\mathsf{StepEvalPK}$ and $\mathsf{StepEvalCT}$ with respect to step circuit C are as follows:

$\underline{\mathsf{StepEvalPK}\,(C, \mathsf{ListMxPK}_{\tau-1}, \mathbf{A}, \mathbf{T}_{\mathbf{A}}, \{\mathbf{A}_i\}_{i\in[N]}, \boldsymbol{u})}$: the key evaluation algorithm does the following:

- Parse the encoding matrices $\mathsf{ListMxPK}_{\tau-1}$ as

$$\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau-1}\right\}_{i\in[L_{\mathsf{st}}]},\ \left\{\mathbf{A}_i^{\mathsf{rd},\tau-1}\right\}_{i\in[L_{\mathsf{rd}}]},\ \mathbf{A}^{\mathsf{val},\tau-1}\right)$$

- Compute $\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right) = \mathsf{PubEval}\,(\mathsf{ListMxPK}_{\tau-1}, C)$.
- Sample $\mathbf{A}^{\mathsf{val},\tau} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$. For $i \in [N]$, compute $\mathbf{T}_i^{\mathsf{rd},\tau}$ as

$$\mathbf{T}_i^{\mathsf{rd},\tau} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G}, \mathbf{A}^{\mathsf{val},\tau} - \mathbf{A}_i, s)$$

  where $\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(f, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right)$, such that

$$\left[\mathbf{A}||\mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G}||\mathbf{A}_i + b\mathbf{G}\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + b\mathbf{G}$$

- Compute $\boldsymbol{t}^{\mathsf{st},\tau}$ as

$$\boldsymbol{t}^{\mathsf{st},\tau} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{A}_1^{\mathsf{st},\tau}, \boldsymbol{u}, s)$$

  such that

$$\left[\mathbf{A}||\mathbf{A}_1^{\mathsf{st},\tau}\right] \cdot \boldsymbol{t}^{\mathsf{st},\tau} = \boldsymbol{u}$$

Set $\mathsf{StepKey}_\tau = \left(\left\{\mathbf{T}_i^{\mathsf{rd},\tau}\right\}_{i\in[N]}, \boldsymbol{t}^{\mathsf{st},\tau}\right)$. Output $(\mathsf{StepKey}_\tau, \mathsf{ListMxPK}_\tau)$.

$\underline{\mathsf{StepEvalCT}}\,(C,\ \mathsf{ListVecCT}_{\tau-1},\ \{\mathsf{CT}_i\}_{i\in[N]},\ \mathsf{StepKey}_\tau)$: the ciphertext evaluation algorithm does the following:

- Parse the ciphertext $\mathsf{ListVecCT}_{\tau-1}$ as

$$\left(\left\{\mathsf{CT}_i^{\mathsf{st},\tau-1}\right\}_{i\in[L_{\mathsf{st}}]},\ \left\{\mathsf{CT}_i^{\mathsf{rd},\tau-1}\right\}_{i\in[L_{\mathsf{rd}}]},\ \mathsf{CT}^{\mathsf{val},\tau-1}\right)$$

  along with its associated value $\mathsf{ListST}_{\tau-1} = \left(\{\mathsf{st}^{\tau-1}\}_{i\in[L_{\mathsf{st}}]}, \{\mathsf{rd}^{\tau-1}\}_{i\in[L_{\mathsf{rd}}]}, \mathsf{val}^{\tau-1}\right)$.
- Compute $\left(\left\{\mathsf{CT}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]},\ \left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right) = \mathsf{CtEval}\,(\mathsf{ListMxPK}_{\tau-1}, \mathsf{ListST}_{\tau-1}, C)$.
- If $\mathsf{st}_i^\tau = 0$, then terminate the $\mathsf{StepEvalCT}$ execution. Otherwise, compute

$$\mathsf{CT}^{\mathsf{val},\tau} = \left(\widehat{\mathsf{CT}}, \mathsf{CT}^{\mathsf{rd},\tau}, \mathsf{CT}_{\mathsf{rd}^\tau}\right)\begin{pmatrix}\mathbf{T}_{\mathsf{rd}^\tau}^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix}$$

  where $\mathsf{CT}^{\mathsf{rd},\tau} = \mathsf{CtEval}\left(\left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \{\mathsf{rd}_i^\tau\}_{i\in[L_{\mathsf{rd}}]}, f\right)$.

Output $\mathsf{ListVecCT}_\tau = (\mathsf{CT}^{\mathsf{st},\tau}, \mathsf{CT}^{\mathsf{rd},\tau}, \mathsf{CT}^{\mathsf{val},\tau})$.

Construction. In our construction below, we assume the initial states are all 1, the initial read address is always the first index of database and initial read value is 1. We also assume that if the first bit of state is 0, then the RAM program terminates.

Our read-only ABE for RAMs construction $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ can be described as follows:

Setup, $\mathsf{Setup}(1^\lambda, 1^N, T)$: On input security parameter λ, database maximum length N and time bound T, the setup algorithm computes:

- $(\mathbf{A}, \mathbf{T}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(1^n, 1^q, m)$, the anchor matrix and its associated trapdoor.
- $\forall i \in [L_{\mathsf{st}}]$, sample $\mathbf{A}_i^{\mathsf{st},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the initial state.
- $\forall i \in [L_{\mathsf{rd}}]$, sample $\mathbf{A}_i^{\mathsf{rd},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the initial read address.
- $\forall i \in [N]$, sample $\mathbf{A}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the database.
- Sample $\mathbf{A}^{\mathsf{val},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, encoding matrix for the initial read value.
- Sample $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$, encoding vector for the plaintext.

Let $\mathsf{ListMxPK}_0 = \left(\left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathbf{A}^{\mathsf{val},0}\right)$, denoted as the public keys for the initial step. Output $\mathsf{PP} = (\mathbf{A}, \{\mathbf{A}_i\}_{i\in[N]}, \mathsf{ListMxPK}_0, \boldsymbol{u})$ and $\mathsf{MSK} = (\mathsf{PP}, \mathbf{T}_\mathbf{A})$.

Key Generation, $\mathsf{KeyGen}(\mathsf{MSK}, P)$: On input master secret key MSK and RAM program P with step circuit C. For $\tau \in [T]$, do the following

- Compute $\mathsf{StepEvalPK}$ for the τ-th step,

$$\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st},\tau}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd},\tau}]}, \mathbf{A}^{\mathsf{val},\tau}, \left\{\mathbf{T}_i^{\mathsf{rd},\tau}\right\}_{i\in[N]}, \boldsymbol{t}^{\mathsf{st},\tau}\right)$$
$$\leftarrow \mathsf{StepEvalPK}\left(C, \mathsf{ListMxPK}_{\tau-1}, \{\mathbf{A}_i\}_{i\in[N]}, \mathbf{A}, \mathbf{T}_\mathbf{A}, \boldsymbol{u}\right)$$

- Set $\mathsf{ListMxPK}_\tau$ and $\mathsf{StepKey}_\tau$ as

$$\mathsf{ListMxPK}_\tau = \left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st},\tau}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd},\tau}]}, \mathbf{A}^{\mathsf{val},\tau}\right), \quad \mathsf{StepKey}_\tau = \left(\left\{\mathbf{T}_i^{\mathsf{rd},\tau}\right\}_{i\in[N]}, \boldsymbol{t}^{\mathsf{st},\tau}\right)$$

Output $\mathsf{SK}_P = (P, \{\mathsf{StepKey}_\tau\}_{\tau\in[T]})$.

Encryption, $\mathsf{Enc}(\mathsf{PP}, D, \mu)$: On input public parameters PP, database $D = \{D_i\}_{i=1}^N$, message μ, the encryption algorithm does the following:

- Sample vector $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$ and error vectors $\widehat{\mathbf{e}}, \mathbf{e}^*$ from Gaussian distribution $\mathcal{D}_{\mathbb{Z}^m}$.
- $\forall i \in [L_{\mathsf{st}}]$, compute $\mathsf{CT}_i^{\mathsf{st},0} = \boldsymbol{s}\left(\mathbf{A}_i^{\mathsf{st},0} + \mathbf{G}\right) + \widehat{\mathbf{e}}\mathbf{R}_i^{\mathsf{st},0}$, encoding of the initial state, where $\mathbf{R}_i^{\mathsf{st},0} \leftarrow \{0,1\}^{m \times m}$.
- $\forall i \in [L_{\mathsf{rd}}]$, compute $\mathsf{CT}_i^{\mathsf{rd},0} = \boldsymbol{s}\left(\mathbf{A}_i^{\mathsf{rd},0} + \mathsf{rd}_i^0\mathbf{G}\right) + \widehat{\mathbf{e}}\mathbf{R}_i^{\mathsf{rd},0}$, encoding of the initial read address, where $\mathbf{R}_i^{\mathsf{rd},0} \leftarrow \{0,1\}^{m \times m}$ and $\{\mathsf{rd}_i^0\}_{i \in [L_{\mathsf{rd}}]}$ is the bit representation of 1.
- Compute $\mathsf{CT}^{\mathsf{val},0} = \boldsymbol{s}\left(\mathbf{A}^{\mathsf{val},0} + \mathbf{G}\right) + \widehat{\mathbf{e}}\mathbf{R}_i^{\mathsf{val},0}$, encoding of the initial read value, where $\mathbf{R}_i^{\mathsf{val},0} \leftarrow \{0,1\}^{m \times m}$.
- $\forall i \in [N]$, compute $\mathsf{CT}_i = \boldsymbol{s}(\mathbf{A}_i + D_i\mathbf{G}) + \widehat{\mathbf{e}}\mathbf{R}_i$, encoding of the database, where $\mathbf{R}_i \leftarrow \{0,1\}^{m \times m}$.
- Compute $\widehat{\mathsf{CT}} = \boldsymbol{s}\mathbf{A} + \widehat{\mathbf{e}}$ and $\mathsf{CT}^* = \boldsymbol{s}\boldsymbol{u}^\mathsf{T} + \mu\lceil q/2 \rceil + e^*$.

Let $\mathsf{ListVecCT}_0 = \left(\left\{\mathsf{CT}_i^{\mathsf{st},0}\right\}_{i \in [L_{\mathsf{st}}]}, \left\{\mathsf{CT}_i^{\mathsf{rd},0}\right\}_{i \in [L_{\mathsf{rd}}]}, \mathsf{CT}^{\mathsf{val},0}\right)$. Output ciphertext

$$\mathsf{CT}_D = \left(\widehat{\mathsf{CT}}, \mathsf{CT}^*, \mathsf{ListVecCT}_0, \{\mathsf{CT}_i\}_{i \in [N]}, D\right)$$

Decryption, $\mathsf{Dec}(\mathsf{SK}_P, \mathsf{CT}_D)$: On input secret key $\mathsf{SK}_P = (P, \{\mathsf{StepKey}_\tau\}_{\tau \in [T]})$, ciphertext CT_D, the decryption algorithm outputs $\perp$ if $P^D \neq 0$. Otherwise, let C denote the step circuit associated with P and do the following: for $\tau \in [t]$, where t is the input-specific running time of P^D.

- Compute $\mathsf{StepEvalCT}$ for the τ^{th} step,

$$\left(\left\{\mathsf{CT}_i^{\mathsf{st},\tau}\right\}_{i \in [L_{\mathsf{st}}]}, \left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i \in [L_{\mathsf{rd}}]}, \left\{\mathsf{CT}^{\mathsf{val},\tau}\right\}\right) \leftarrow \mathsf{StepEvalCT}\left(C, \mathsf{ListVecCT}_{\tau-1}, \{\mathsf{CT}_i\}_{i \in [N]}, \mathsf{StepKey}_\tau\right)$$

  If $\forall i \in [L_{\mathsf{st}}]$, $\mathsf{st}_i = 0$, then terminate the execution.

- Set $\mathsf{ListVecCT}_\tau = \left(\left\{\mathsf{CT}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st},\tau}]}, \left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd},\tau}]}, \mathsf{CT}^{\mathsf{val},\tau}\right)$.

Check if $\left\|\left([\widehat{\mathsf{CT}}\|\mathsf{CT}_1^{\mathsf{st},t}]\cdot(\boldsymbol{t}^{\mathsf{st},t})^{\mathsf{T}}\right)-\mathsf{CT}^*\right\|_\infty < q/4$ and if so, output 0, else output 1.

4.5.3 Analysis of Correctness, Efficiency and Parameters

In this part, we show that the ABE construction described above is correct (c.f. Definition 4.2.1), then analysis decryption time and set lattice parameters afterwards.

Lemma 4.5.1. *The ABE construction for read-only RAMs satisfies correctness as defined in Definition 4.2.1 (with appropriate modification).*

Proof. Let the ciphertext be CT_D and secret key be SK_P, such that $P^D = 0$. At the τ-th step, by evaluating the ciphertext using algorithm $\mathsf{StepEvalCT}$ with respect to the step circuit, we have $\left\{\mathsf{CT}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}$, $\left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}$ are encryption of state and read address at the τ-th step respectively. Unfolding ciphertext $\mathsf{CT}^{\mathsf{val},\tau}$ (ignoring the error terms), we obtain

$$\begin{aligned}\mathsf{CT}^{\mathsf{val},\tau} &= \left(\widehat{\mathsf{CT}}, \mathsf{CT}^{\mathsf{rd},\tau}, \mathsf{CT}_k\right)\begin{pmatrix}\mathbf{T}_k^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix}\\ &\approx \boldsymbol{s}\left[\mathbf{A}\|\mathbf{A}^{\mathsf{rd},\tau}+i\mathbf{G}\|\mathbf{A}_i+b\mathbf{G}\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix}\\ &\approx \mathbf{A}^{\mathsf{val},\tau}+b\mathbf{G}\end{aligned}$$

Thus, ciphertext $\mathsf{CT}^{\mathsf{val},\tau}$ encodes the read value of database at rd^τ index, which can be used in the next step evaluation.

Suppose at step t, where $t \leq T$, we have $P^D = 0$, then $\mathsf{CT}_1^{\mathsf{st},t}$ encrypts state value 0. Thus,

$$\begin{aligned}\left([\widehat{\mathsf{CT}}\|\mathsf{CT}_1^{\mathsf{st},t}] \cdot \boldsymbol{t}^{\mathsf{st},t}\right) - \mathsf{CT}^* &= \boldsymbol{s}\left[\mathbf{A}\|\mathbf{A}_1^{\mathsf{st},\tau}\right] \cdot (\boldsymbol{t}^{\mathsf{st},t})^{\mathsf{T}} + \mathbf{e}^t - \mathsf{CT}^* \\ &= \mathbf{e}^t - \mu\lceil q/2\rceil - \mathbf{e}^*\end{aligned}$$

By setting parameters appropriately as follows, our ABE construction is correct.

□

Parameters Setting. If the step circuit being evaluated has length d, then the noise in ciphertext grows in the worst case by a factor of $O(m^d)$. Thus, to support a RAM program with maximum running time T (the unit of time corresponds to one step), we set (n, m, q) as

- Lattice dimension n is an integer such that $n \geq (Td \log n)^{1/\epsilon}$, for some fixed $0 < \epsilon < 1/2$.
- Modulus q is set to be $q = 2^{n^\epsilon}$, since the noise in the ciphertexts grows by a factor of $O(m^{Td})$. Hence, we need q to be on the order of $\Omega(Bm^{Td})$, where $B = O(n)$ is the maximum magnitude of noise (from discrete Gaussian distribution) added during encryption. To ensure correctness of decryption and hardness of LWE, we set $q = 2^{n^\epsilon}$.
- Lattice column parameter m is set to be $m = \Theta(n \log q)$ to make the leftover hash lemma hold.

The parameter s used in algorithms SampleLeft and SampleRight are set as $s > \sqrt{n \log q} \cdot \omega(\sqrt{\log m})$, as required by Lemma 2.3.7.

For security we rely on the hardness of the LWE problem, which requires that the ratio q/B is not too large, where $B = O(n)$ is the maximum magnitude of noise

(from discrete Gaussian distribution) added during encryption. In particular, the underlying problem is believed to be hard even when q/B is 2^{n^ϵ}.

Efficiency Analysis. The (space/time) complexity of our construction can be analyzed by the following aspects. The polynomial $n(\cdot,\cdot)$ denotes the lattice dimension.

- The public parameters contain $(L_{\mathsf{st}} + L_{\mathsf{rd}} + N + 2)$ random $n \times m$ matrices in $\mathbb{Z}_q$, which is $\tilde{O}(n(\lambda, T)^2 \cdot N)$ in bit complexity. The master secret key is one $m \times m$ matrix.
- The secret key for RAM program P contains $T(N+1)$ small $m \times m$ matrices, which is $\tilde{O}(n(\lambda, T)^2 \cdot NT)$ in bit complexity.
- The ciphertext for database with length N contains $(L_{\mathsf{st}} + L_{\mathsf{rd}} + N + 2)$ dimension-m vectors in $\mathbb{Z}_q$, which is $\tilde{O}(n(\lambda, T) \cdot N)$ in bit complexity.
- Suppose the running time of P^D is t, and decryption involves matrix-vector multiplication. The time complexity of decryption is $\tilde{O}(n(\lambda, T)^2 \cdot t)$.

Next, we would like to show the following: if a program P on input database D takes time at most T then correspondingly, the decryption of secret key for P on input an encryption of message x associated with attribute database D takes time $p(\lambda, T)$, for a fixed polynomial p.

We analyze the time to decrypt an encryption of database D associated with message μ using a key of RAM program with runtime bounded by T. The essential algorithm StepEvalCT, which may be computed T times, in decryption algorithm can be divided into two steps, as analyzed below

- **Step circuit**: The runtime of CtEval with respect to step circuit C is a polynomial in $(\lambda, L_{\mathsf{st}}, L_{\mathsf{rd}})$. Observe that L_{st} is the length of the state, which is independent of the input length, and $L_{\mathsf{rd}} = \log N$. Thus, the runtime of CtEval is upper bounded by a polynomial in $(\lambda, L_{\mathsf{st}})$.
- **Recoding part**: In this step, we compute CtEval with respect to the gadget circuit f and then the recoding multiplication. This part is upper bounded by a polynomial in $(\lambda, L_{\mathsf{rd}})$.

From the above observations, it follows that the runtime of the decryption algorithm is a polynomial in (λ, T), where the polynomial is independent of the length of the database. In particular, notice that if T is polylogarithmic in the input length then the decryption time is sub-linear in the input length.

4.5.4 Security Proof

In this part, we show the security of our ABE for read-only RAM construction, assuming the hardness of LWE assumption. We first describe algorithms $(\mathsf{Sim.Setup}, \mathsf{Sim.Enc}, \mathsf{Sim.StepEvalPK})$ in the following:

- $\mathsf{Sim.Setup}$ produces "programmed" public parameters. That is, every pubic matrix produced as part of algorithm $\mathsf{Sim.Setup}$ has hardwired in it, a bit of the challenge ciphertext, initial state, read address, etc.
- $\mathsf{Sim.Enc}$ produces a simulated encryption of the message.
- $\mathsf{Sim.StepEvalPK}$ takes as input the $(\tau - 1)$-th layer of simulated public keys $\mathsf{Sim.ListMxPK}_{\tau-1}$ and produces the τ-th layer of simulated public keys $\mathsf{Sim.ListMxPK}_{\tau}$ and τ-th layer of step keys $\mathsf{Sim.StepKey}_{\tau}$.

These simulated algorithms can be constructed as follows:

$\underline{\mathsf{Sim.Setup}}(1^\lambda, D^*)$: On input the challenge database D^*, the simulated setup algorithm does:

- Computes $(\mathbf{A}, \mathbf{T}_\mathbf{A}) \leftarrow \mathsf{TrapGen}(1^n, 1^q, m)$ and sample $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$.
- $\forall i \in [L_{\mathsf{st}}]$, set $\mathbf{A}_i^{\mathsf{st},0} = \mathbf{A}\mathbf{R}_i^{\mathsf{st},0} - \mathbf{G}$, where $\mathbf{R}_i^{\mathsf{st},0} \leftarrow \{0,1\}^{m\times m}$.
- $\forall i \in [L_{\mathsf{rd}}]$, set $\mathbf{A}_i^{\mathsf{rd},0} = \mathbf{A}\mathbf{R}_i^{\mathsf{rd},0} - \mathbf{G}$, where $\mathbf{R}_i^{\mathsf{rd},0} \leftarrow \{0,1\}^{m\times m}$.
- $\forall i \in [N]$, set $\mathbf{A}_i = \mathbf{A}\mathbf{R}_i - D^*[i] \cdot \mathbf{G}$, where $\mathbf{R}_i \leftarrow \{0,1\}^{m\times m}$.
- Set $\mathbf{A}^{\mathsf{val},0} = \mathbf{A}\mathbf{R}^{\mathsf{val},0} - \mathbf{G}$, where $\mathbf{R}^{\mathsf{val},0} \leftarrow \{0,1\}^{m\times m}$.

Let $\mathsf{Sim.ListMxPK}_0 = \left(\left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathbf{A}^{\mathsf{val},0}\right)$, denoted as the public keys for the initial step, and $\mathsf{ListMxTD}_0 = \left(\left\{\mathbf{R}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{R}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathbf{R}^{\mathsf{val},0}\right)$, denoted as the trapdoor matrix for initial step. Output $\mathsf{Sim.PP} = (\mathbf{A}, \{\mathbf{A}_i\}_{i\in[N]}, \mathsf{Sim.ListMxPK}_0, \boldsymbol{u})$ and $\mathsf{MSK} = (\mathsf{PP}, \mathbf{T}_\mathbf{A})$.

$\underline{\mathsf{Sim.Enc}(\mathsf{Sim.PP}, D^*, 1^{|\mu|}, (\mathbf{A}, \boldsymbol{u}), (\boldsymbol{b}, b'))}$: On input simulated public parameters $\mathsf{Sim.PP}$, challenge database D^* and message length $|\mu|$ and LWE instance $((\mathbf{A}, \boldsymbol{u}), (\boldsymbol{b}, b'))$, the simulated encryption algorithm does

- $\forall i \in [L_{\mathsf{st},0}]$, compute $\mathsf{CT}_i^{\mathsf{st},0} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{st},0}$, where $\mathbf{R}_i^{\mathsf{st},0}$ is generated in $\mathsf{Sim.Setup}$.
- $\forall i \in [L_{\mathsf{rd},0}]$, compute $\mathsf{CT}_i^{\mathsf{rd},0} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{rd},0}$, where $\mathbf{R}_i^{\mathsf{rd},0}$ is generated in $\mathsf{Sim.Setup}$.
- Compute $\mathsf{CT}^{\mathsf{val},0} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{val},0}$, where $\mathbf{R}_i^{\mathsf{val},0}$ is generated in $\mathsf{Sim.Setup}$.
- $\forall i \in [N]$, compute $\mathsf{CT}_i = \boldsymbol{b}\mathbf{R}_i$, where $\mathbf{R}_i$ is generated in $\mathsf{Sim.Setup}$.
- Set $\widehat{\mathsf{CT}} = \boldsymbol{b}$ and $\mathsf{CT}^* = b'$.

Output challenge ciphertext $\mathsf{CT}_{D^*} = \left(\widehat{\mathsf{CT}},\ \mathsf{CT}^*,\ \mathsf{ListVecCT}_0, \{\mathsf{CT}_i\}_{i\in[N]}, D^*\right)$.

$\underline{\mathsf{Sim.StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})}$: On input the step circuit C of program P satisfying $P^{D^*} = 1$, simulated $(\tau-1)$-th layer of simulated public keys $\mathsf{Sim.ListMxPK}_{\tau-1}$ and simulated public parameters $\mathsf{Sim.PP}$, it does

- Compute $\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right) = \mathsf{PubEval}\left(\mathsf{Sim.ListMxPK}_{\tau-1}, C\right)$, and then $\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(f, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right)$, where $\mathbf{A}^{\mathsf{rd},\tau}$ encodes the actual read address rd^τ of P^{D^*} at step τ.
- Sample $\mathbf{T}_{\mathsf{rd}^\tau}^{\mathsf{rd},\tau} = \left(\mathbf{T}_{\mathsf{rd}^\tau,0}^{\mathsf{rd},\tau}, \mathbf{T}_{\mathsf{rd}_\tau,1}^{\mathsf{rd},\tau}\right) \leftarrow \mathcal{D}_{Z^{m\times m}}$ and set $\mathbf{A}^{\mathsf{val},\tau} = \mathbf{A}\left(\mathbf{T}_{\mathsf{rd}^\tau,0}^{\mathsf{rd},\tau} + \mathbf{R}^{\mathsf{rd},\tau} + \mathbf{R}_i\right)$, where $\mathbf{R}^{\mathsf{rd},\tau} = \mathsf{TrapEval}\left(f \circ C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{ListMxTD}_{\tau-1}\right)$
- For $i \in [N] - \{\mathsf{rd}^\tau\}$, compute $\mathbf{T}_i^{\mathsf{rd},\tau}$ as

$$\mathbf{T}_i^{\mathsf{rd},\tau} \leftarrow \mathsf{SampleRight}\left(\mathbf{A}, (i-\mathsf{rd}_\tau)\mathbf{G}, \mathbf{R}^{\mathsf{rd},\tau}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^{\mathsf{val},\tau} - \mathbf{A}_i, s\right)$$

  such that

$$\left[\mathbf{A}||\mathbf{A}\mathbf{R}^{\mathsf{rd},\tau} + (i-\mathsf{rd}_\tau)\mathbf{G}||\mathbf{A}_i + b\mathbf{G}\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + b\mathbf{G}$$

- As $P^{D^*} = 1$, so $\mathbf{A}_1^{\mathsf{st},\tau}$ is the encoding of 1, i.e., $\mathbf{A}_1^{\mathsf{st},\tau} = \mathbf{A}\mathbf{R}_1^{\mathsf{st},\tau} - \mathbf{G}$. Compute $\boldsymbol{t}^{\mathsf{st},\tau}$ as

$$\boldsymbol{t}^{\mathsf{st},\tau} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}_1^{\mathsf{st},\tau}, \mathbf{T}_{\mathbf{G}}, \boldsymbol{u}, s)$$

  such that

$$\left[\mathbf{A}||\mathbf{A}\mathbf{R}_1^{\mathsf{st},\tau} - \mathbf{G}\right]\cdot \boldsymbol{t}^{\mathsf{st},\tau} = \boldsymbol{u}$$

Set $\mathsf{Sim.StepKey}_\tau = \left(\{\mathbf{T}_i^{\mathsf{rd},\tau}\}_{i\in[N]}, \boldsymbol{t}^{\mathsf{st},\tau}\right)$. Output $(\mathsf{Sim.StepKey}_\tau, \mathsf{Sim.ListMxPK}_\tau)$.

Theorem 4.5.2. *Assuming the hardness of LWE assumption (with parameters as specified above), our ABE construction is secure (c.f. Definition 4.2.5).*

Proof. Let Q be the number of key queries made by the adversary. We first describe a sequence of hybrids in the following:

Hybrid Hyb_1: This corresponds to the real experiment:

- $\mathcal{A}$ specifies challenge attribute database D^* and message μ.
- Challenger computes $\mathsf{Setup}(1^\lambda)$ to obtain the public parameters PP and secret key MSK. Then challenger generates the challenge ciphertext $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{PP}, D^*, \mu)$. It sends CT^* and PP to $\mathcal{A}$.
- For $\gamma \in [Q]$, adversary $\mathcal{A}$ specifies the programs P_γ such that $P_\gamma^{D^*} = 1$. Challenger generates the attribute keys for P_i, for $\gamma \in [Q]$, $\mathsf{SK}_{P_\gamma} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, P_\gamma)$.
- Let b be the output of adversary. Output b.

Hybrid Hyb_2: Hyb_2 is the same as Hyb_1 except that it uses $\mathsf{Sim.Setup}(1^\lambda, D^*)$ to generate $\mathsf{Sim.PP}$.

Hybrid $\{\mathsf{Hyb}_{3,i,j}\}_{i\in[Q],j\in[T]}$: Simply put, in hybrid $\mathsf{Hyb}_{3,i,j}$, for $\gamma < i$, the secret key for query P_γ is simulated. For query P_i, upto the j-th step, the step keys are simulated. For $\tau > j$, the step keys are normally generated. For query P_γ, where $\gamma > i$, the secret key is normally generated. We describe it in details below:

- Adversary specifies challenge attribute database D^* and message μ.
- Challenger computes $\mathsf{Setup}(1^\lambda)$ to obtain the public parameters PP and secret key MSK. Then challenger generates the challenge ciphertext $\mathsf{CT}^* \leftarrow \mathsf{Enc}(\mathsf{PP}, D^*, \mu)$. It sends CT^* and PP to $\mathcal{A}$.

- For $\gamma \in [Q]$, adversary $\mathcal{A}$ specifies the programs P_γ such that $P_\gamma^{D^*} = 1$. Challenger generates the secret key $\mathsf{SK}_{P_\gamma} = (P_\gamma, \{\mathsf{StepKey}_\tau\}_{\tau\in[T]})$ for P_γ as follows:
  - For $\gamma < i$, answer secret key query P_γ as
    1. For every $\tau \in [T]$, compute

       $$(\mathsf{Sim.ListMxPK}_\tau, \mathsf{Sim.StepKey}_\tau) \leftarrow \mathsf{Sim.StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})$$

    2. Set $\mathsf{SK}_\gamma = (\{\mathsf{Sim.StepKey}_\tau\}_{\tau\in[T]})$.
  - For $\gamma = i$, answer secret key query P_i as
    1. For $\tau < j$, generate

       $$(\mathsf{Sim.ListMxPK}_\tau, \mathsf{Sim.StepKey}_\tau) \leftarrow \mathsf{Sim.StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})$$

    2. For $\tau \geq j$, generate

       $$(\mathsf{ListMxPK}_\tau, \mathsf{StepKey}_\tau) \leftarrow \mathsf{StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})$$

    Set $\mathsf{SK}_i = (\{\mathsf{Sim.StepKey}_\tau\}_{\tau<i}, \{\mathsf{StepKey}_\tau\}_{\tau\geq i})$.
  - For $\gamma > i$, answer secret key query P_γ as
    1. For every $\tau \in [T]$, generate

       $$(\mathsf{ListMxPK}_\tau, \mathsf{StepKey}_\tau) \leftarrow \mathsf{StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})$$

    2. Set $\mathsf{SK}_\gamma = (\{\mathsf{StepKey}_\tau\}_{\tau\in[T]})$.

Hybrid Hyb_4: Hyb_4 is the same as $\mathsf{Hyb}_{3,Q,T}$ except that the anchor public key $\mathbf{A}$ is sampled randomly from $\mathbb{Z}_q^{n\times m}$. In $\mathsf{Hyb}_{3,Q,T}$ the secret keys for all queries are simulated without using $\mathsf{MSK} = \mathbf{T_A}$.

Hybrid Hyb_5: Hyb_5 is the same as Hyb_4 except that it uses algorithm $\mathsf{Sim.Enc}$ to generate the challenge ciphertext.

Lemma 4.5.3. *By Leftover Hash Lemma 5.2.2, the output distributions of hybrids* Hyb_1 *and* Hyb_2 *are statistically close.*

Proof. The only difference between hybrid Hyb_1 and Hyb_2 is in hybrid Hyb_2, the public parameters are generated using $\mathsf{Sim.Setup}(1^\lambda, D^*)$, where they are set like this: $\mathbf{AR}_i - \mathsf{val} \cdot \mathbf{G}$ for $\mathbf{R}_i \xleftarrow{\$} \{0,1\}^{m \times m}$ and $\mathsf{val} \in \{0,1\}$. We point out that one has to be careful when applying Leftover Hash Lemma here, as the anchor matrix $\mathbf{A}$ is only statistically close to uniform, and it is generated together with $\mathbf{T_A}$. However, by Markov chain property, we can first sample $\mathbf{A}$, and then sample $\mathbf{T_A}$ and $\mathbf{AR}_i$ independently from the marginals. Therefore, since $(\mathbf{A}, \mathbf{AR}_i)$ is statistically close to uniform when $\mathbf{A}$ is uniform, it also holds true when $\mathbf{A}$ is only statistically close to uniform and when $\mathbf{T_A}$ is known as well. Thus we have that the output distributions of hybrids Hyb_1 and Hyb_2 are statistically close. □

Lemma 4.5.4. *The output distributions of hybrids* Hyb_2 *and* $\mathsf{Hyb}_{3,1,0}$ *are identical.*

Proof. As described above, hybrid $\mathsf{Hyb}_{3,1,0}$ is obtained by using $\mathsf{Sim.Setup}$ to simulate public parameters, containing $\mathsf{Sim.ListMxPK}_0$, and generating all secret keys normally. Therefore, we have that the output distributions of hybrids Hyb_2 and $\mathsf{Hyb}_{3,1,0}$ are identical. □

Lemma 4.5.5. *By properties of sampling algorithms (Lemma 2.3.7 and 2.3.8) and Leftover Hash Lemma 5.2.2, the output distributions of hybrids* $\mathsf{Hyb}_{3,i,j}$ *and* $\mathsf{Hyb}_{3,i,j+1}$ *are statistically close.*

Proof. The main difference between hybrid $\mathsf{Hyb}_{3,i,j}$ and $\mathsf{Hyb}_{3,i,j+1}$ is the generation of temporary matrices $\mathsf{ListMxPK}_{j+1}$ and $\mathsf{StepKey}_j$, which is in the SK_i. In $\mathsf{Hyb}_{3,i,j+1}$, we use algorithm $\mathsf{Sim.StepEvalPK}$ to generate $\mathsf{Sim.ListMxPK}_{j+1}, \mathsf{Sim.StepKey}_j$.

In $\mathsf{Sim.StepKey}_j$, matrices $\{\mathbf{T}_i^{\mathsf{rd},\tau}\}_{i\in[N]}$ are generated either using algorithm SampleRight or sampling from Gaussian distribution $\mathcal{D}$. By Lemma 2.3.7, we obtain that the distribution of $\mathsf{Sim.StepKey}_j$ is statistically close to $\mathsf{StepKey}_j$.

Next, we discuss the temporary public matrices. The only difference between $\mathsf{ListMxPK}_{j+1}$ and $\mathsf{Sim.ListMxPK}_{j+1}$ is the generation of $\mathbf{A}^{\mathsf{val},j}$. Instead of sampling $\mathbf{A}^{\mathsf{val},\tau}$ from random, in $\mathsf{Sim.ListMxPK}_{j+1}$, we set $\mathbf{A}^{\mathsf{val},j} = \mathbf{A}\left(\mathbf{T}_{\mathsf{rd}_\tau,0}^{\mathsf{rd},j} + \mathbf{R}^{\mathsf{rd},j} + \mathbf{R}_i\right)$, where $\mathbf{T}_{\mathsf{rd}_\tau,0}^{\mathsf{rd},j}, \mathbf{R}^{\mathsf{rd},j}, \mathbf{R}_i$ comes from discrete Gaussian distribution. Thus by lemma 2.3.8, we have that the distribution of $\mathsf{Sim.ListMxPK}_{j+1}$ is statistically close to $\mathsf{ListMxPK}_{j+1}$.

Combing the two components, we conclude that output distributions of hybrids $\mathsf{Hyb}_{3,i,j}$ and $\mathsf{Hyb}_{3,i,j+1}$ are statistically close. □

Lemma 4.5.6. *The output distributions of hybrids* $\mathsf{Hyb}_{3,i,T}$ *and* $\mathsf{Hyb}_{3,i+1,1}$ *are identical.*

Proof. As described above, the only difference between hybrid $\mathsf{Hyb}_{3,i,T}$ and $\mathsf{Hyb}_{3,i+1,1}$ is that in $\mathsf{Hyb}_{3,i+1,1}$ the public matrices for initial step, $\mathsf{Sim.ListMxPK}_0$, is simulated. As $\mathsf{Sim.ListMxPK}_0$ does not exist in SK_{i+1}, we have the output distributions of hybrids $\mathsf{Hyb}_{3,i,T}$ and $\mathsf{Hyb}_{3,i+1,1}$ are identical. □

Lemma 4.5.7. *By Lemma 2.3.6, the output distributions of hybrids* $\mathsf{Hyb}_{3,Q,T}$ *and* Hyb_4 *are statistically close.*

Proof. The only difference between hybrid $\mathsf{Hyb}_{3,Q,T}$ and Hyb_4 is that in Hyb_4, the anchor public matrix $\mathbf{A}$ is sampled from uniform distribution over $\mathbb{Z}_q^{n\times m}$ instead of using algorithm TrapGen. By Lemma 2.3.6, since TrapGen sampled $\mathbf{A}$ which is statistically close to uniform distribution, we conclude that the output distributions of hybrids $\mathsf{Hyb}_{3,Q,T}$ and Hyb_4 are statistically close. □

Lemma 4.5.8. *Assuming the hardness of LWE assumption, the output distributions of hybrids* Hyb_4 *and* Hyb_5 *are computationally close.*

Proof. The only difference between hybrid Hyb_4 and Hyb_5 is the generation of challenge ciphertext. In Hyb_5, the challenge ciphertext is generated using algorithm $\mathsf{Sim.Enc}$. The simulated ciphertext is simulated using LWE instance. By unfolding one component of ciphertext,

$$\mathsf{CT}_i^{\mathsf{st},0,} = \boldsymbol{b}\mathbf{R}_i^{\mathsf{st},0} = (\boldsymbol{s}\mathbf{A} + \boldsymbol{e})\mathbf{R}_i^{\mathsf{st},0}$$

which is of the same form of normally generated ciphertext. By the hardnss of LWE assumption, we have that the distribution of$(\widehat{\mathsf{CT}}, \mathsf{CT}^*)$ in simulated ciphertext is computationally close to it normally generated counterpart. Also by Leftover Hash Lemma 5.2.2, the distribution of $\left(\mathsf{PP}, \mathsf{CT}^{D^*} - \{\widehat{\mathsf{CT}}, \mathsf{CT}^*\}\right)$ is statistically close to $\left(\mathsf{Sim.PP}, \mathsf{Sim.CT}_{D^*} - \{\widehat{\mathsf{CT}}, \mathsf{CT}^*\}\right)$.

There we conclude that the output distributions of hybrids Hyb_4 and Hyb_5 are computationally close. □

Combining the hybrids and lemmas proved above, we prove that our ABE construction for read-only RAMs is secure, as defined in Definition 4.2.5. □

4.5.5 Handling Write Operations

In this part, we show how to construct ABE for a full-fledged RAM. The step circuit C here is defined as

$$(\mathsf{st}_i, \mathsf{addr}_i^{\mathsf{wt}}, b_i^{\mathsf{wt}}, \mathsf{addr}_i^{\mathsf{rd}}) \leftarrow C(\mathsf{st}_{i-1}, \mathsf{addr}_{i-1}^{\mathsf{rd}}, b_{i-1}^{\mathsf{rd}})$$

One additional parameter we use here, $L_{\mathsf{wt}} = \log N$ for write address bit-length. Then we define an array for write operations upto step τ as $\mathsf{trace}^{\tau} = \{(i, \mathsf{wt}^i)\}_{i=1}^{\tau-1}$, namely it records all the write operations from the first step to the $(\tau-1)$-th step. We also define an auxiliary circuit $C^{\mathsf{up}}(\tau, \mathsf{trace}^{\tau}, \mathsf{rd}^{\tau})$ as

- $C^{\mathsf{up}}(\tau, \mathsf{trace}^{\tau}, \mathsf{rd}^{\tau})$: On input the step index τ, the write trace trace^{τ} and read address $\mathsf{rd}^{\tau} \in [N]$, the update circuit outputs the correct address $\widehat{\mathsf{rd}}^{\tau} \in [N + \tau - 1]$ to read.

The intuition of handling write operations is: The database here is an append-only data structure. Instead of updating the database after each write, we write the newly computed value to a new location. For instance, at the τ-th step, the value is writed at the end of the append-only data structure, i.e., the $(N + \tau)$-th location. For each read operation, we first compute the update circuit $C^{\mathsf{up}}(\tau, \mathsf{trace}_{\tau}, \mathsf{rd}_{\tau})$ to obtain the correct address to read.

Description of StepEvalPK **and** StepEvalCT**.** We revise the construction of algorithms StepEvalPK and StepEvalCT as follows:

$\underline{\mathsf{StepEvalPK}\,(C, \mathsf{ListMxPK}_{\tau-1}, \mathbf{A}, \mathbf{T}_{\mathbf{A}}, \{\mathbf{A}_i\}_{i\in[N]}, \boldsymbol{u})}$: The key evaluation algorithm does:

- Parse the encoding matrices $\mathsf{ListMxPK}_{\tau-1}$ as

$$\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau-1}\right\}_{i\in[L_{\mathsf{st}}]},\ \left\{\mathbf{A}_i^{\mathsf{rd},\tau-1}\right\}_{i\in[L_{\mathsf{rd}}]},\ \mathbf{A}^{\mathsf{val},\tau-1}\right)$$

- Compute the step circuit as

$$\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{wt},\tau}\right\}_{i\in[L_{\mathsf{wt}}]}, \mathbf{A}^{\mathsf{wb},\tau}\right) = \mathsf{PubEval}\,(\mathsf{ListMxPK}_{\tau-1}, C)$$

  Let $\mathbf{A}_{N+\tau} = \mathbf{A}^{\mathsf{wb},\tau}$.

- Compute the update circuit as

$$\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(\{i, \mathbf{A}^{\mathsf{wt},i}\}_{i=1}^{\tau-1}, \{\mathbf{A}_i^{\mathsf{rd},\tau}\}_{i\in[L_{\mathsf{rd}}]}, C^{\mathsf{up}}\right)$$

- Sample $\mathbf{A}^{\mathsf{val},\tau} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$. For $i \in [N+\tau-1]$, compute $\mathbf{T}_i^{\mathsf{rd},\tau}$ as

$$\mathbf{T}_i^{\mathsf{rd},\tau} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G}, \mathbf{A}^{\mathsf{val},\tau} - \mathbf{A}_i, s)$$

  where $\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(f, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}\right)$, such that

$$\left[\mathbf{A}||\mathbf{A}^{\mathsf{rd},\tau} + i\mathbf{G}||\mathbf{A}_i + b\mathbf{G}\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + b\mathbf{G}$$

- Compute $\mathbf{T}^{\mathsf{st},\tau}$ as

$$\mathbf{T}^{\mathsf{st},\tau} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{A}_1^{\mathsf{st},\tau}, \boldsymbol{u}, s)$$

  such that

$$\left[\mathbf{A}||\mathbf{A}_1^{\mathsf{st},\tau}\right] \cdot \boldsymbol{t}^{\mathsf{st},\tau} = \boldsymbol{u}$$

Set $\mathsf{StepKey}_\tau = \left(\{\mathbf{T}_i^{\mathsf{rd},\tau}\}_{i=1}^{N+\tau-1}, \boldsymbol{t}^{\mathsf{st},\tau}\right)$. Output $(\mathsf{StepKey}_\tau, \mathsf{ListMxPK}_\tau)$.

$\underline{\mathsf{StepEvalCT}\,(C,\ \mathsf{ListVecCT}_{\tau-1},\ \{\mathsf{CT}_i\}_{i\in[N+\tau-1]},\ \mathsf{StepKey}_\tau)}$: the ciphertext evaluation algorithm does the following:

- Parse the ciphertext $\mathsf{ListVecCT}_{\tau-1}$ as

$$\left(\left\{\mathsf{CT}_i^{\mathsf{st},\tau-1}\right\}_{i\in[L_{\mathsf{st}}]},\ \left\{\mathsf{CT}_i^{\mathsf{rd},\tau-1}\right\}_{i\in[L_{\mathsf{rd}}]},\ \mathsf{CT}^{\mathsf{val},\tau-1}\right)$$

  along with its associated value $\mathsf{ListST}_{\tau-1} = \left(\{\mathsf{st}^{\tau-1}\}_{i\in[L_{\mathsf{st}}]}, \{\mathsf{rd}^{\tau-1}\}_{i\in[L_{\mathsf{rd}}]}, \mathsf{val}^{\tau-1}\right)$.

- Compute

$$\left(\left\{\mathsf{CT}_i^{\mathsf{st},\tau}\right\}_{i\in[L_{\mathsf{st}}]},\ \left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathsf{CT}_i^{\mathsf{wt},\tau}\right\}_{i\in[L_{\mathsf{wt}}]}, \mathsf{CT}^{\mathsf{wb},\tau}\right) = \mathsf{CtEval}\,(\mathsf{ListMxPK}_{\tau-1}, \mathsf{ListST}_{\tau-1}, C)$$

- If $\mathsf{st}_i^\tau = 0$, then terminate the StepEvalCT execution. Otherwise, compute

$$\mathsf{CT}^{\mathsf{rd},\tau} = \mathsf{CtEval}(\{i, \mathsf{CT}^{\mathsf{wt},i}\}_{i=0}^{\tau-1}, \left\{\mathsf{CT}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, C^{\mathsf{up}})$$

  along with the actual read address $\mathsf{rd}^\tau \in [N + \tau - 1]$.

- Choose the corresponding $\mathbf{T}_{\mathsf{rd}^\tau}^{\mathsf{rd},\tau}$ and compute

$$\mathsf{CT}^{\mathsf{val},\tau} = \left(\widehat{\mathsf{CT}}, \mathsf{CT}^{\mathsf{rd},\tau}, \mathsf{CT}_{\mathsf{rd}^\tau}\right) \begin{pmatrix} \mathbf{T}_{\mathsf{rd}^\tau}^{\mathsf{rd},\tau} \\ \mathbf{I} \end{pmatrix}$$

Output $\mathsf{ListVecCT}_\tau = (\mathsf{CT}^{\mathsf{st},\tau}, \mathsf{CT}^{\mathsf{rd},\tau}, \mathsf{CT}^{\mathsf{val},\tau})$.

Construction, Efficiency Analysis and Security Proof. The construction of ABE for full-fledged RAMs based on StepEvalPK and StepEvalCT is similar to the counterpart of read-only RAMs. We sketch the construction below:

- $\mathsf{Setup}(1^\lambda, 1^N, T)$: On input security parameter λ, maximum database size N and time bound T, the setup algorithm generates $\left(\mathbf{A}, \left\{\mathbf{A}_i^{\mathsf{st},0}\right\}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}, \{\mathbf{A}_i\}, \mathbf{A}^{\mathsf{val},0}, \boldsymbol{u}\right)$ in the same way as the read-only case. Additionally, it also samples random matrix $\mathbf{A}^{\mathsf{wt},0} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, the encoding matrix for initial write address.

  Let $\mathsf{ListMxPK}_0 = \left(\left\{\mathbf{A}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathbf{A}^{\mathsf{val},0}\right)$, denoted as the public keys for the initial step. Output $\mathsf{PP} = \left(\mathbf{A}, \{\mathbf{A}_i\}_{i\in[N]}, \mathbf{A}^{\mathsf{wt},0}, \mathsf{ListMxPK}_0, \boldsymbol{u}\right)$ and $\mathsf{MSK} = (\mathsf{PP}, \mathbf{T}_{\mathbf{A}})$.

- $\mathsf{KeyGen}(\mathsf{MSK}, P)$: The key generation algorithm is the same as the read-only construction.

- $\mathsf{Enc}(\mathsf{PP}, D, \mu)$: On input public parameters PP, database $D = \{D_i\}_{i=1}^N$, message μ, the encryption algorithm generates $\left(\{\mathsf{CT}_i^{\mathsf{st},0}\}, \{\mathsf{CT}_i^{\mathsf{rd},0}\}, \{\mathsf{CT}^{\mathsf{val},0}\}, \{\mathsf{CT}_i\}, \widehat{\mathsf{CT}}, \mathsf{CT}^*\right)$

in the same way as the read-only construction. Additionally, it also computes $\mathsf{CT}^{\mathsf{wt},0} = \boldsymbol{s}\left(\mathbf{A}^{\mathsf{wt},0} + \mathbf{G}\right) + \widehat{\mathbf{e}}\mathbf{R}^{\mathsf{wt},0}$, encoding of the initial write address, where $\mathbf{R}^{\mathsf{wt},0} \leftarrow \{0,1\}^{m\times m}$.

Let $\mathsf{ListVecCT}_0 = \left(\left\{\mathsf{CT}_i^{\mathsf{st},0}\right\}_{i\in[L_{\mathsf{st}}]}, \left\{\mathsf{CT}_i^{\mathsf{rd},0}\right\}_{i\in[L_{\mathsf{rd}}]}, \mathsf{CT}^{\mathsf{val},0}\right)$. Output ciphertext $\mathsf{CT}_D = \left(\widehat{\mathsf{CT}}, \mathsf{CT}^*, \mathsf{CT}^{\mathsf{wt},0}, \mathsf{ListVecCT}_0, \{\mathsf{CT}_i\}_{i\in[N]}, D\right)$.

- $\mathsf{Dec}(\mathsf{SK}_P, \mathsf{CT}_D)$: The decryption algorithm is the same as the read-only construction.

The correctness proof and parameters setting are similar to the counterpart of the read-only construction, thus we omit the details here.

Efficiency Analysis. The (space/time) complexity of our construction can be analyzed by the following aspects. The polynomial $n(\cdot,\cdot)$ denotes the lattice dimension.

- The public parameters contain $(L_{\mathsf{st}} + L_{\mathsf{rd}} + N + 3)$ random $n \times m$ matrices in $\mathbb{Z}_q$, which is $\tilde{O}(n(\lambda, T)^2 \cdot N)$ in bit complexity. The master secret key is one $m \times m$ matrix.
- The secret key for RAM program P contains $T(N+T)$ small $m\times m$ matrices, which is $\tilde{O}(n(\lambda, T)^2 \cdot T(N+T))$ in bit complexity.
- The ciphertext for database with length N contains $(L_{\mathsf{st}} + L_{\mathsf{rd}} + N + 3)$ dimension-m vectors in $\mathbb{Z}_q$, which is $\tilde{O}(n(\lambda, T) \cdot N)$ in bit complexity.
- Suppose the running time of P^D is t, and decryption involves matrix-vector multiplication. The time complexity of decryption is $\tilde{O}(n(\lambda, T)^2 \cdot t)$.

Next, we show the decryption time in our ABE for full-fledged RAMs is independent of the length of database. Similarly, the essential algorithm StepEvalCT,

which may be computed T times, in decryption algorithm can be divided into two steps, as analyzed below

- **Step circuit**: The runtime of CtEval with respect to step circuit C is a polynomial in $(\lambda, L_{\mathsf{st}}, L_{\mathsf{rd}})$. Observe that L_{st} is the length of the state, which is independent of the input length, and $L_{\mathsf{rd}} = \log N$. Thus, the runtime of CtEval is upper bounded by a polynomial in $(\lambda, L_{\mathsf{st}})$.
- **Recoding part**: In this step, we first compute CtEval with respect to the update circuit C^{up} and then do the recoding multiplication. This part is upper bounded by a polynomial in $(\lambda, T, L_{\mathsf{rd}})$.

From the above observation, it follows that that the runtime of the decryption algorithm is a polynomial in (λ, T), where the polynomial is independent of the length of the database.

Security Proof. The strategy of proving the security of this construction closely follows the proof of the prior one, i.e., using a sequence of hybrids:

- **Hybrid** Hyb_1: This corresponds to the real experiment.
- **Hybrid** Hyb_2: Hyb_2 is the same as Hyb_1 except that it uses $\mathsf{Sim.Setup}(1^\lambda, D^*)$ to generate $\mathsf{Sim.PP}$.
- **Hybrid** $\mathsf{Hyb}_{3,i,j}$, for $\gamma < i$, the secret key for query P_γ is simulated. For query P_i, upto the j-th step, the step keys are simulated. For $\tau > j$, the step keys are normally generated. For query P_γ, where $\gamma > i$, the secret key is normally generated.

- **Hybrid** Hyb_4: Hyb_4 is the same as $\mathsf{Hyb}_{3,Q,T}$ except that the anchor public key $\mathbf{A}$ is sampled randomly from $\mathbb{Z}_q^{n\times m}$. In $\mathsf{Hyb}_{3,Q,T}$ the secret keys for all queries are simulated without using $\mathsf{MSK} = \mathbf{T}_\mathbf{A}$.
- **Hybrid** Hyb_5: Hyb_5 is the same as Hyb_4 except that it uses algorithm $\mathsf{Sim.Enc}$ to generate the challenge ciphertext.

However, the algorithms ($\mathsf{Sim.Setup}, \mathsf{Sim.Enc}, \mathsf{Sim.StepEvalPK}$) slightly differs from those used in the read-only setting. In the following, we mainly discuss the difference in these algorithms:

- $\mathsf{Sim.Setup}(1^\lambda, D^*)$: The simulated setup algorithm generates $(\mathbf{A}, \{\mathbf{A}_i\}_{i\in[N]}, \mathsf{Sim.ListMxPK}_0, \boldsymbol{u})$ in the same way as the read-only case. Additionally, it sets $\mathbf{A}^{\mathsf{wt},0} = \mathbf{A}\mathbf{R}^{\mathsf{wt},0} - \mathbf{G}$, where $\mathbf{R}^{\mathsf{wt},0} \leftarrow \{0,1\}^{m\times m}$.

  Output $\mathsf{Sim.PP} = \left(\mathbf{A}, \{\mathbf{A}_i\}_{i\in[N]}, \mathbf{A}^{\mathsf{wt},0}, \mathsf{Sim.ListMxPK}_0, \boldsymbol{u}\right)$ and $\mathsf{MSK} = (\mathsf{PP}, \mathbf{T}_\mathbf{A})$.
- $\mathsf{Sim.Enc}(\mathsf{Sim.PP}, D^*, 1^{|\mu|}, (\mathbf{A}, \boldsymbol{u}), (\boldsymbol{b}, b'))$: The simulated encryption algorithm generates $(\widehat{\mathsf{CT}},\ \mathsf{CT}^*,$
  $\mathsf{ListVecCT}_0, \{\mathsf{CT}_i\}_{i\in[N]}, D^*)$ in the same way as the read-only case. Additionally, it computes $\mathsf{CT}^{\mathsf{wt},0} = \boldsymbol{b}\mathbf{R}^{\mathsf{wt},0}$, where $\mathbf{R}^{\mathsf{wt},0}$ is generated in $\mathsf{Sim.Setup}$.

  Output challenge ciphertext $\mathsf{CT}_{D^*} = \left(\widehat{\mathsf{CT}},\ \mathsf{CT}^*, \mathsf{CT}^{\mathsf{wt},0}, \mathsf{ListVecCT}_0, \{\mathsf{CT}_i\}_{i\in[N]}, D^*\right)$.
- $\mathsf{Sim.StepEvalPK}(C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{Sim.PP})$: The simulated algorithm does the following:

  – Compute the step circuit as

  $$\left(\left\{\mathbf{A}_i^{\mathsf{st},\tau}\right\}_{i\in[L\text{`}_{\mathsf{st}}]}, \left\{\mathbf{A}_i^{\mathsf{rd},\tau}\right\}_{i\in[L_{\mathsf{rd}}]}, \left\{\mathbf{A}_i^{\mathsf{wt},\tau}\right\}_{i\in[L_{\mathsf{wt}}]}, \mathbf{A}^{\mathsf{wb},\tau}\right) = \mathsf{PubEval}\left(\mathsf{Sim.ListMxPK}_{\tau-1}, C\right)$$

  Let $\mathbf{A}_{N+\tau} = \mathbf{A}^{\mathsf{wb},\tau}$.

– Compute the update circuit as

$$\mathbf{A}^{\mathsf{rd},\tau} = \mathsf{PubEval}\left(\{i, \mathbf{A}^{\mathsf{wt},i}\}_{i=1}^{\tau-1}, \{\mathbf{A}_i^{\mathsf{rd},\tau}\}_{i\in[L_{\mathsf{rd}}]}, C^{\mathsf{up}}\right)$$

where $\mathbf{A}^{\mathsf{rd},\tau}$ encodes the actual read address rd^τ of execution P^{D^*} at the τ-th step.

– Sample $\mathbf{T}_{\mathsf{rd}^\tau}^{\mathsf{rd},\tau} = \left(\mathbf{T}_{\mathsf{rd}^\tau,0}^{\mathsf{rd},\tau}, \mathbf{T}_{\mathsf{rd}_\tau,1}^{\mathsf{rd},\tau}\right) \leftarrow \mathcal{D}_{Z^{m\times m}}$ and set $\mathbf{A}^{\mathsf{val},\tau} = \mathbf{A}\left(\mathbf{T}_{\mathsf{rd}^\tau,0}^{\mathsf{rd},\tau} + \mathbf{R}^{\mathsf{rd},\tau} + \mathbf{R}_i\right)$, where $\mathbf{R}^{\mathsf{rd},\tau} = \mathsf{TrapEval}\left(f \circ C, \mathsf{Sim.ListMxPK}_{\tau-1}, \mathsf{ListMxTD}_{\tau-1}\right)$

– For $i \in [N+\tau-1] - \{\mathsf{rd}^\tau\}$, compute $\mathbf{T}_i^{\mathsf{rd},\tau}$ as

$$\mathbf{T}_i^{\mathsf{rd},\tau} \leftarrow \mathsf{SampleRight}\left(\mathbf{A}, (i-\mathsf{rd}_\tau)\mathbf{G}, \mathbf{R}^{\mathsf{rd},\tau}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}^{\mathsf{val},\tau} - \mathbf{A}_i, s\right)$$

such that

$$\left[\mathbf{A}||\mathbf{A}\mathbf{R}^{\mathsf{rd},\tau} + (i-\mathsf{rd}_\tau)\mathbf{G}||\mathbf{A}_i + b\mathbf{G}\right]\begin{pmatrix}\mathbf{T}_i^{\mathsf{rd},\tau}\\ \mathbf{I}\end{pmatrix} = \mathbf{A}^{\mathsf{val},\tau} + b\mathbf{G}$$

– As $P^{D^*} = 1$, so $\mathbf{A}_1^{\mathsf{st},\tau}$ is the encoding of 1, i.e., $\mathbf{A}_1^{\mathsf{st},\tau} = \mathbf{A}\mathbf{R}_1^{\mathsf{st},\tau} - \mathbf{G}$. Compute $\boldsymbol{t}^{\mathsf{st},\tau}$ as

$$\boldsymbol{t}^{\mathsf{st},\tau} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}_1^{\mathsf{st},\tau}, \mathbf{T}_{\mathbf{G}}, \boldsymbol{u}, s)$$

such that

$$\left[\mathbf{A}||\mathbf{A}\mathbf{R}_1^{\mathsf{st},\tau} - \mathbf{G}\right]\cdot \boldsymbol{t}^{\mathsf{st},\tau} = \boldsymbol{u}$$

Set $\mathsf{Sim.StepKey}_\tau = \left(\{\mathbf{T}_i^{\mathsf{rd},\tau}\}_{i=1}^{N+\tau-1}, \boldsymbol{t}^{\mathsf{st},\tau}\right)$. Output $\left(\mathsf{Sim.StepKey}_\tau, \mathsf{Sim.ListMxPK}_\tau\right)$.

The indistinguiability proofs between two adjacent hybrids are the same as those in the read-only setting, thus we omit the proofs.

CHAPTER 5

SYMBOLIC PROOFS FOR LATTICE-BASED CRYPTOGRAPHY

5.1 Technical Overview

The technical core of our contributions [19] are a set of (semi-)decision procedures for checking deducibility in the theory of Diffie-Hellman exponentiation, in its standard, bilinear and multilinear versions, and in the theories of fields, non-commutative rings, and matrices. In particular, we give decision procedures for checking deducibility in the theory of Diffie-Hellman exponentiation. This procedure has immediate applications to reasoning about security of cryptographic constructions based on bilinear and multilinear maps. The central idea behind our algorithm is to transform a deducibility problem into a problem from commutative algebra. The latter can be resolved through standard computations of Gröbner basis. Furthermore, we give a semi-decision procedure for checking deducibility in the theory of matrices. This has immediate applications to reasoning about security of lattice-based constructions. In this case, our algorithm extracts from a deducibility question a problem from non-commutative algebra. The problem can be resolved through semi-decision procedures based on non-commutative variants of Gröbner bases known as Subalgebra Analog of Gröbner Basis on Ideals (SAGBI) [103].

5.2 Example: Dual Regev Encryption

In this section, we describe an example public-key encryption scheme and show how it will be encoded in our formal system. We provide some mathematical background in Section 5.5.2. Recall that a public-key cryptosystem is given by three probabilistic algorithms ($\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}$) for generating keys, encryption, and decryption, such that with overwhelming probability, decryption is the inverse of encryption for valid key pairs.

We consider the Dual Regev Encryption scheme [75], an optimization of Regev's original encryption [112]. We focus on a simple version that encrypts single bits; however, standard techniques can be used to encrypt longer messages.

Definition 5.2.1 (Dual Regev Encryption)**.** *Below, let $\lambda = n$ be the security parameter, $m = O(n \log q)$, $q = O(m)$ and χ (or χ^n) be discrete Gaussian distribution over $\mathbb{Z}$ (or $\mathbb{Z}^n$).*

- *The key generation algorithm, $\mathsf{KeyGen}(1^\lambda)$, chooses a uniformly sampled random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a vector $\boldsymbol{r} \in \{-1, 1\}^m$ sampled uniformly, interpreted as a vector in $\mathbb{Z}_q^m$. The public key is $\mathsf{PK} = (\mathbf{A}, \boldsymbol{u})$, where $\boldsymbol{u} = \mathbf{A}\boldsymbol{r}$, and the secret key is $\mathsf{SK} = \boldsymbol{r}$.*
- *To encrypt a message $b \in \{0, 1\}$, the encryption algorithm $\mathsf{Enc}(\mathsf{PK}, b)$ chooses a random vector $\boldsymbol{s} \in \mathbb{Z}_q^n$, a vector $\boldsymbol{x}_0$ sampled from χ^n and an integer x_1 sampled from χ. The ciphertext consists of the vector $\boldsymbol{c}_0 = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{x}_0^\mathsf{T}$ and the integer $c_1 = \boldsymbol{s}^\mathsf{T}\boldsymbol{u} + x_1 + b\lceil q/2 \rceil$, where T denotes the transpose operation on matrices.*
- *The decryption algorithm checks whether the value $c_1 - \langle \boldsymbol{r}, \boldsymbol{c}_0 \rangle$ is closer to 0 or $b\lceil q/2 \rceil$ modulo p, and returns 0 in the first case, and 1 in the second.*

Decryption is correct with overwhelming probability, since we compute that $c_1 - \langle \boldsymbol{r}, \boldsymbol{c}_0 \rangle = x_1 + b\lceil q/2 \rceil - \langle \boldsymbol{r}, \boldsymbol{x}_0 \rangle$, so the norm of the term $x_1 - \langle \boldsymbol{r}, \boldsymbol{x}_0 \rangle$ will be much smaller than $b\lceil q/2 \rceil$.

Gentry, Peikert and Vaikuntanathan [75] show that Dual Regev Encryption achieves chosen-plaintext indistinguishability under the *decisional LWE assumption*, defined below. Traditionally, chosen-plaintext indistinguishability is modeled by a probabilistic experiment, where an adversary proposes two messages m_0 and m_1, and is challenged with a ciphertext c^* corresponding to an encryption of message m_b, where b is sampled uniformly at random. The adversary is then requested to return a bit b'. The winning condition for the experiment is $b = b'$, which models that the adversary guesses the bit b correctly. Formally, one defines the advantage of an adversary $\mathcal{A}$ against chosen-plaintext security as:

$$\mathsf{Adv}^{\mathsf{cpa}}_{\mathcal{A}} = \left| \mathsf{Pr} Gb = b' - \frac{1}{2} \right|$$

where G is the probabilistic experiment that models chosen-plaintext security and $\frac{1}{2}$ represents the probability that a trivial adversary which flips a coin b' at random guesses the bit b correctly. We note that in our case, since the message space is $\{0, 1\}$, we can wlog set $m_0 = 0$ and $m_1 = 1$; thus, the adversary only needs to be queried once in this experiment.

The formal definition of G, instantiated to Dual Regev Encryption, is shown in Figure 5.1. We inline the key generation and encryption subroutines. In line 1, the public key $(\mathbf{A}, \boldsymbol{u})$ and its associated secret key $\boldsymbol{r}$ are randomly sampled. In lines 2 and 3, the message bit b is sampled uniformly, and the ciphertext (c_0, c_1) of this message is generated. Finally, in line 4, the adversary outputs a bit b', given as input the public key and the ciphertext.

Now, we outline the hardness assumptions and lemmas used in the proof of Dual Regev Encryption.

Leftover Hash Lemma. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be a collection of m samples of uniform vectors from $\mathbb{Z}_q^n$. The Leftover Hash Lemma (LHL) states that, given enough samples, the result of multiplying $\mathbf{A}$ with a random $\{-1, 1\}$-valued matrix $\mathbf{R}$ is statistically close to uniform. Additionally, this result holds in the presence of an arbitrary linear leakage of the elements of $\mathbf{R}$. Specifically, the following leftover hash lemma is proved in [2] (Lemma 13).

Lemma 5.2.2 (Leftover Hash Lemma)**.** *Let q, n, m be as in Definition 5.2.1. Let k be a polynomial of n. Then, the distributions $\{(\mathbf{A}, \mathbf{AR}, \mathbf{R}^\mathsf{T}\mathbf{w})\}$ $\{(\mathbf{A}, \mathbf{B}, \mathbf{R}^\mathsf{T}\mathbf{w})\}$ are negligibly close in n, where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ in both distributions, $\mathbf{R} \xleftarrow{\$} \{0, 1\}^{m \times k}$, $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n \times k}$, and $\mathbf{w} \in \mathbb{Z}_q^m$ is any arbitrary vector.*

Given the above, security of Dual Regev Encryption is stated as follows:

Proposition 5.2.3 ([75])**.** *For any adversary $\mathcal{A}$ against chosen-plaintext security of Dual Regev Encryption, there exists an adversary $\mathcal{B}$ against LWE, such that:*

- $\mathsf{Adv}^{\mathsf{cpa}}_{\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{lwe}}_{\mathcal{B}} + \epsilon_{LHL}$;
- $t_{\mathcal{A}} \approx t_{\mathcal{B}}$;

where $\mathsf{Adv}^{\mathsf{lwe}}_{\mathcal{B}}$ denotes the advantage of $\mathcal{B}$ against decisional LWE problem, ϵ_{LHL} is a function of the scheme parameters determined by the Leftover Hash Lemma, and $t_{\mathcal{A}}$ and $t_{\mathcal{B}}$ respectively denote the execution time of $\mathcal{A}$ and $\mathcal{B}$.

Game $G^{\mathsf{pke}}_{\mathsf{org}}$:

$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \boldsymbol{r} \xleftarrow{\$} \{-1,1\}^m;$

let $\boldsymbol{u} = \mathbf{A}\boldsymbol{r}$;

$b \xleftarrow{\$} \{0,1\}, \boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n, \boldsymbol{x}_0 \xleftarrow{\$} \mathcal{D}_{\mathbb{Z}^m}, x_1 \xleftarrow{\$} \mathcal{D}_{\mathbb{Z}};$

let $\boldsymbol{c}_0 = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{x}_0, c_1 = \boldsymbol{s}^\mathsf{T}\boldsymbol{u} + x_1 + b\lceil q/2\rceil;$

$b' \leftarrow \mathcal{A}(\mathbf{A}, \boldsymbol{u}, \boldsymbol{c}_0, c_1);$

Figure 5.1: IND-CPA security of dual-Regev PKE.

Security proof We now outline the proof of Proposition 5.2.3.

The proof proceeds with a series of *game transformations*, beginning with the game in Figure 5.1. The goal is to transform the game into one in which the adversary's advantage is obviously zero. Each transformation is justified semantically either by semantic identities or by probabilistic assertions, such as the LWE assumption; in the latter case, the transformation incurs some error probability which must be recorded.

The first transformation performs an information-theoretic step based on the Leftover Hash Lemma. The Leftover Hash Lemma allows us to transform the joint distribution $(\mathbf{A}, \mathbf{A}\boldsymbol{r})$ (where $\mathbf{A}$ and $\boldsymbol{r}$ are independently randomly sampled) into the distribution $(\mathbf{A}, \boldsymbol{u})$ (where $\boldsymbol{u}$ is a fresh, uniformly sampled variable). (This invocation does not use the linear leakage $\boldsymbol{w}$ from Lemma 5.2.2). In order to apply this lemma, we *factor* the security game from Figure 5.1 into one which makes use of $\boldsymbol{A}$ and $\boldsymbol{u}$, but not $\boldsymbol{r}$. That is, if G_0 is the original security game, then we have factored G into

$$G_0 = G'\{\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m};\ \boldsymbol{r} \xleftarrow{\$} \{-1,1\}^m;\ \mathsf{let}\ \boldsymbol{u} = \mathbf{A}\boldsymbol{r}\}_p,$$

where $G'\{\cdot\}_p$ is a game context with a hole at position p, such that G' does not make reference to $\boldsymbol{r}$ except in the definition of $\boldsymbol{u}$. By the Leftover Hash Lemma,

we may now move to the game:

$$G_1 = G'\{\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m};\ \boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n\}_p.$$

This transformation effectively removes $\boldsymbol{r}$ from the security game, thus removing any contribution of the secret key $\boldsymbol{r}$ to the information gained by the adversary $\mathcal{A}$. This transformation incurs the error probability ϵ_{LHL}. The resultant game is shown in Figure 5.2.

Game G_2 :
$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n;$
$b \xleftarrow{\$} \{0,1\}, \boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n, \boldsymbol{x}_0 \xleftarrow{\$} \mathcal{D}_{\mathbb{Z}^m}, x_1 \xleftarrow{\$} \mathcal{D}_{\mathbb{Z}};$
let $\boldsymbol{c}_0 = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{x}_0, c_1 = \boldsymbol{s}^\mathsf{T}\boldsymbol{u} + x_1 + b\lceil q/2\rceil;$
$b' \leftarrow \mathcal{A}(\mathbf{A}, \boldsymbol{u}, \boldsymbol{c}_0, c_1);$

Figure 5.2: Dual-Regev PKE: Game 2

The second transformation performs a reduction step based on the LWE assumption. Indeed, note that after the first transformation, the ciphertexts $(\boldsymbol{c}_0, c_1)$ contain an LWE distribution of dimension $n\times(m+1)$, with the message bit added to c_1. By applying LWE, we then may safely transform $\boldsymbol{c}_0$ to be uniformly random, and c_1 to be uniformly random added to to the message bit. The resulting security game is shown in Figure 5.3.

Game G_3 :
$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n;$
$b \xleftarrow{\$} \{0,1\}, \boldsymbol{r_0} \xleftarrow{\$} \mathbb{Z}_q^m, r_1 \xleftarrow{\$} \mathbb{Z}_q;$
let $\boldsymbol{c}_0 = \boldsymbol{r_0}, c_1 = r_1 + b\lceil q/2\rceil;$
$b' \leftarrow \mathcal{A}(\mathbf{A}, \boldsymbol{u}, \boldsymbol{c}_0, c_1);$

Figure 5.3: Dual-Regev PKE: Game 3

The next transformation applies a semantics-preserving transformation known as *optimistic sampling*. To remove the message bit from the adversary input, note that the term c_1 is equal to the sum of r_1 and $b\lceil q/2 \rceil$, where r_1 is uniformly sampled and does not appear anywhere else in the game. Because of this, we know that c_1 itself is uniformly random. Thus, we can safely rewrite the body of c_1 to be equal to a fresh uniformly sampled r_1. The resulting game is shown in Figure 5.4.

Game G_4 :
$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}, \boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n;$
$b \xleftarrow{\$} \{0,1\}, \boldsymbol{r_0} \xleftarrow{\$} \mathbb{Z}_q^m, r_1 \xleftarrow{\$} \mathbb{Z}_q;$
let $\boldsymbol{c}_0 = \boldsymbol{r_0}, c_1 = r_1;$
$b' \leftarrow \mathcal{A}(\mathbf{A}, \boldsymbol{u}, \boldsymbol{c}_0, c_1);$

Figure 5.4: Dual-Regev PKE: Game 4

In this final game, there is no dependence between the challenge given to the adversary and the challenge b, so the probability that the adversary guesses b is upper bounded by $\frac{1}{2}$.

The most important point about the above proof is that while the cryptographic theory underlying the Leftover Hash Lemma and Learning with Errors assumption is in nature analytic, the proof of security which uses them is only algebraic. That is, no complicated analytic arguments must be made in order to carry out the above proof; instead, each transformation is a straightforward syntactic transformation of the security game.

Our logic is designed to handle game transformations such as the ones in the above proof. Our implemented security proof for Dual Regev Encryption

is shown in Figure 5.5. In lines 1-3, we apply the Leftover Hash Lemma. The `move` tactic is used to reorder samplings in the security game, as long as the two reorderings are semantically equivalent. The `assumption_decisional` tactic is used to apply hardness assumptions and information-theoretic lemmas. Note that all required factorings of games in this proof are performed automatically, handled by our use of the SAGBI method in Section 5.4.3. This is reflected by the "!" at the end of the tactic, which asks the proof system to automatically factor the game. (More complicated applications of `assumption_decisional` do require the user to provide some hints to the proof system about how to factor the game. These hints are minimal, however.) The arrow `->` after the tactic specifies that we wish to apply the transformation in the forward direction. (It is possible to apply the LHL and the LWE assumption in reverse, as well. This is used in later proofs.) Throughout, we use the `//` tactic to normalize the game. This tactic unfolds let bindings, and applies a syntactic normal form algorithm to all expressions in the game. The `mat_fold` and `mat_unfold` tactics are used to reason about uniformity of matrices of the form $\mathbb{Z}_q^{n\times(m+k)}$: the `mat_unfold` tactic will separate a uniform sampling of type $\mathbb{Z}_q^{n\times(m+k)}$ into two uniform samplings of types $\mathbb{Z}_q^{n\times m}$ and $\mathbb{Z}_q^{n\times k}$ respectively; the `mat_fold` does the corresponding inverse operation.

The `rnd` tactic is used to reason about transformations of uniform samplings: given two functions f, f^{-1} which must be mutual inverses, the `rnd` tactic allows one to "pull" a uniform sampling through f^{-1}. This is used in two ways in the proof: on lines 13 and 15, we use `rnd` to show that instead of sampling a matrix, we may instead sample its transpose. Whenever the original matrix is used, we now take the transpose of the new sampled matrix. Similarly, on line 19 we use `rnd` to perform an optimistic sampling operation, in which `B` is transformed in

order to remove the additive factor `b?Mu(()):0_{1,1}`. Here, `Mu` is an uninterpreted function from the unit type to 1 by 1 matrices, modelling the message content $\lceil q/2 \rceil$, and `0_{1,1}` is the constant zero matrix of dimension 1 by 1. The notation `_?_:_` is the standard ternary if-then-else construct; thus, we can model the expression $b\lceil q/2 \rceil$ present in the Dual Regev scheme as the expression `b?Mu(()):0_{1,1}`.

```
(* apply LHL *)
move A 1.
assumption_decisional! LHL -> u; //.

(* fold A, u into single matrix Au *)
mat_fold 1 2 Au; //.

(* apply LWE assumption *)
move s 2.
assumption_decisional! LWE -> w; //.

(* unfold LWE distribution *)
rnd w (λ w. tr w) (λ w. tr w); //.
mat_unfold 2 wa wb; //.
rnd wb (λ B. tr B) (λ B. tr B); //.

(* perform optimistic sampling *)
move wb 4.
rnd wb (λ B. B - (b?Mu(()):0_{1,1}))
       (λ B. B + (b?Mu(()):0_{1,1})); //.
indep!.

qed.
```

Figure 5.5: AutoLWE proof for Dual Regev Encryption.

Finally, the `indep!` tactic is used to reason about games such as the game in Figure 5.4, in which the adversary trivially has no advantage. Detail about the proof rules present in our logic is given in Section 5.3.4.

5.3 Logic

Dimensions			
d	::=	n	dimension variable
	|	$d_1 + d_2$	addition
	|	1	constant dimension 1
Types			
t	::=	$\mathbb{B}$	boolean value
	|	$\mathbb{Z}_q$	prime field of order q
	|	$\mathbb{Z}_q^{d_1 \times d_2}$	integer matrix
	|	$\mathsf{list}_d\ t$	list
	|	$t \times \ldots \times t$	tuple
Expressions			
M	::=	$\mathbf{0}$	null matrix
	|	$\mathbf{I}$	identity matrix
	|	$[M]$	constant list
	|	$M + M$	addition
	|	$M \times M$	multiplication
	|	$-M$	inverse
	|	$M \parallel M$	concatenation
	|	$\mathsf{sl}\ M$	left projection
	|	$\mathsf{sr}\ M$	right projection
	|	$M^\top$	transpose

Figure 5.6: Syntax of expressions (selected)

Our logic reasons about probabilistic expressions P, built from atomic expressions of the form $\mathsf{Pr}G\phi$, where G is a game, and ϕ is an event. Games are probabilistic programs with oracle and adversary calls, and ϕ is the winning condition of the game. The proof rules of the logic formalize common patterns of reasoning from the game-playing approach to security proofs. In their simpler form, proof steps will transform a proof goal $\mathsf{Pr}G\phi \leq p$ into a proof goal $\mathsf{Pr}G'\phi' \leq p'$, with $p = p' + c$, and G' a game derived from G; alternatively, they will directly discharge the proof goal $\mathsf{Pr}G\phi \leq p$ (and give a concrete value for p) when the

proof goal is of a simple and specific form, e.g., bounding the probability that an adversary guesses a uniformly distributed and secret value.

In order to be able to accommodate lattice-based constructions, the following novelties are necessary: the expression language includes vectors and matrices; new rules for probabilistic samplings and for oracle-relative assumptions (both in the information-theoretic and computational forms). These extensions do not pose any foundational challenge, but must be handled carefully to obtain the best trade-off between generality and automation.

5.3.1 Games

Games consist of a security experiment in which an adversary with oracle access interacts with a challenger and of an assertion that determines the winning event.

Expressions The expression language operates over booleans, lists, matrices, and integers modulo q, and includes the usual algebraic operations for integer modulo q and standard operators for manipulating lists and matrices. The operations for matrices include addition, multiplication and transposition, together with *structural operations* that capture the functionalities of block matrices, and can be used for (de)composing matrices from smaller matrices. *concatenation*, *split left*, and *split right*. The type of lists, list_d, denotes a list of length d. Lists are manipulated symbolically, so do not support arbitrary destructuring. Lists may be constructed through the *constant list* operation $[\cdot]$, which takes a type τ to the type $\mathsf{list}_d\ \tau$, for any d. All of the matrix operations are lifted pointwise to lists.

Assertions (event expressions)

ϕ	::=	e	expression
	|	$\exists\, b_1, \ldots, b_k.\, e$	existential queries
	|	$\forall\, b_1, \ldots, b_k.\, e$	universal queries
where			
b	::=	$x \in Q_{\mathsf{O}}$	x ranges over queries

for all queries

Game commands

gc	::=	$\mathsf{let}\ x = e$	assignment
	|	$x \xleftarrow{\$} \mu$	sampling from distr.
	|	$\mathsf{assert}(\phi)$	assertion
	|	$y \leftarrow \mathcal{A}(x)\ \mathsf{with}\ \overrightarrow{\emptyset}$	adversary call

Oracle commands

oc	::=	$\mathsf{let}\ x = e$	assignment
	|	$x \xleftarrow{\$} \mu$	sampling from distr.
	|	$\mathsf{guard}(b)$	guard

Oracle definitions

$\emptyset$	::=	$\mathsf{O}(x) = \{\overrightarrow{oc};\ \mathsf{return}\ e\}$

Game definitions

G	::=	$\{\overrightarrow{gc};\ \mathsf{return}\ e\}; \overrightarrow{\emptyset}$

where $\mathcal{A}$ and $\emptyset$ range over adversary and oracle names respectively.

Figure 5.7: Syntax of games

The syntax of expressions (restricted to expressions for matrices) is given in Figure 5.6. Selected typing rules for expressions are given in the Appendix, in Figure 5.12. Expressions are deterministic, and are interpreted as values over their intended types. Specifically, we first interpret dimensions as (positive) natural numbers. This fixes the interpretation of types. Expressions are then interpreted in the intended way; for instance, transposition is interpreted as matrix transposition, etc.

Games Games are defined by a sequence of commands (random samplings, assignments, adversary calls) and by an assertion. The command defines the computational behavior of the experiment whereas the assertion defines the winning event. Each adversary call contains a list of oracles that are available to the adversary; oracles are also defined by a sequence of commands (random samplings, assignments, assert statements) and by a return expression. The grammars for oracle definitions and game definitions are given in Figure 5.7.

The operational behavior of oracles is defined compositionally from the operational behavior of commands:

- random sampling $x \xleftarrow{\$} \mu$: we sample a value from μ and store the result in the variable x;
- assignments: $\mathsf{let}\ x = e$: we evaluate the expression e and store the result in the variable x;
- assertion $\mathsf{guard}(b)$: we evaluate b and return $\perp$ if the result is false. Guards are typically used in decryption oracles to reject invalid queries.

In addition, we assume that every oracle $\emptyset$ comes with a value $\delta_\emptyset$ that fixes the maximal number of times that it can be called by an adversary. To enforce this upper bound, the execution is instrumented with a counter $c_\emptyset$ that is initially set to 0. Then, whenever the oracle is called, one checks $c_\emptyset \geq \delta_o$; if so, then $\perp$ is returned. Otherwise, the counter $c_\emptyset$ is increased, and the oracle body is executed. In order to interpret events, we further instrument the semantics of the game to record the sequence of interactions between the adversary and the oracle. Specifically, the semantics of oracles is instrumented with a query set variable $Q_\emptyset$ that is initially set to $\emptyset$. Then, for every call the query parameters are stored

in $Q_\emptyset$. (Following [16] it would be more precise to hold a single list of queries, rather than a list of queries per oracle, but the latter suffices for our purposes.)

Informally, adversaries are probabilistic computations that must execute within a specific amount of resources and are otherwise arbitrary. One simple way to give a semantics to adversaries is through syntax, i.e., by mapping adversary names to commands, and then interpret these commands using the afore described semantics. However, our language of games is too restrictive; therefore, we map adversary names to commands in a more expressive language, and then resort to the semantics of this richer language. For convenience of meta-theoretic proofs, e.g., soundness, it is preferable to choose a language that admits a set-theoretical semantics. For instance, one can use the probabilistic programming language pWhile to model the behavior of the adversaries.

The semantics of games is defined compositionally from the operational behavior of commands, oracles, and adversaries:

- assertion $\mathsf{assert}(\phi)$: we evaluate ϕ and abort if the result is false.
- adversary call $y \leftarrow \mathcal{A}(e)$ $\mathsf{with}\ \vec{\mathcal{O}}$: we evaluate e, call the adversary $\mathcal{A}$ with the result as input, and bind the output of the adversary to y. The adversary is provided with access to the oracles $\vec{\mathcal{O}}$.

Finally, the interpretation of $\mathsf{Pr}[G\phi]$ is to be the probability of ϕ in the sub-distribution obtained by executing G.

Throughout the paper, we assume that the games satisfy the following well-formedness conditions and (without loss of generality) hygiene conditions: (WF1) all variables must be used in scope; (WF2) commands must be well-

typed; (Hyg1) adversary and oracle names are distinct; (Hyg2) bound variables are distinct.

5.3.2 Reasoning about expressions

Our indistinguishability logic makes use of two main relations between expressions: equality and deducibility. Equality is specified through a set of axioms $\mathcal{E}$, from which further equalities can be derived using standard rules of equational reasoning: reflexivity, symmetry, transitivity of equality, functionality of operators, and finally instantiation of axioms. We write $\Gamma \vdash_{\mathcal{E}} e = e'$ if e and e' are provably equal from the axioms $\mathcal{E}$ and the set of equalities Γ. Throughout the paper, we implicitly assume that the set of axioms includes standard identities on matrices.

Deducibility is defined using the notion of contexts. A context C is an expression that only contains a distinguished variable $\bullet$. We write $e \vdash^C_{\mathcal{E}} e'$, where e, e' are expressions and C is a context, if $\vdash_{\mathcal{E}} C[e] = e'$. We write $e \vdash_{\mathcal{E}} e'$ if there exists a context C such that $e \vdash^C_{\mathcal{E}} e'$. Similarly, we write $\Gamma \models e \vdash^C_{\mathcal{E}} e'$ if $\Gamma \vdash_{\mathcal{E}} C[e] = e'$ and $\Gamma \models e \vdash_{\mathcal{E}} e'$ if there exists a context C such that $\Gamma \models e \vdash_{\mathcal{E}} e'$. More generally, a (general) context C is an expression that only contains distinguished variables $\bullet_1, \ldots, \bullet_n$. We write $e_1, \ldots, e_n \vdash^C_{\mathcal{E}} e'$, where $e_1, \ldots, e_n, e'$ are expressions and C is a context, if $\vdash_E C[e_1, \ldots, e_n] = e'$. We write $e_1, \ldots, e_n \vdash_{\mathcal{E}} e'$ if there exists a context C such that $e_1, \ldots, e_n \vdash^C_{\mathcal{E}} e'$. Similarly, we write $\Gamma \models e_1, \ldots, e_n \vdash^C_{\mathcal{E}} e'$ if $\Gamma \models C[e_1, \ldots, e_n] =_{\mathcal{E}} e'$ and $\Gamma \models e_1, \ldots, e_n \vdash_{\mathcal{E}} e'$ if there exists a context C such that $\Gamma \models e_1, \ldots, e_n \vdash_{\mathcal{E}} e'$. Intuitively, a context is a recipe that shows how some expression may be computed given other expressions. If we consider matrices,

we may have $M + N, O, N \vdash M \times O$ with the context $C(\bullet_1, \bullet_2, \bullet_3) := (\bullet_1 - \bullet_3) \times \bullet 2$.

5.3.3 Strongest postcondition

A desirable property of any logic is that one can replace equals by equals. In particular, it should always be possible to replace an expression e by an expression e' that is provably equivalent to e. However, it is often desirable to use a stronger substitution property which allows to replace e by an expression e' that is provably equivalent to e relative to the context in which the replacement is to be performed. To achieve this goal, our proof system uses a *strongest postcondition* to gather all facts known at a position p in the main command. The computation of ${}^G p$ is done as usual, starting from the initial position of the program with the assertion true and adding at each step the assertion ϕ_c corresponding to the current command c, where:

$$\begin{aligned} \phi_{\mathsf{let}\ x=e} &= x = e \\ \phi_{\mathsf{guard}(b)} &= b \\ \phi_{\mathsf{assert}(e)} &= e \\ \phi_{\forall/\exists b_1,\ldots,b_k.\, e} &= \mathsf{true} \end{aligned}$$

5.3.4 Judgment and Proof Rules

Our computational logic manipulates judgments of the form $P \preceq P'$ where P and P' are probability expressions drawn from the following grammar:

$$P, P' ::= \epsilon \mid c \mid P + P' \mid P - P' \mid c \times P \mid |P| \mid \mathsf{Pr} G\phi,$$

where ϵ ranges over variables, c ranges over constants, $|P|$ denotes absolute value, and $\mathsf{Pr} G\phi$ denotes the success probability of event ϕ in game G. Constants

$$[\textsc{False}]\ \frac{}{\Pr G\mathsf{false} \preceq 0} \qquad [\textsc{Case}]\ \frac{\Pr G\phi \wedge c \preceq \epsilon_1 \quad \Pr G\phi \wedge \neg c \preceq \epsilon_2}{\Pr G\phi \preceq \epsilon_1 + \epsilon_2}$$

$$[\textsc{Refl}]\ \frac{}{\Pr G\phi \preceq \Pr G\phi} \qquad [\textsc{Add}]\ \frac{P \preceq \epsilon_1 \qquad P' \preceq \epsilon_2}{P + P' \preceq \epsilon_1 + \epsilon_2} \qquad [\textsc{Eq}]\ \frac{P \preceq \epsilon \qquad \vdash P' \preceq P}{P' \preceq \epsilon}$$

$$[\textsc{Swap}]\ \frac{\Pr G\{c';\ c\}_p\phi \preceq \epsilon}{\Pr G\{c;\ c'\}_p\phi \preceq \epsilon} \qquad [\textsc{Insert}]\ \frac{\Pr G\{c;\ c'\}_p\phi \preceq \epsilon}{\Pr G\{c'\}_p\phi \preceq \epsilon}\ \boxed{\begin{array}{l}c \text{ sampling, let,}\\ \text{or } \mathsf{guard}(\mathsf{true})\end{array}}$$

$$[\textsc{Subst}]\ \frac{\Pr G\{e\}_p\phi \preceq \epsilon}{\Pr G\{e'\}_p\phi \preceq \epsilon}\ \boxed{{}^{SE}p \models e =_{\mathcal{E}} e'}$$

$$[\textsc{Abstract}]\ \frac{\left|\Pr G'_1\phi_1 - \Pr G'_2\phi_2\right| \preceq \epsilon}{|\Pr G_1\phi_1 - \Pr G_2\phi_2| \preceq \epsilon}\ \boxed{\begin{array}{l}G_1 \equiv G'_1[\mathcal{B}]\\ G_2 \equiv G'_2[\mathcal{B}]\end{array}}$$

$$[\textsc{Rand}]\ \frac{\Pr G\{s \xleftarrow{\$} t';\ \mathsf{let}\ r = C[s]\}_p\phi \preceq \epsilon}{\Pr G\{r \xleftarrow{\$} t\}_p\phi \preceq \epsilon}\ \boxed{{}^{G}p \models C'[C] =_{\mathcal{E}} \bullet}$$

$$[\textsc{RFold}]\ \frac{\Pr G\{x \xleftarrow{\$} \mathbb{Z}_q^{d_1\times(d_2+d'_2)};\ \mathsf{let}\ x_1 = \mathsf{sl}\ x;\ \mathsf{let}\ x_2 = \mathsf{sr}\ x\}_p\phi \preceq \epsilon}{\Pr G\{x_1 \xleftarrow{\$} \mathbb{Z}_q^{d_1\times d_2};\ x_2 \xleftarrow{\$} \mathbb{Z}_q^{d_1\times d'_2}\}_p\phi \preceq \epsilon}$$

$$[\textsc{RUnfold}]\ \frac{\Pr G\{x_1 \xleftarrow{\$} \mathbb{Z}_q^{d_1\times d_2};\ x_2 \xleftarrow{\$} \mathbb{Z}_q^{d_1\times d'_2};\ \mathsf{let}\ x = x_1 \parallel x_2\}_p\phi \preceq \epsilon}{\Pr G\{x \xleftarrow{\$} \mathbb{Z}_q^{d_1\times(d_2+d'_2)}\}_p\phi \preceq \epsilon}$$

$$[\textsc{Upto}]\ \frac{\Pr G\{\mathsf{guard}(c)\}_p\phi \preceq \epsilon_1 \qquad \Pr G\{\mathsf{guard}(c)\}_p\exists\, x \in Q_o.\, c(x) \neq c'(x) \preceq \epsilon_2}{\Pr G\{\mathsf{guard}(c')\}_p\phi \preceq \epsilon_1 + \epsilon_2}\ \boxed{p \text{ first position in } o}$$

$$[\textsc{Guess}]\ \frac{\Pr G;\ x \leftarrow \mathcal{A}()\phi \preceq \epsilon}{\Pr G\exists\, x \in Q_o.\, \phi \preceq \epsilon}$$

$$[\textsc{Find}]\ \frac{\Pr G;\ x \leftarrow \mathcal{A}(e)\phi_1 \wedge \phi_2 \preceq \epsilon}{\Pr G(\exists\, x \in Q_o.\, \phi_1) \wedge \phi_2 \preceq \epsilon}\ \boxed{\begin{array}{l}C \text{ efficient and}\\ {}^{G}|G| \models C[(e,x)] =_{\mathcal{E}} \phi_1\end{array}}$$

Figure 5.8: Selected proof rules

include concrete values, e.g., 0 and $\frac{1}{2}$, as well as values whose interpretation will depend on the parameters of the scheme and the computational power of the adversary, e.g., its execution time or maximal number of oracle calls.

Proof rules are of the form

$$\frac{P_1 \preceq \epsilon_1 \quad \ldots \quad P_k \preceq \epsilon_k}{P \preceq \epsilon}$$

where P_is and P are probability expressions, ϵ_is are variables and finally ϵ is a probability expression built from variables and constants.

Figure 5.8 present selected rules of the logic. In many cases, rules consider judgments of the form $\mathsf{Pr}G\phi \preceq \epsilon$; similar rules exist for judgments of the form $|\mathsf{Pr}G\phi - \mathsf{Pr}G'\phi'| \preceq \epsilon$.

Rules [FALSE] and [CASE] formalize elementary axioms of probability theory. Rules [REFL] and [ADD] formalize elementary facts about real numbers. Rule [EQ] can be used to replace a probability expression by another probability expression that is provably smaller within the theory of reals. For instance, derivations commonly use the identity $\epsilon_1 \leq |\epsilon_1 - \epsilon_2| + \epsilon_2$.

Rules [SWAP], [INSERT], [SUBST] are used for rewriting games in a semantics-preserving way. Concretely, rule [SWAP] swaps successive commands (at position p) that can be reordered (are dataflow independent in the programming language terminology). By chaining applications of the rule, one can achieve more general forms of code motion. Rule [INSERT] inserts at position p command that does not carry any operational behaviour. Rule [SUBST] substitutes at position p an expression e by another expression e' that is contextually equivalent at p, i.e., ${}^{G}p \models e =_{\mathcal{E}} e'$ holds.

The rule [RAND] performs a different transformation known as optimistic sampling. It replaces a uniform sampling from t by $s \xleftarrow{\$} t'$; return $C[s]$. To ensure that this transformation is correct, the rule checks that C is provably bijective at the program point where the transformation arises, using a candidate inverse context C' provided by the user. Rules [RFOLD] and [RUNFOLD] are dual and are used to manipulate random samplings of matrices. The rule [RFOLD] is used to turn two uniform samplings of matrices into one uniform sampling of the concatenation; conversely, the rule [RUNFOLD] may be used to turn one uniform sampling of a concatenation into uniform samplings of its component parts. (We also have similar rules [LFOLD] and [LUNFOLD] in order to manipulate the vertical component of the dimension.) These rules are primarily used to apply axioms which are stated about matrices of compound dimension.

The rule [ABSTRACT] is used for applying computational assumptions. The rule can be used to instantiate a valid judgment with a concrete adversary. The side-conditions ensure that the experiments G_1 and G_2 are syntactically equivalent to the experiment $G_1'[\mathcal{B} := B]$ and $G_2'[\mathcal{B} := B]$, where the notation $G'[\mathcal{B} := B]$ represents the game obtained by inlining the code of $\mathcal{B}$ in G'. Because of the requirement on syntactic equivalence, it is sometimes necessary to apply multiple program transformations before applying an assumption.

The rule [UPTO] rule is used for replacing guard(c') at position p in an oracle with guard(c). According to the usual principle for reasoning up to failure events, the rule yields two proof obligations: bound the probability of the original event and the probability that the adversary performs a query where the results of c and c' differ.

The rules [GUESS] and [FIND] rules are used to deal with winning events

involving existential quantification.

The logic also contains a rule for hybrid arguments. The rule is similar to [22] and ommitted For lack of space.

5.3.5 Soundness

All proof rules of the logic are sound. To state soundness, we lift the interpretation of games to an interpretation of judgments and derivations. This is done by first defining a fixed interpretation of dimensions that is used for all the games of the derivation. Then, we define the interpretation of P inductively. We say that judgment $P \leq P'$ is *valid* iff the inequality holds for every valid interpretation of P and P'. Finally, one can prove that $P \leq P'$ is valid whenever $P \leq P'$ is derivable in the logic.

5.3.6 Axioms Used

Here, we describe the axioms used to prove the schemes in Sections 5.2 and 5.5 secure. Each axiom is *decisional*, in that it is a claim about the closeness of two games. This is modeled by having both games end with a bit output b, so that each axiom is a claim of the form $|\mathsf{Pr}G_0 b - \mathsf{Pr}G_1 b| \leq \epsilon$. This allows us to apply the [ABSTRACT] rule from Figure 5.8.

Learning with Errors. Recall from Section 5.2 that the LWE assumption states that the distribution $(\boldsymbol{A}, \boldsymbol{s}^\mathsf{T}\boldsymbol{A} + \boldsymbol{e})$ is indistinguishable from uniform, where $\boldsymbol{A}$ and

$\boldsymbol{s}$ are uniformly sampled elements of $\mathbb{Z}_q^{n\times m}$ and $\mathbb{Z}_q^n$ respectively, and $\boldsymbol{e}$ is sampled from some given error distribution.

Our concrete encoding is given in Figure 5.9. Since our logic only deals with uniform samplings, in order to encode more complicated sampling algorithms such as the error distribution for LWE, we separate the sampling algorithm into a *coin sampling* stage and a *deterministic* stage. In the coin sampling stage, an element of $\{0,1\}^c$ is sampled, where c is the number of coins the sampling algorithm will use. (Since the sampling algorithm is polynomial time, c will be a polynomial of the security parameter.) In the deterministic stage, we call an uninterpreted function (here, Chi) which uses the sampled coins to produce the output of the distribution.

In various applications of the LWE assumption, the parameter settings of Figure 5.9 will alter slightly – for instance, in the Dual Regev scheme from Section 5.2, we do not use m on the nose, but rather $m+1$. This difference is immaterial to the validity of the assumption.

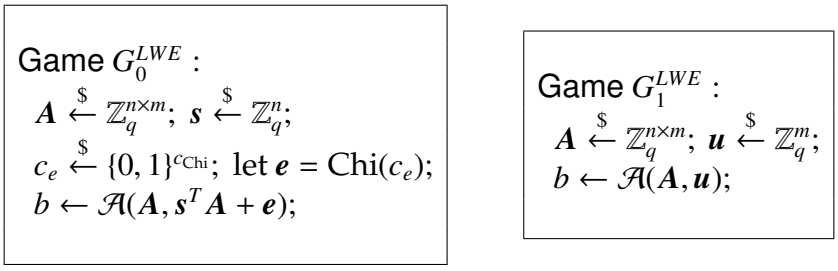

Figure 5.9: The LWE assumption, encoded in AutoLWE.

Leftover Hash Lemma. The most subtle part of our proofs is often not applying the LWE assumption, but rather applying the Leftover Hash Lemma. This is because the LHL is an *information-theoretic* judgment rather than a computa-

tional one; information-theoretic judgments enjoy stronger composition properties than computational judgments.

Recall that the (basic) LHL states that the distribution $(\boldsymbol{A}, \boldsymbol{AR}, \boldsymbol{wR})$ is statistically close to the distribution $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{wR})$, where $\boldsymbol{A}$ is a uniformly random element of $\mathbb{Z}_q^{n \times m}$, $\boldsymbol{R}$ is a uniformly random element of $\{-1, 1\}^{m \times k}$ (interpreted as a matrix), and $\boldsymbol{w}$ is a fixed arbitrary vector in $\mathbb{Z}_q^m$. For the LHL to hold, however, we can actually relax the requirements on $\boldsymbol{A}$: instead of $\boldsymbol{A}$ being sampled uniformly, we only require that $\boldsymbol{A}$ is sampled from a distribution which is *statistically close* to uniform.

In the literature, it is often the case that the lemma being applied is not the LHL on the nose, but rather this weakened (but still valid) form in which $\boldsymbol{A}$ only need to be close to uniform. In many of our proofs, this occurs because $\boldsymbol{A}$ is not uniformly sampled, but rather sampled using an algorithm, TrapGen, which produces a vector $\boldsymbol{A}$ statistically close to uniform along with a *trapdoor* T_A, which is kepts secret from the adversary.

By combining the LHL with the TrapGen construction, we obtain the security games in Figure 5.10. Both games are displayed at once: the expressions which vary between the two games are annotated with which game they belong in. In order to model how $\boldsymbol{R}$ is sampled, we sample the component bits of $\boldsymbol{R}$ from $\{0, 1\}^{d_{LHL}}$, and apply a symbolic function, bitinj, which converts these component bits into a matrix. Note in this security game that $\mathbf{w}$ comes from a symbolic adversary, $\mathcal{A}_1$. This models the universal quantification of $\mathbf{w}$ in the LHL. Additionally, note that $\mathcal{A}_2$ actually receives the trapdoor $T_{\mathbf{A}}$. This is counterintuitive, because adversaries in the cryptosystems do not have access to the trapdoor. However, remember that here we are constructing the adver-

sary *for the LHL*; giving $\mathcal{A}_2$ the trapdoor reflects the assertion that the distribution $(\mathbf{A}, \mathbf{AR}, \mathbf{wR}, T_{\mathbf{A}})$ is statistically close to the distribution $(\mathbf{A}, \mathbf{B}, \mathbf{wR}, T_{\mathbf{A}})$, which follows from the information theoretic nature of the LHL.

While we use the assumption from Figure 5.10 in our proofs, we also use several small variations which are also valid. One such variation is in the proof of Dual Regev, where we do not use the TrapGen algorithm, but rather sample $\boldsymbol{A}$ uniformly (and do not give the adversary T_A); additionally, we do not include this linear leakage $\boldsymbol{w}$. Another such variation is used in our CCA proof from Section 5.5. In this instance, we do not transform $\boldsymbol{AR}$ to $\boldsymbol{B}$, but rather to $\boldsymbol{AR} + \boldsymbol{B}$ (thus generalizing our [RAND] rule.) Additionally, we must state the LHL in the CCA proof to be relative to the decryption oracle, which makes use of $\boldsymbol{R}$. This relativized lemma is still valid, however, since the decryption oracle does not leak any information about $\boldsymbol{R}$. It will be interesting future work in order to unify these small variations of the LHL.

Game G_β^{LHL} :

$c \xleftarrow{\$} \{0,1\}^{d_{TG}}$; let $(\mathbf{A}, T_{\mathbf{A}}) = \text{TrapGen}(c)$;

$r \xleftarrow{\$} \{0,1\}^{d_{LHL}}$; let $R = \text{bitinj}(r)$;

$\overline{\text{if } \beta=1}$

$\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$; $\mathbf{w} \leftarrow \mathcal{A}_1()$;

$b \leftarrow \mathcal{A}_2(\mathbf{A}, \overline{\text{if } \beta=0\ \mathbf{AR}}^{\text{if } \beta=1\ \mathbf{B}}, \mathbf{wR}, T_{\mathbf{A}}, \mathbf{w})$;

Figure 5.10: The LHL assumption combined with TrapGen, encoded in AutoLWE.

Distribution Equivalences. In addition to the two main axioms above, we also rely on several opaque probabilistic judgments about distributions from

which the adversary may sample, but are written in terms of private variables which the adversary may not access. For instance, in an Identity-Based Encryption scheme, the adversary could have access to a KeyGen oracle, which must use the master secret key in order to operate. This is the case in Section 5.5.2. In the concrete proof, there is a step in which we change the implementation of the KeyGen oracle from one uninterpreted function to another. Transformations of this sort are encoded using oracle-relative assumptions, which are generalizations of axioms in AutoG&P which allow adversaries to query oracles.

For example, in Figure 5.11, we state closeness of the distributions $D_0(\boldsymbol{s_0}, \cdot)$ and $D_1(\boldsymbol{s_1}, \cdot)$, where both $\boldsymbol{s_0}$ and $\boldsymbol{s_1}$ are unknown to the adversary. (As before, each distribution is separated into a coin sampling stage and a deterministic stage.) Note that $\boldsymbol{s_0}$ and $\boldsymbol{s_1}$ need not be of the same type, since the adversary does not see them. Jumping ahead in (H)IBE part in the case study, D_0, D_1 correspond to the real/simulated key generation algorithms, where s_0 is the master secret key, and s_1 is the secret trapdoor information the simulator knows in order to answer secret key queries.

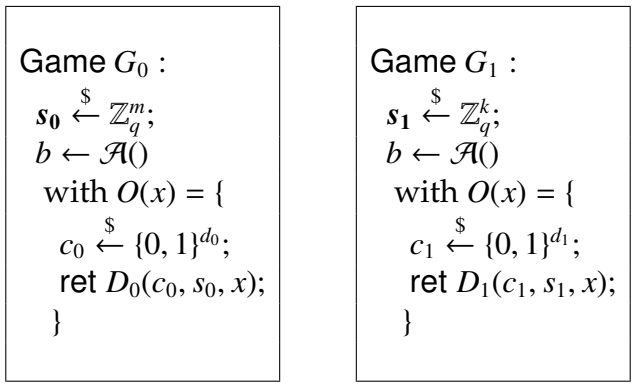

Figure 5.11: Example axiom capturing computational closeness of distributions.

5.4 Deciding deducibility

Several rules involve deducibility problems as side-conditions. For instance, in the [ABSTRACT] rule from Fig 5.8, we may transform a bound involving G_1 and G_2 into a bound involving G_1' and G_2', if there exists a common subgame $\mathcal{B}$ which can be used to factor the former pair into the latter. Finding this subgame $\mathcal{B}$ will induce deducibility subproblems. In order to automate the application of the rules, it is thus necessary to provide algorithms for checking whether deducibility problems are valid. As previously argued, it is desirable whenever possible that these algorithms are based on decision procedures rather than heuristics.

In this section, we provide decision procedures for the theory of Diffie-Hellman exponentiation, both in its basic form and in its extension to bilinear groups, and for the theory of fields. The decision procedures for Diffie-Hellman exponentiation are based on techniques from Gröbner bases. In addition to being an important independent contribution on its own, the algorithms for Diffie-Hellman exponentiation also serve as a natural intermediate objective toward addressing the theory of matrices (although the problems are formally independent). For the latter, we require significantly more advanced algebraic tools. For the clarity of exposition, we proceed incrementally. Concretely, we start by considering the case of fields and non-commutative rings. We respectively provide a decision procedure and a semi-decision procedure. Subsequently, we give a reduction from deducibility for matrices to deducibility for non-commutative rings. The reduction yields a semi-decision procedure for matrices. The algorithms for non-commutative rings and matrices are based on so-called SAGBI [113] (Subalgebra Analog to Gröbner Basis for Ideals) techniques, which as justified below provide a counterpart of Gröbner basis computations for subalge-

bras.

5.4.1 Diffie-Hellman exponentiation

Diffie-Hellman exponentiation is a standard theory that is used for analyzing key-exchange protocols based on group assumptions. It is also used, in its bilinear and multilinear version, in AutoG&P for proving security of pairing-based cryptography. In this setting, the adversary (also often called attacker in the symbolic setting) can multiply groups elements between them, i.e., perform addition in the field, and can elevate a group element to some power he can deduce in the field. Previous work only provides partial solutions: for instance, Chevalier et al [48] only consider products in the exponents, whereas Dougherty and Guttman [61] only consider polynomials with maximum degree of 1 (linear expressions).

The standard form of deducibility problems that arises in this context is defined as follows: let Y be a set of names sampled in $\mathbb{Z}_q$, g some group generator, $\mathcal{E}$ the equational theory capturing field and groups operations, some set $X \subset Y$, $f_1, ...f_k, h \in \mathbb{K}[Y]$ be a set of polynomials over the names, and Γ be a coherent set of axioms. The deducibility problem is then:

$$\Gamma \models X, g^{f_1}, ..., g^{f_k} \vdash_{\mathcal{E}} g^h$$

Proposition 5.4.1. *Deducibility for Diffie-Hellman exponentiation is decidable.*

The algorithm that supports the proof of the proposition proceeds by reducing an input deducibility problem to an equivalent membership problem of the

saturation of some $\mathbb{Z}_q[X]$-module in $\mathbb{Z}_q[Y]$, and by using an extension for modules [62] of Buchberger's algorithm [42] to solve the membership problem.

The reduction to the membership problem proceeds as follows: first, we reduce deducibility to solving a system of polynomial equations. We then use the notion of saturation for submodules and prove that solving the system of polynomial equations corresponding to the deducibility problem is equivalent to checking whether the polynomial h is a member of the saturation of some submodule M. The latter problem can be checked using Gröbner basis computations.

5.4.2 Fields and non-commutative rings

Another problem of interest is when we consider deducibility inside the field rather than the group. The deducibility problem can then be defined as follows: let Y be a set of names sampled in $\mathbb{Z}_q$, $\mathcal{E}$ the equational theory capturing field operations, $f_1, ...f_k, h \in \mathbb{K}[Y]$ be a set of polynomials over the names, and Γ be a coherent set of axioms. The deducibility problem is then:

$$f_1, ..., f_k \vdash_{\mathcal{E}} h$$

We emphasize that this problem is in fact not an instance of the problem for Diffie-Hellman exponentiation. In the previous problem, if we look at field elements, the adversary could compute any polynomial in $K[X]$ but he may now compute any polynomial in $K[f_1, ..., f_k]$, the subalgebra generated by the known polynomials.

Decidability is obtained thanks to [120], where they solve the subalgebra membership problem using methods based on classical Gröbner basis.

Proposition 5.4.2. *Deducibility for fields is decidable.*

If we wish to characterize the full adversary knowledge as done for Diffie-Hellman exponentiation using Gröbner basis, we would have to resort to so-called SAGBI [113] (Subalgebra Analog to Gröbner Basis for Ideals) techniques, which form the counterpart of Gröbner basis computations. However, some finitely generated subalgebras are known to have infinite SAGBI bases [113], thus it can only provide semi-decision for the membership problem.

For the case of non-commutative rings, we are not aware of any counterpart to [120], we resort to the non-commutative SAGBI [103] theory.

Proposition 5.4.3. *Deducibility for non-commutative rings is semi-decidable.*

It is an open problem whether one can give a decision procedure for non-commutative rings. We note that the problem of module membership over a non-commutative algebra is undecidable [102], as there is a reduction from the word problem over a finitely presented group. On the other hand, the problem is known to be decidable for some classes of subalgebras, notably in the the homogeneous case where all monomials are of the same degree.

5.4.3 Matrices

The case of matrices introduces a final difficulty: expressions may involve structural operations. To address the issue, we show that every deducibility problem in the theory of matrices is provably equivalent to a deducibility problem that does not involve structural operations, nor transposition—said otherwise, a deducibility problem in the theory of non-commutative rings.

Proposition 5.4.4. *Deducibility for matrices is semi-decidable.*

The algorithm that supports the proof of semi-decidability for matrices operates in two steps:

1. it reduces the deducibility problem for matrices to an equivalent deducibility problem for non-commutative rings;
2. it applies the semi-decision procedure for non-commutative rings.

The reduction to non-commutative rings is based on a generalization of the techniques introduced in [17] for the theory of bitstrings—note that the techniques were used for a slightly different purpose, i.e., deciding equivalence between probabilistic expressions, rather than for proving deducibility constraints.

The general idea for eliminating concatenation and splitting comes from two basic facts:

- $\mathcal{M} \vdash M \parallel N \Leftrightarrow \mathcal{M} \vdash M \wedge \mathcal{M} \vdash N$
- $\mathcal{M} \cup \{M \parallel N\} \vdash T \Leftrightarrow \mathcal{M} \cup \{M, N\} \vdash T$

For transposition, we observe that it commutes with the other operations, so in a proof of deducibility, we can push the transposition applications to the leaves. Everything that can be deduced from a set of matrices $\mathcal{M}$ and the transpose operation can also be deduced if instead of the transpose operation we simply provide the transposition of the matrices in $\mathcal{M}$.

$$[0]\ \frac{}{\Gamma \vdash 0 : \mathbb{Z}_q^{n,m}} \qquad [\text{ID}]\ \frac{}{\Gamma \vdash \mathbf{I} : \mathbb{Z}_q^{n,n}} \qquad [\text{TR}]\ \frac{\Gamma \vdash M : \mathbb{Z}_q^{m,n}}{\Gamma \vdash M^\top : \mathbb{Z}_q^{n,m}}$$

$$[\text{SL}]\ \frac{\Gamma \vdash M : \mathbb{Z}_q^{n,m+m'}}{\Gamma \vdash \mathsf{sl}\ M : \mathbb{Z}_q^{n,m}} [\text{SR}]\ \frac{\Gamma \vdash M : \mathbb{Z}_q^{n,m+m'}}{\Gamma \vdash \mathsf{sr}\ M : \mathbb{Z}_q^{n,m'}} \qquad [\text{-}]\ \frac{\Gamma \vdash M : \mathbb{Z}_q^{n,m}}{\Gamma \vdash -M : \mathbb{Z}_q^{n,m}} \qquad [\in]\ \frac{M \in \mathcal{M}}{\mathcal{M} \vdash M}$$

$$[\times]\ \frac{\Gamma \vdash M : \mathbb{Z}_q^{n,\ell} \quad \Gamma \vdash M' : \mathbb{Z}_q^{\ell,n}}{\Gamma \vdash M \times M' : \mathbb{Z}_q^{n,m}} \qquad [+]\ \frac{\Gamma \vdash M : \mathbb{Z}_q^{n,m} \quad \Gamma \vdash M' : \mathbb{Z}_q^{n,m}}{\Gamma \vdash M + M' : \mathbb{Z}_q^{n,m}}$$

$$[——]\ \frac{\Gamma \vdash M : \mathbb{Z}_q^{n,m} \quad \Gamma \vdash M' : \mathbb{Z}_q^{n,m'}}{\Gamma \vdash M||M' : \mathbb{Z}_q^{n,m+m'}}$$

Figure 5.12: Typing rules for matrix operators.

Proofs for Lemma 5.4.4 We provide a more detailed proof of Proposition 5.4.4. To reason about matrices deducibility, written $\mathcal{M} \vdash M$ for a set of matrices $\mathcal{M}$ and a matrix M, we use the natural formal proof system $\mathcal{K}$ which matches the operations on expressions (see Figure 5.12), that we extend with the equality rule

$$[\text{EQ}]\ \frac{\mathcal{M} \vdash M_1 \quad \mathcal{M} \vdash M_1 = M_2}{\mathcal{M} \vdash M_2}$$

For ease of writing, we denote $(\frac{A}{B}) := (A^\top||B^\top)^\top$.

Splits elimination

Proposition 5.4.5. *Given a set of matrices $\mathcal{M}$ and a matrix M, we can obtain $\mathcal{S}(\mathcal{M})$ a set of matrices without any concats, such that $\mathcal{M} \vdash M \Leftrightarrow \mathcal{S}(\mathcal{M}) \vdash H$.*

Proof. We notice that the concat operations commute with all the other operators: $(A||B) + (C||D) = (A + C||B + D)$, $(A||B) - (C||D) = (A - C||B - D)$,

$A \times (B||C) = (AB||AC)$, $(A||B) \times \frac{C}{D} = AC + BD$, $(A||B)^\top = (\frac{A^\top}{B^\top})$. Given a set of matrices $\mathcal{M}$, we rewrite the matrices so that the concatenations operators are at the top. We can see the matrices as block matrices with submatrices without any concat, and then, we can create a set $\mathcal{S}(\mathcal{M})$ containing all the submatrices. This preserves deducibility thanks to the Eq rule for the rewriting, and to the split rules for the submatrices. □

Definition 5.4.6. *We call $\mathcal{N}$ the proof system based on $\mathcal{K}$ without splits, and write the deducibility with $\mathcal{M} \vdash_{\mathcal{N}} M$.*

Lemma 5.4.7. *If $\mathcal{M} \vdash (R||S)$ with a proof π (resp. $\mathcal{M} \vdash (\frac{R}{S})$) then $\mathcal{M} \vdash R$ and $\mathcal{M} \vdash S$ with smaller proofs (resp. $\mathcal{M} \vdash R$, $\mathcal{M} \vdash S$) .*

Proof. We prove it by induction on the size of the proof, and by disjunction on the last rule applied.

Base case $|\pi| = 2$, then the proof is a concat on axioms and we can then obtain the sub matrix directly, with a proof of size one.

Induction case

- The last rule is

$$[\text{TR}]\ \frac{(\frac{R}{S})}{(R|S)}$$

Then, we directly obtain by induction on $(\frac{R}{S})$ smaller proofs for R and S.

- The last rule is

$$[\times]\ \frac{M \quad (N^l||N^r)}{(MsN^l|MN^r)}.$$

Then, by induction on the proof of N, we obtain proofs of size smaller than $|\pi| - 1$ of N^l and N^r, and we just have to add a [×] to those proofs, yielding smaller proofs of MN^l and MN^r.

- If the last rule is [+], [-], [——], the proof can be done similarly to the two previous cases.
- The last rule is

$$[\text{SL}]\ \frac{((M|N)|L)}{(M|N)}$$

  Then, we have a proof of $((M|N)|L)$ of size $|\pi| - 1$, so by induction we have a proof of $(M|N)$ smaller than $|\pi| - 1$, and by adding a sL, we for instance obtain M with a proof smaller than $|\pi|$.
- [SR] is similar.

□

Lemma 5.4.8. *If $\mathcal{M}$ is a set of matrix without concatenations, and if $\mathcal{M} \vdash M$, then $\mathcal{M} \vdash_N M$.*

Proof. We prove it by induction on the size of the proof, and by disjunction on the last rule applied.

Base case $|\pi| = 1$, then the only problem might be if the rule used was a split, but as we have matrices without concatenations, this is not possible.

Induction case

- If the last rule is [TR], [+], [-], [——], we conclude by applying the induction hypothesis to the premise of the rule.

- The last rule is

$$[\text{sL}]\ \frac{(M|N)}{M}$$

  Then, we have a proof of $(M|N)$ of size $\pi - 1$, and with lemma 5.4.7 we have a smaller proof of M, on which we can then apply our induction hypothesis to obtain a proof of M without split.

- *splitR* is similar.

□

Concatenations elimination

Definition 5.4.9. *We call $\mathcal{T}$ the proof system based on N without concatenations, and write the deducibility $\mathcal{M} \vdash_{\mathcal{T}} M$.*

Lemma 5.4.10. *If $\mathcal{M}, M, N$ do not contain any concat, then:*

$$\mathcal{M} \vdash_{\mathcal{N}} (M|N) \Leftrightarrow \mathcal{M} \vdash_{\mathcal{T}} M \wedge \mathcal{M} \vdash_{\mathcal{T}} N$$

Proof. The left implication is trivial. For the right one, we once more do a proof by induction on the size of the proof.

Base case $|\pi| = 1$, the last rule is a [——], and we do have a proof of M and N.

Induction case

- The last rule is

$$[\times]\ \frac{M \quad (N^l||N^r)}{(MN^l|MN^r)}$$

Then, by induction on the proof of N, we obtain proofs of size smaller than $|\pi| - 1$ of N^l and N^r without concats, and we just have to add a [×] to those proofs, yielding proofs of MN^l and MN^r without concats.

- If the last rule is [TR], [+], [-], [SL], [SR], we proceed as in the previous case
- The last rule is [——]. Then the induction rule instantly yields the expected proofs. Then, we have a proof of $(M|N)$ of size $\pi - 1$, and with lemma 5.4.7 we have a smaller proof of M, on which we can then apply our induction hypothesis to obtain a proof of M without split.

□

Lemma 5.4.11. *$\mathcal{M} \vdash_{\mathcal{N}} M \Leftrightarrow \forall G \sqsubseteq M, \mathcal{M} \vdash_{\mathcal{T}} G$ Where $G \sqsubseteq H$ denotes the fact that G is a sub matrix of M without any concatenation.*

Proof. The left implication is trivial, we prove the right one. As was done in Lemma 5.4.5, we can see M has a bloc matrix, i.e., with all the concat at the top.

We are given a proof of $\mathcal{M} \vdash_{\mathcal{N}} M$, which must contain all its concatenations at the bottom of the proof tree. If we look at all the highest concat rule in the proof such that no concat is made before, we have some proof of $\mathcal{M} \vdash_{\mathcal{N}} (M_i|M_j)$, and thanks to lemma 5.4.10, we have $\mathcal{M} \vdash_{\mathcal{T}} M_i \wedge (A_i) \vdash_{\mathcal{T}} M_j$. Applying this to all the highest concat rules in the proof yields the result. □

Removal of the transposition

Definition 5.4.12. *We call $\mathcal{V}$ the proof system based on $\mathcal{N}$ without concat, and write the deducibility $\mathcal{M} \vdash_{\mathcal{V}} M$.*

The transposition commutes with the other operations, given a matrix M we define the normal form $N(M)$ where the transposition is pushed to the variables. We extend the notation for normal form to sets of matrices.

Lemma 5.4.13. $\mathcal{M} \vdash_{\mathcal{T}} M \Leftrightarrow \mathcal{M} \cup (N(\mathcal{M}^t)) \vdash_{\mathcal{V}} N(M)$

Proof. $\Leftarrow$ This is trivial, as the normal form can be deduced using the rule [EQ]. $\Rightarrow$ Given the proof of M, we can commute the trans rule with all the others, and obtain a proof tree where all the transposition are just after a [$\in$] rule. Then, any [$\in$] followed by [TRANS] can be replaced by a [$\in$] and a [EQ] when given the input $\mathcal{M} \cup (N(\mathcal{M}^t))$ instead of $\mathcal{M}$. We can thus construct a valid proof of $\mathcal{M} \cup (N(\mathcal{M}^t)) \vdash_{\mathcal{V}} N(M)$ □

Conclusion The proof of proposition 5.4.4 is a direct consequence of Lemmas 5.4.5, 5.4.8, 5.4.11 and 5.4.13.

5.5 Implementations and Case Studies

The implementation of our logic, called AutoLWE, is available at:

`https://github.com/autolwe/autolwe`

AutoLWE is implemented as a branch of AutoG&P and thus makes considerable use of its infrastructure.

Moreover, we have used AutoLWE to carry several case studies (see Figure 5.13): an Identity-Based Encryption scheme and an Hierarchical Identity-

Based Encryption scheme by Agrawal, Boneh and Boyen [2], a Chosen-Ciphertext Encryption scheme from Micciancio and Peikert [100], and an Inner Product Encryption scheme and proof from Agrawal, Freeman, and Vaikuntanathan [3]. These examples are treated in Sections 5.5.2, 5.5.4, 5.5.3 and 5.5.5 respectively.

Globally, our tool performs well, on the following accounts: formal proofs remains close to the pen and paper proofs; verification time is fast (less than 3 seconds), and in particular the complexity of the (semi-)decision procedures is not an issue; formalization time is moderate (requiring at most several hours of programmer effort per proof). One of the main hurdles is the Leftover Hash Lemma, which must be applied in varying levels of sophistication. The Leftover Hash Lemma (and more generally all oracle-relative assumptions) increase the difficulty of guessing (chained) applications of assumptions, and consequently limits automation.

Case study			Proof
Reference	Scheme	Property	LoC
Gentry et al. '08 [75]	dual-Regev PKE	IND-CPA	11
Micciancio et al. '12 [100]	MP-PKE	IND-CCA	98
Agrawal et al. '10 [2]	ABB-IBE	IND-sID-CPA	56
Agrawal et al. '10 [2]	ABB-HIBE	IND-sID-CPA	77
Agrawal et al. '11 [3]	AFV-IPE	IND-wAH-CPA	106

Figure 5.13: Overview of case studies. All proofs took less than three seconds to complete.

5.5.1 Implementation

Security games are written in a syntax closely resembling that shown in Figure 5.1. See Figure 5.5 for an example concrete proof in our system. Each line of the proof corresponds to a proof rule in our logic, as seen in Figure 5.8. All tactic applications are fully automated, except for the application of oracle-relative assumptions. The user must provide some hints to AutoLWE about how the security game needs to be factored in order to apply an oracle-relative assumption. The system in [22] additionally supports a proof search tactic which automatically finds a series of tactics to apply to finish the goal; we do not have a version of that in our setting.

Oracle-relative Assumptions. AutoG&P allows one to add user defined axioms, both to express decisional assertions (two distributions are computationally close) and computational assertions (a certain event has small chance of happening). In AutoG&P, these user-defined axioms are stated in terms of symbolic adversaries, which are related to the main security game by rules such as [ABSTRACT] in Section 5.3.4. However, the symbolic adversaries present in axioms may not have oracles attached to them. While these restricted adversaries can be used to define the LWE assumption, they are not expressive enough to state the oracle-relative axioms we use throughout our proofs. In AutoLWE, we remove this restriction. An example axiom we now support which we did not before is that in Figure 5.11.

Recall that in order to apply a user defined axiom using [ABSTRACT], we must factor the security game into one which is in terms of the axiom's game. This is done essentially by separating the security game into sections, where

each section either reflects the setup code for the axiom, or an instantiation of one of the adversaries in the axiom. We still do this factoring in the case of oracle-relative axioms, but we must also factor oracles in the security game in terms of oracles in the axiom. Once this second step of factoring is done, oracles in the axiom can be compared syntactically to factored oracles in the security game.

Theory of Lists and Matrices. Note that in our case studies, we manipulate both matrices and *lists* of matrices (often simultaneously). Thus, both our normal form algorithm and our deducibility reduction from Section 5.4.3 must be lifted to apply to lists of matrices as well. This is what allows our system to reason about the more complicated HIBE scheme in a manner similar to the IBE scheme, which does not use lists.

In order to do this, we do not implement our main algorithms on expressions of matrices directly, but instead over a general *signature* of matrices, encoded as a certain type of an ML module. We then instantiate this signature both with matrices and lists of matrices. By doing so, we receive an implementation for our new algorithms which operate uniformly across these two types of expressions.

Deduction algorithms. Many implementations of Gröbner basis computations can be found online, but all of them are only usable for polynomial ideals. In order to handle module and non-commutative subalgebra, we thus implemented generic versions of the Buchberger algorithm for $K[X]$-module and the SAGBI algorithm and plugged them into AutoLWE. The algorithms performed well: we could prove all the LWE examples, and the pairing-based examples

very quickly, using the SAGBI methods. The efficiency of the computations contrasts with the complexity of the algorithms, which is high because the saturation squares up the number of inputs terms and the Gröbner Basis can be at worst a double exponential. However, we are dealing with relatively small instances of our problem that are extracted from concrete primitives.

5.5.2 Identity-Based Encryption

Identity-based encryption is a generalization of public key encryption. In IBE, the secret key and ciphertext are associated with different identity strings, and decryption succeeds if and only if the two identity strings are equivalent. The security model, IND-sID-CPA, requires adversary to declare challenge identity upfront before seeing the public parameters, and allows adversary to ask for secret key for any identity except for the challenge identity, and CPA security holds for ciphertext associated with the challenge identity.

The IBE scheme our system supports is constructed by Agrawal et al. [2]. The scheme operates as follows:

- Matrix $\mathbf{A}$ is generated by algorithm TrapGen, which outputs a random $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a small norm matrix $\mathbf{T} \in Z_q^{m \times m}$ such that $\mathbf{A} \cdot \mathbf{T_A} = 0$. Matrices $\mathbf{A}_1, \mathbf{B}$ are sampled randomly from $\mathbb{Z}_q^{n \times m}$, and $\boldsymbol{u}$ is sampled randomly from $\mathbb{Z}_q^n$. Set $\mathsf{PP} = (\mathbf{A}, \mathbf{A}_1, \mathbf{B}, \boldsymbol{u})$ and $\mathsf{MSK} = \mathbf{T_A}$.
- To encrypt a message $\mu \in \{0, 1\}$ with identity $\mathsf{id} \in \mathbb{Z}_q^n$, one generates a uniform $\boldsymbol{s} \in \mathbb{Z}_q^n$, error vector $\boldsymbol{e}_0 \leftarrow \chi^m$ and error integer $e_1 \leftarrow \chi$ from discrete

Gaussian, a random $\mathbf{R} \in \{0,1\}^{m \times m}$, and computes ciphertext

$$\mathsf{CT} = \boldsymbol{s}^\mathsf{T} [\mathbf{A} || \mathbf{A}_1 + M(\mathsf{id})\mathbf{B} || \boldsymbol{u}] + (\boldsymbol{e}^\mathsf{T} || \boldsymbol{e}^\mathsf{T}\mathbf{R} || e') + (0 || 0 || \lceil q/2 \rceil \mu)$$

- The secret key for identity $\mathsf{id} \in \mathbb{Z}_q^n$ is generated by procedure $\boldsymbol{r} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}_1 + M(\mathsf{id})\mathbf{B}, \mathbf{T}_\mathbf{A}, \boldsymbol{u})$, where we have $\boldsymbol{r}$ is statistically close to χ^{2m}, and $[\mathbf{A} || \mathbf{A}_1 + M(\mathsf{id})\mathbf{B}]\,\boldsymbol{r} = \boldsymbol{u}$.

The idea of the proof is first to rewrite $\mathbf{A}_1$ as $\mathbf{AR} - M(\mathsf{id}^*)\mathbf{B}$, where id^* is the adversary's committed identity. If we do so, we then obtain that the challenge ciphertext is of the form

$$\mathbf{s}^\mathsf{T} [\mathbf{A} || \mathbf{AR} || \mathbf{u}] + (\mathbf{e}^\mathsf{T} || \mathbf{e}^T\mathbf{R} || e') + (0 || 0 || \lceil q/2 \rceil \mu),$$

where $\mathbf{A}$ comes from TrapGen. We then apply a computational lemma about SampleLeft, in order to rewrite the KeyGen oracle to be in terms of another probabilistic algorithm, SampleRight. This is a statement about equivalence of distributions from which the adversary may sample, so must be handled using an oracle-relative assumption. This is done as described in Section 5.3.6. The computational lemma states that, for appropriately sampled matrices,

$$\text{SampleLeft}(\mathbf{A}, \mathbf{AR} + \mathbf{B}, T_A, \boldsymbol{u}) \approx \text{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, T_B, \boldsymbol{u}),$$

where $\mathbf{A}$ is sampled from TrapGen in the first and uniform in the second, and $\mathbf{B}$ is sampled uniformly in the first and from TrapGen in the second. By applying this transformation to our KeyGen oracle, we transform our matrix $\mathbf{A}$ from one sampled from TrapGen to uniform.

Now that $\mathbf{A}$ is uniform, we finish the proof by noticing that our challenge ciphertext is equal to $\mathbf{b} || \mathbf{bR} || b + \lceil q/2 \rceil \mu$, where $(\mathbf{b}, b)$ forms an LWE distribution of

dimension $n \times m + 1$. Thus we may randomize b to uniform, and apply the `rnd` tactic to erase μ from the ciphertext.

The main point of interest in this proof is the initial rewrite $\mathbf{A}_1 \rightarrow \mathbf{AR} - M(\mathsf{id}^*)\mathbf{B}$. Given that $\mathbf{A}_1$ is uniform, we may first apply optimistic sampling to rewrite $\mathbf{A}_1$ to $\mathbf{A}_2 - M(\mathsf{id}^*)\mathbf{B}$, where $\mathbf{A}_2$ is uniformly sampled. Thus, we now only need to perform the rewrite $\mathbf{A}_2 \rightarrow \mathbf{AR}$. This rewrite is not at all trivial, because $\mathbf{A}$ at this point in the proof comes from TrapGen. However, as noted in Section 5.3.6, it is sound to apply the LHL in this case, because TrapGen generates matrices which are close to uniform in distribution. Thus, we can use the LHL as encoded in Figure 5.10.

5.5.3 CCA1-PKE

The CCA1-PKE scheme we study is proposed by Micciancio and Peikert [100]. In comparison with the CPA-PKE scheme [75] described in Section 5.2, the security model of CCA1-PKE is stronger: the adversary can query a decryption oracle for any ciphertext he desires before receiving the challenge ciphertext. The scheme operates as follows:

- Matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is sampled randomly and $\mathbf{R} \leftarrow \{-1, 1\}^{m \times m}$. Set $\mathsf{PK} = (\mathbf{A}, \mathbf{AR})$ and $\mathsf{SK} = \mathbf{R}$.
- Let $M : \mathbb{Z}_q^n \rightarrow \mathbb{Z}_q^{n \times m}$ be an embedding from $\mathbb{Z}_q^n$ to matrices, such that for distinct $\mathbf{u}$ and $\mathbf{v}$, $M(\mathbf{u}) - M(\mathbf{v})$ is full rank. To encrypt a message $\mu \in \{0, 1\}$, one generates a uniform $\mathbf{s} \in \mathbb{Z}_q^n$, a uniform $\mathbf{u} \in \mathbb{Z}_q^n$, a uniform matrix $R' \in \{-1, 1\}^{m \times m}$ and an error vector $\mathbf{e} \in \mathbb{Z}_q^m$ sampled from a discrete Gaussian,

and computes the ciphertext

$$\mathbf{c}_0 = \mathbf{u}, \mathbf{c}_1 = \mathbf{s}^{\mathsf{T}}\mathbf{A}_{\mathbf{u}} + (\mathbf{e}^{\mathsf{T}} || \mathbf{e}^{\mathsf{T}} * R') + (0 || \mathsf{Encode}(\mu))$$

where $\mathbf{A}_{\mathbf{u}} := [\mathbf{A} || -\mathbf{AR} + M(\mathbf{u})\mathbf{G}]$, $\mathbf{G}$ is a publicly known gadget matrix, and $\mathsf{Encode} : \{0,1\} \to \mathbb{Z}_q^m$ sends μ to $\mu\lceil q/2 \rceil(1, \ldots, 1)$.

- To decrypt a ciphertext ($\mathbf{u} := \mathbf{c}_0, \mathbf{c}_1$) with $\mathsf{SK} = \mathbf{R}$ and $\mathbf{u} \neq 0$, one computes $\mathbf{A}_{\mathbf{u}}$ and calls a procedure $\mathsf{Invert}(\mathbf{A}_{\mathbf{u}}, \mathbf{R}, \mathbf{c}_1)$, which will output $\mathbf{s}$ and $\mathbf{e}$ such that $\mathbf{c}_1 = \mathbf{s}^T\mathbf{A}_{\mathbf{u}} + \mathbf{e}$, where $\mathbf{e}$ has small norm. By doing a particular rounding procedure using $\mathbf{c}_1, \mathbf{s}, \mathbf{e}$, and $\mathbf{R}$, the message bit μ can be derived.

The main subtlety of the proof is that the secret key $\mathbf{R}$ is used in the decryption oracle. Because of this, we must apply the Leftover Hash Lemma relative to this oracle, by using oracle-relative axioms. As we will see, not all uses of the LHL are valid in this new setting; care must be taken to ensure that the axioms derived from the LHL are still cryptographically sound.

The high-level outline of the proof is as follows: first, we note that instead of using a fresh $\mathbf{R}'$ to encrypt, we can actually use the secret key $\mathbf{R}$. This is justified by the following corollary of the Leftover Hash Lemma: the distribution $(\mathbf{A}, \mathbf{AR}, \boldsymbol{e}, \boldsymbol{e}\mathbf{R}')$ is statistically close to the distribution $(\mathbf{A}, \mathbf{AR}, \boldsymbol{e}, \boldsymbol{e}\mathbf{R})$ where $\mathbf{A}, \mathbf{R}, \mathbf{R}'$, and $\boldsymbol{e}$ are sampled as in the scheme. This corollary additionally holds true relative to the decryption oracle, which makes use of $\mathbf{R}$.

Once we use $\mathbf{R}$ to encrypt instead of $\mathbf{R}'$, we again use the Leftover Hash Lemma to transform $\mathbf{AR}$ into $-\mathbf{AR} + M(\boldsymbol{u})G$, where $\boldsymbol{u}$ is generated from the challenge encryption. Again, this invocation of the Leftover Hash Lemma is stated relative to the decryption oracle. Crucially, note here that we do *not* transform $\mathbf{AR}$ directly into uniform, as we did before: the reason being is that this trans-

formation would actually be *unsound*, because it would decouple the public key from $\mathbf{R}$ as it appears in the decryption oracle. Thus, we must do the transformation $\mathbf{AR} \rightarrow -\mathbf{AR} + M(\boldsymbol{u})G$ in one step, which is cryptographically sound relative to the decryption oracle. (Currently, we must write this specialized transformation as a unique variant of the Leftover Hash Lemma, as discussed in Section 5.3.6; future work will involve unifying these separate variants.)

At this point, we may apply the LWE assumption along with a more routine invocation of the LHL in order to erase the message content from the challenge ciphertext, which finishes the proof.

5.5.4 Hierarchical Identity-Based Encryption

Hierarchical IBE is an extension of IBE. In HIBE, the secret key for ID string id can delegate secret keys for ID strings id', where id is a prefix for id. Moreover, decryption succeeds if the ID string for the secret key is a prefix of (or equal to) the ID string for the ciphertext. The security model can be adapted according to the delegation functionality.

The HIBE construction our system supports is described in [2]. The ID space for HIBE is $\mathsf{id}_i \in (\mathbb{Z}_q^n)^d$. The secret key for ID string $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_\ell)$, where $\mathsf{id}_i \in \mathbb{Z}_q^n$, is a small-norm matrix $\mathbf{T}$, such that $\mathbf{F}_{\mathsf{id}}\mathbf{T} = 0$, and

$$\mathbf{F}_{\mathsf{id}} = [\mathbf{A}_0 || \mathbf{A}_1 + M(\mathsf{id}_1)\mathbf{B}|| \cdots || \mathbf{A}_\ell + M(\mathsf{id}_\ell)\mathbf{B}]$$

We note that $\mathbf{T}$ can be computed as long as we know the secret key for id', where id' is a prefix of id. Ciphertext for ID string id can be generated similarly with respect to matrix $\mathbf{F}_{\mathsf{id}}$.

The security proof of HIBE is similar to the counterpart of IBE. The challenge ID string $\mathsf{id}^* = (\mathsf{id}_1^*, \ldots, \mathsf{id}_\ell^*)$ is embedded in PP as

$$\forall i \in [\ell], \mathbf{A}_i = \mathbf{A}\mathbf{R}_i - M(\mathsf{id}_i^*)\mathbf{B},\ \forall \ell < j \leq d, \mathbf{A}_j = \mathbf{A}\mathbf{R}_j$$

For admissible query $\mathsf{id} = (\mathsf{id}_1, \ldots, \mathsf{id}_k)$, where id is not a prefix of id^*, we have

$$\mathbf{B}_k = [(M(\mathsf{id}_1) - M(\mathsf{id}_1^*))\mathbf{B}\| \cdots \|(M(\mathsf{id}_k) - M(\mathsf{id}_k^*))\mathbf{B}] \neq 0$$

Then we can generate secret key for id using information $\mathbf{B}_k$ and $\mathbf{R}_k = (\mathbf{R}_1\| \cdots \|\mathbf{R}_k)$. In previous cases, we manipulate and apply rewriting rules to matrices. However, in order to reason about the security in a similar manner to pen-and-paper proof, we introduce the *list* notation, and adapt our implementation to operate uniformly across these two types of expressions.

5.5.5 Inner Product Encryption

The IPE scheme our scheme supports is described in [3]. We briefly recall their construction as

- Matrix $\mathbf{A}$ is generated by algorithm $\mathsf{TrapGen}$. Matrices $\{matB_i\}_{i\in[d]}$ are sampled randomly from $\mathbb{Z}_q^{n\times m}$, and random vector $\boldsymbol{u}$ is from $\mathbb{Z}_q^n$. The public parameters $\mathsf{PP} = (\mathbf{A}, \{\mathbf{B}_i\}_{i\in[d]}, \boldsymbol{u})$, and $\mathsf{MSK} = \mathbf{T}_\mathbf{A}$.
- Secret key $\mathsf{SK}_{\boldsymbol{v}} = \boldsymbol{r}$ for vector $\boldsymbol{v} \in \mathbb{Z}_q^d$ is computed by algorithm $\boldsymbol{r} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \sum_{i\in[d]} \mathbf{B}_i\mathbf{G}^{-1}(v_i\mathbf{G}), \mathbf{T}_\mathbf{A}, \boldsymbol{u})$, where for operation $\mathbf{G}^{-1}(\cdot) : \mathbb{Z}_q^{n\times m} \to \mathbb{Z}_q^{m\times m}$, for any $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, it holds that $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$ and $\mathbf{G}^{-1}(\mathbf{A})$ has small norm.
- To encrypt a message $\mu \in \{0, 1\}$ for attribute $\boldsymbol{w}$, one generates a uniform $\boldsymbol{s} \in \mathbb{Z}_q^n$, error vector $\boldsymbol{e}_0 \leftarrow \chi^m$ and error integer $e_1 \leftarrow \chi$ from discrete Gaussian,

random matrices $\{\mathbf{R}_i\}_{i\in[d]} \in \{0,1\}^{m\times m}$, and computes ciphertext $(\boldsymbol{c}_0, \{\boldsymbol{c}_i\}_{i\in[d]}, c)$ as

$$\boldsymbol{c}_0 = \boldsymbol{s}^\top \mathbf{A} + \boldsymbol{e}_0^\top,\ \boldsymbol{c}_i = \boldsymbol{s}^\top(\mathbf{B}_i + w_i\mathbf{G}) + \boldsymbol{e}_0^\top \mathbf{R}, c = \boldsymbol{s}^\top \boldsymbol{u} + e + \lceil q/2 \rceil \mu$$

The main challenge in the proof is to answer secret key queries for any vector $\boldsymbol{v}$ as long as $\langle \boldsymbol{v}, \boldsymbol{w}_0 \rangle, \langle \boldsymbol{v}, \boldsymbol{w}_1 \rangle$ are both not 0, where $(\boldsymbol{w}_0, \boldsymbol{w}_1)$ is declared by adversary upfront. The attribute $\boldsymbol{w}_b$ (b is a random bit) is first embedded in PP, i.e., $\mathbf{B}_i = \mathbf{A}\mathbf{R}_i - w_{bi}\mathbf{G}, \forall i \in [d]$, where $\mathbf{R}_i$ is a small matrix. By unfolding the matrix for query $\boldsymbol{v}$, we have

$$\left[\mathbf{A} \| \sum_{i\in[d]} \mathbf{B}_i \mathbf{G}^{-1}(v_i \mathbf{G})\right] = \left[\mathbf{A} \| \mathbf{A} \sum_{i\in[d]} \mathbf{R}_i \mathbf{G}^{-1}(v_i \mathbf{G}) + \langle \boldsymbol{w}_b, \boldsymbol{v} \rangle \mathbf{G}\right]$$

If $\langle \boldsymbol{w}_b, \boldsymbol{v} \rangle \neq 0$, the algorithm $\mathsf{SampleRight}$ can be used to generate secret key for $\boldsymbol{v}$.

The sequence of hybrids generated in symbolic proof is a bit different from the pen-and-paper proof. In particular, instead of transforming from embedding of challenge attribute $\boldsymbol{w}_0$ directly to embedding of $\boldsymbol{w}_1$, we use the original scheme as a middle game, i.e., from embedding of $\boldsymbol{w}_0$ to original scheme, then to embedding of $\boldsymbol{w}_1$. The reason for using the original scheme again in the proof is that when using LHL to argue the indistinguishability between $(\mathbf{A}, \{\mathbf{B}_i = \mathbf{A}\mathbf{R}_i - w_{0i}\mathbf{G}\}_i)$ and $(\mathbf{A}, \{\mathbf{B}_i = \mathbf{A}\mathbf{R}_i - w_{1i}\mathbf{G}\}_i)$, the real public parameters $(\mathbf{A}, \{\mathbf{B}_i\}_i)$ actually serves as a middleman. Therefore, to ensure the consistency with respect to public parameters and secret key queries, the real scheme is used to make the transformation valid.

BIBLIOGRAPHY

[1] Shashank Agrawal and Melissa Chase. A study of pair encodings: Predicate encryption in prime order groups. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 259–288. Springer, Heidelberg, January 2016.

[2] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Heidelberg, May / June 2010.

[3] 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.

[4] Shweta Agrawal and Ishaan Preet Singh. Reusable garbled deterministic finite automata from learning with errors. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 80. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[5] Jacob Alperin-Sheriff and Chris Peikert. Circular and KDM security for identity-based encryption. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 334–352. Springer, Heidelberg, May 2012.

[6] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 297–314. Springer, Heidelberg, August 2014.

[7] Prabhanjan Ananth, Xiong Fan, and Elaine Shi. Towards attribute-based encryption for rams from lwe: Sub-linear decryption, and more. In *Advances in Cryptology – ASIACRYPT*, 2019.

[8] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. Cryptology ePrint Archive, Report 2015/730, 2015. `http://eprint.iacr.org/2015/730`.

[9] Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. In *Theory of Cryptography Conference*, pages 125–153. Springer, 2016.

[10] Daniel Apon, Xiong Fan, and Feng-Hao Liu. Deniable attribute based encryption for branching programs from LWE. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 299–329. Springer, Heidelberg, October / November 2016.

[11] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

[12] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 557–577. Springer, Heidelberg, May 2014.

[13] David A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc 1. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 1–5. ACM, 1986.

[14] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[15] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and Santiago Zanella Béguelin. Fully automated analysis of padding-based encryption in the computational model. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 1247–1260. ACM, 2013.

[16] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, and Yassine Lakhnech. Computational indistinguishability logic. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10*, pages 375–386. ACM Press, October 2010.

[17] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, Yassine Lakhnech, and Vincent Laporte. On the equality of probabilistic terms. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*, pages 46–63. Springer, 2010.

[18] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2013.

[19] Gilles Barthe, Xiong Fan, Joshua Gancher, Benjamin Grégoire, Charlie Jacomme, and Elaine Shi. Symbolic proofs for lattice-based cryptography. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 18*, pages 538–555. ACM Press, October 2018.

[20] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In Zhong Shao and Benjamin C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 90–101. ACM, 2009.

[21] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 71–90. Springer, Heidelberg, August 2011.

[22] Gilles Barthe, Benjamin Grégoire, and Benedikt Schmidt. Automated proofs of pairing-based cryptography. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1156–1168. ACM, 2015.

[23] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 1–35. Springer, Heidelberg, April 2009.

[24] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In *Annual International Conference on the The-*

ory and Applications of Cryptographic Techniques, pages 792–821. Springer, 2016.

[25] Rikke Bendlin, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. Lower and upper bounds for deniable public-key encryption. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 125–142. Springer, Heidelberg, December 2011.

[26] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. Implementing TLS with verified cryptographic security. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 445–459. IEEE Computer Society, 2013.

[27] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016*, pages 345–356. ACM, January 2016.

[28] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 82–96. IEEE Computer Society, 2001.

[29] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *27th IEEE Symposium on Security and Privacy, S&P 2006*, pages 140–154. IEEE Computer Society, 2006.

[30] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology—CRYPTO 2001*, pages 213–229. Springer, 2001.

[31] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

[32] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. *IACR Cryptology ePrint Archive*, 2015:1167, 2015.

[33] 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.

[34] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 236–261. Springer, 2014.

[35] Zvika Brakerski, David Cash, Rotem Tsabary, and Hoeteck Wee. Targeted homomorphic attribute-based encryption. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 330–360. Springer, Heidelberg, October / November 2016.

[36] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013.

[37] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 535–564. Springer, Heidelberg, April / May 2018.

[38] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 264–302. Springer, Heidelberg, November 2017.

[39] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *ITCS 2014*, pages 1–12. ACM, January 2014.

[40] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 1–30. Springer, Heidelberg, March 2015.

[41] Zvika Brakerski and Vinod Vaikuntanathan. Circuit-abe from lwe: unbounded attributes and semi-adaptive security. In *Annual Cryptology Conference*, pages 363–384. Springer, 2016.

[42] B. Buchberger. A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.*, 10(3):19–29, August 1976.

[43] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 90–104. Springer, Heidelberg, August 1997.

[44] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996.

[45] Ran Canetti, Shafi Goldwasser, and Oxana Poburinnaya. Adaptively secure two-party computation from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 557–585. Springer, Heidelberg, March 2015.

[46] Angelo De Caro, Vincenzo Iovino, and Adam O'Neill. Deniable functional encryption. In *Public-Key Cryptography - PKC 2016 - 19th IACR International Conferenceon Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, pages 196–222, 2016.

[47] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 523–552. Springer, Heidelberg, May / June 2010.

[48] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. Deciding the security of protocols with diffie-hellman exponentiation and products in exponents. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS 2003: Foundations of Software Technology and Theoretical Computer Science: 23rd Conference, Mumbai, India, December 15-17, 2003. Proceedings*, pages 124–135, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[49] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.*, pages 271–280, June 2003.

[50] Pierre Corbineau, Mathilde Duclos, and Yassine Lakhnech. Certified security proofs of cryptographic protocols in the computational model: An application to intrusion resilience. In Jean-Pierre Jouannaud and Zhong

Shao, editors, *Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings*, volume 7086 of *Lecture Notes in Computer Science*, pages 378–393. Springer, 2011.

[51] Ricardo Corin and Jerry den Hartog. A probabilistic hoare-style logic for game-based cryptographic proofs. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2006.

[52] Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. Towards automated proofs for asymmetric encryption schemes in the random oracle model. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 371–380. ACM, 2008.

[53] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1773–1788. ACM, 2017.

[54] Dana Dachman-Soled, Jonathan Katz, and Vanishree Rao. Adaptively secure, universally composable, multiparty computation in constant rounds. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 586–613. Springer, Heidelberg, March 2015.

[55] Dana Dachman-Soled, Feng-Hao Liu, and Hong-Sheng Zhou. Leakage-resilient circuits revisited - optimal number of computing components without leak-free hardware. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 131–158. Springer, Heidelberg, April 2015.

[56] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. Implementing and proving the TLS 1.3 record layer. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 463–482. IEEE Computer Society, 2017.

[57] Apoorvaa Deshpande, Venkata Koppula, and Brent Waters. Constrained pseudorandom functions for unconstrained inputs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 124–153. Springer, 2016.

[58] 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.

[59] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, 29(2):198–207, 1983.

[60] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *Annual International Cryptology Conference*, pages 537–569. Springer, 2017.

[61] Daniel J. Dougherty and Joshua D. Guttman. Decidability for lightweight diffie-hellman protocols. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 217–231, 2014.

[62] David Eisenbud. *Commutative Algebra: with a view toward algebraic geometry*, volume 150. Springer Science & Business Media, 2013.

[63] Xiong Fan and Qiang Tang. Making public key functional encryption function private, distributively. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 218–244. Springer, Heidelberg, March 2018.

[64] Martin Gagné, Pascal Lafourcade, and Yassine Lakhnech. Automated security proofs for almost-universal hash for MAC verification. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 291–308. Springer, 2013.

[65] Martin Gagné, Pascal Lafourcade, Yassine Lakhnech, and Reihaneh Safavi-Naini. Automated security proof for symmetric encryption modes. In Anupam Datta, editor, *Advances in Computer Science - ASIAN 2009. Information Security and Privacy, 13th Asian Computing Science Conference, Seoul, Korea, December 14-16, 2009. Proceedings*, volume 5913 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2009.

[66] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[67] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

[68] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *Advances in Cryptology–CRYPTO 2013*, pages 479–499. Springer, 2013.

[69] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In *International Cryptology Conference*, pages 518–535. Springer, 2014.

[70] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure attribute based encryption from multilinear maps. Cryptology ePrint Archive, Report 2014/622, 2014. `http://eprint.iacr.org/2014/622`.

[71] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled ram. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 210–229. IEEE, 2015.

[72] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled ram from one-way functions. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 449–458. ACM, 2015.

[73] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 614–637. Springer, Heidelberg, March 2015.

[74] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 405–422. Springer, Heidelberg, May 2014.

[75] 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.

[76] 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.

[77] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Heidelberg, August 2013.

[78] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

[79] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.

[80] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. *Journal of the ACM (JACM)*, 62(6):45, 2015.

[81] 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.

[82] 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.

[83] 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.

[84] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. *Automata, languages and programming*, pages 579–591, 2008.

[85] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[86] 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.

[87] Viet Tung Hoang, Jonathan Katz, and Alex J. Malozemoff. Automated analysis and synthesis of authenticated encryption schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 84–95. ACM, 2015.

[88] Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 372–383. IEEE Computer Society, 2003.

[89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 12–24. ACM, 1989.

[90] Shuichi Katsumata and Shota Yamada. Partitioning via non-linear polynomial functions: More compact IBEs from ideal lattices and bilinear maps. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 682–712. Springer, Heidelberg, December 2016.

[91] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption sup-

porting disjunctions, polynomial equations, and inner products. *Advances in Cryptology–EUROCRYPT 2008*, pages 146–162, 2008.

[92] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, Jun 1994.

[93] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 435–450. IEEE, 2017.

[94] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 568–588. Springer, 2011.

[95] 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 *Eurocrypt*, volume 6110, pages 62–91. Springer, 2010.

[96] Andreas Lochbihler. Probabilistic functions and cryptographic oracles in higher order logic. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 503–531. Springer, 2016.

[97] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[98] Alex J. Malozemoff, Jonathan Katz, and Matthew D. Green. Automated analysis and synthesis of block-cipher modes of operation. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 140–152. IEEE Computer Society, 2014.

[99] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 465–484. Springer, Heidelberg, August 2011.

[100] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

[101] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM CCS 01*, pages 166–175. ACM Press, November 2001.

[102] Teo Mora. An introduction to commutative and noncommutative gröbner bases. *Theoretical Computer Science*, 134(1):131–173, 1994.

[103] Patrik Nordbeck. Canonical subalgebraic bases in non-commutative polynomial rings. In *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, ISSAC '98, pages 140–146, New York, NY, USA, 1998. ACM.

[104] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. `http://eprint.iacr.org/2010/556`.

[105] Adam O'Neill, Chris Peikert, and Brent Waters. Bi-deniable public-key encryption. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 525–542. Springer, Heidelberg, August 2011.

[106] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 195–203. ACM, 2007.

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

[108] Lawrence Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6, 12 2000.

[109] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009.

[110] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016.

[111] Adam Petcher and Greg Morrisett. The foundational cryptography framework. In Riccardo Focardi and Andrew C. Myers, editors, *Principles of Security and Trust - 4th International Conference, POST*, volume 9036 of *Lecture Notes in Computer Science*, pages 53–72. Springer, 2015.

[112] 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.

[113] Lorenzo Robbiano and Moss Sweedler. Subalgebra bases. In Winfried Bruns and Aron Simis, editors, *Commutative Algebra*, pages 61–87, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[114] M. Rusinowitch, R. Küsters, M. Turuani, and Y. Chevalier. An np decision procedure for protocol insecurity with xor. In *Logic in Computer Science, Symposium on(LICS)*, volume 00, page 261, 06 2003.

[115] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions and composed keys is np-complete. *Theoretical Computer Science*, 299(1):451 – 475, 2003.

[116] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[117] 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.

[118] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In Stephen Chong, editor, *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 78–94. IEEE Computer Society, 2012.

[119] Steve Schneider. Security properties and csp. In *Proceedings of the 1996 IEEE Conference on Security and Privacy*, SP'96, pages 174–187, Washington, DC, USA, 1996. IEEE Computer Society.

[120] David Shannon and Moss Sweedler. Using gröbner bases to determine algebra membership, split surjective algebra homomorphisms determine birational equivalence. *J. Symb. Comput.*, 6(2-3):267–273, December 1988.

[121] Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. Secure distributed programming with value-dependent types. *J. Funct. Program.*, 23(4):402–451, 2013.

[122] Eftychios Theodorakis and John C. Mitchell. Semantic security invariance under variant computational assumptions. *IACR Cryptology ePrint Archive*, 2018:51, 2018.

[123] Ashish Tiwari, Adrià Gascón, and Bruno Dutertre. Program synthesis using dual interpretation. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 482–497. Springer, 2015.

[124] Zhedong Wang, Xiong Fan, and Feng-Hao Liu. Fe for inner products and its application to decentralized abe. PKC, 2019.

[125] Brent Waters. Efficient identity-based encryption without random oracles. In *Eurocrypt*, volume 3494, pages 114–127. Springer, 2005.

[126] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *Crypto*, volume 5677, pages 619–636. Springer, 2009.

[127] Brent Waters. Functional encryption for regular languages. In *CRYPTO*, volume 7417, pages 218–235. Springer, 2012.

[128] Hoeteck Wee. Dual system encryption via predicate encodings. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 616–637. Springer, Heidelberg, February 2014.