Building an Efficient Lattice Gadget Toolkit: Subgaussian Sampling and More [*]

Nicholas Genise[†] Daniele Micciancio[‡] Yuriy Polyakov[§]

March 12, 2019

Abstract

Many advanced lattice cryptography applications require efficient algorithms for inverting the so-called "gadget" matrices, which are used to formally describe a digit decomposition problem that produces an output with specific (statistical) properties. The common gadget inversion problems are the classical (often binary) digit decomposition, subgaussian decomposition, Learning with Errors (LWE) decoding, and discrete Gaussian sampling. In this work, we build and implement an efficient lattice gadget toolkit that provides a general treatment of gadget matrices and algorithms for their inversion/sampling. The main contribution of our work is a set of new gadget matrices and algorithms for efficient subgaussian sampling that have a number of major theoretical and practical advantages over previously known algorithms. Another contribution deals with efficient algorithms for LWE decoding and discrete Gaussian sampling in the Residue Number System (RNS) representation.

We implement the gadget toolkit in PALISADE and evaluate the performance of our algorithms both in terms of runtime and noise growth. We illustrate the improvements due to our algorithms by implementing a concrete complex application, key-policy attribute-based encryption (KP-ABE), which was previously considered impractical for CPU systems (except for a very small number of attributes). Our runtime improvements for the main bottleneck operation based on subgaussian sampling range from 18x (for 2 attributes) to 289x (for 16 attributes; the maximum number supported by a previous implementation). Our results are applicable to a wide range of other advanced applications in lattice cryptography, such as GSW-based homomorphic encryption schemes, leveled fully homomorphic signatures, other forms of ABE, some program obfuscation constructions, and more.

1 Introduction

Many advanced applications of lattice cryptography require the generation of a random integer matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ (with uniform entries modulo q) together with a *strong trapdoor* (typically a short[*] basis $\mathbf{S}$ for the lattice defined by $\mathbf{A}$ as a parity check matrix). The strong trapdoor is used to efficiently "invert" the classical Short Integer Solution (SIS) and Learning with Errors (LWE) functions $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}$ and $g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t$ associated to the matrix $\mathbf{A}$. Theoretical solutions to these trapdoor generation and function inversion problems have long been known [1, 6, 7, 33, 25]. However, the trapdoor constructions of [1, 6] and generic inversion algorithms of [7, 33, 25] are rather complex, inefficient, and not suitable for practice.

An important step towards bringing advanced lattice-based cryptographic applications to practice was taken in [37], where a new notion of trapdoor (and associated generation algorithm) is proposed. The

[*] This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Army Research Laboratory (ARL) under Contract Numbers W911NF-15-C-0226 and W911NF-15-C-0233. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the Department of Defense or the U.S. Government.

[†] ngenise@eng.ucsd.edu, UCSD.

[‡] daniele@cs.ucsd.edu, UCSD.

[§] polyakov@njit.edu, NJIT.

[*] In the context of lattice cryptography, "short" typically means much smaller than the modulus q.

trapdoor of [37] transforms the problem of inverting the random functions $f_{\mathbf{A}}, g_{\mathbf{A}}$ to the problem of inverting the same type of functions $f_{\mathbf{G}}, g_{\mathbf{G}}$, but for a specific, carefully designed "gadget" matrix $\mathbf{G}$, which admits much simpler and faster inversion algorithms. Gadget matrices similar to the one of [37] had already been used in a number of previous works, starting from Ajtai's first construction of "solved instances of the shortest basis problem" [1], and including virtually all works on (fully) homomorphic encryption schemes based on the LWE problem (e.g., see [12, 26]); though different works use the gadget matrix for somehow different purposes. In fact, there are several inversion problems associated to the gadget matrix $\mathbf{G}$:

- **Digit Decomposition:** This is the problem of expressing an arbitrary vector $\mathbf{u} \in \mathbb{Z}_q^n$ as a short vector $\mathbf{x}$ such that $\mathbf{Gx} = \mathbf{u} \pmod q$. This is perhaps the most basic use of the gadget matrix $\mathbf{G}$, and plays an important role in the key-switching and multiplication operations of fully homomorphic encryption (FHE) schemes. For example, the binary decomposition gadget matrix $\mathbf{G} = [\mathbf{I}, 2\mathbf{I}, \ldots, 2^{k-1}\mathbf{I}]$ allows to write any vector with entries in $\mathbb{Z}_q$ as a combination $\sum_i 2^i \mathbf{x}_i$ of vectors $\mathbf{x}_i \in \{0,1\}^n$ with the $\{0,1\}$ coefficients corresponding to the binary digits of the entries in $\mathbf{u}$.

- **Subgaussian Decomposition:** This is a type of *randomized* digit decomposition, where a short vector $\mathbf{x}$ satisfying $\mathbf{Gx} = \mathbf{u} \pmod q$ is chosen according to a distribution with desirable statistical properties. This alternative to the standard binary decomposition was suggested in [5] as a method to improve the noise growth in homomorphic computations using variants of the GSW homomorphic encryption scheme [26], and is potentially applicable to the key-switching and homomorphic multiplication operations of many other FHE schemes.

- **LWE Decoding:** Given $\mathbf{s}^t\mathbf{G} + \mathbf{e}^t$ for a sufficiently small error vector $\mathbf{e}$, recover both $\mathbf{s}$ and $\mathbf{e}$. This is the (deterministic) inversion problem for the standard (injective) LWE function $g_{\mathbf{G}}$, which is used, for example, in the decryption algorithms of LWE-based cryptosystems.

- **Discrete Gaussian Sampling:** Produce a sample from a discrete Gaussian distribution over the set of all integer vectors $\mathbf{x}$ such that $f_{\mathbf{G}}(\mathbf{x}) = \mathbf{u}$. This problem was the main focus of [37], and is used, for example, in hash-and-sign lattice-based signatures and trapdoor delegation for identity-based encryption, among many other applications.

Very efficient gadget inversion algorithms were given in [37], but only for the Discrete Gaussian Sampling and LWE Decoding problems, and in the very special setting where the modulus $q = b^k$ is the power of a small base b. For the case of Discrete Gaussian Sampling, an equally efficient, but more general solution, was recently proposed in [23] for arbitrary modulus q, expanding the range of advanced lattice cryptography applications that admit a reasonably practical implementation. (E.g., see [19, 17, 30, 9, 29].) We remark that trapdoor inversion is the most complex operation in many applications of lattice cryptography, and effective solutions to gadget inversion play a critical role in determining the efficiency, quality and other performance characteristics of higher-level algorithms and the final applications.

The main focus of our work is Subgaussian Decomposition, a problem that has received little or no attention so far, and still has the potential to substantially improve the efficiency of many important applications. The importance of subgaussian sampling is easily explained by comparing it to the related problems of Digit Decomposition and Discrete Gaussian Sampling. We recall that lattice-based cryptography directly supports linear homomorphic operations, but ciphertexts are noisy, and their quality degrades when performing homomorphic operations: the noise of a sum $c_0 + c_1$ is the sum of the noise in the original ciphertexts c_0, c_1. More critically, when multiplying a ciphertext by a constant α, the noise scales by a factor α, which can be arbitrarily large. So, one needs to limit linear combinations to use only small coefficients. This is typically done using binary digit decomposition: given encryptions c_i of $2^i m$ (for $i = 0, \ldots, k-1$), one can compute an encryption of αm (for a large $\alpha < 2^k$) by taking a $0-1$ combination $\sum_i \alpha_i c_i$, where $\alpha = \sum_i 2^i \alpha_i$ is the binary representation of α. This way, the resulting noise scales linearly with $k = \log \alpha$, rather than α. Subgaussian decomposition allows to make the resulting noise even smaller: due to cancellations between randomly chosen coefficients, subgaussian decomposition has a "pythagorean additivity" property that makes the noise grow only as $O(\sqrt{k})$. The gain is even more substantial when adding many (say n) ciphertexts, in which case the noise growth is improved by a factor $\sqrt{nk}$. At the other end of the spectrum, pythagorean growth can also be achieved using Discrete Gaussian Sampling, as gaussian distributions are by definition also subgaussians. However, gaussian sampling is considerably more costly than digit decomposition, both in terms of running time, randomness and output quality: even with the improved algorithms of [23, 37], Discrete Gaussian Sampling is much more complex than a simple digit decomposition, and it necessarily produces "digits" (i.e., coefficients) that are larger than naive binary decomposition roughly by a factor $\Omega(\sqrt{\log k})$. The added algorithmic complexity and noise overhead make gaussian sampling unattractive in practice, and, perhaps not surprisingly, all implementations we are aware of use digit decomposition whenever possible.

Subgaussian decomposition has the potential to offer the best of both worlds: pythagorean additivity, but without the $\Omega(\sqrt{\log k})$ noise overhead of a full-blown discrete gaussian sampler. These potential advantages were already outlined in [5], but they were so far considered only of theoretical interest. In fact, none of the subsequent improvements and implementations [21, 15, 16, 39] make use of subgaussian sampling.

Our contribution The main contribution of our work is a set of new gadget matrices and algorithms for efficient subgaussian sampling. The improvements are not just theoretical/asymptotical, but very practical, as demonstrated by a concrete complex application: an implementation of a Key-Policy Attribute-Based Encryption (KP-ABE) scheme that speeds up previous implementation efforts by more than one order of magnitude. (See below and Sections 7 and 8 for details.) On the theoretical side, our algorithms and gadgets result in pythagorean error growth and optimal (essentially linear) time complexity. In practice, the algorithms are easy to implement and have very small hidden constants both in the number of operations they perform and the subgaussian parameters, offering a very attractive alternative to the naive deterministic digit decomposition methods currently used in the implementation of FHE and other related pritimives. Moreover, our gadgets and algorithms have a number of other useful properties that make them even more attractive in practice:

- All our algorithms require very little storage and only a modest (essentially optimal) amount of randomness. In particular, our gadget matrices have a very regular structure, and do not need to be explicitly stored.
- We support an arbitrary modulus q. This is not just of theoretical interest, as fast implementations of lattice cryptography [34, 36, 8, 32] require moduli of special form in order to make use of the Number Theoretic Transform (NTT).
- Our gadgets and algorithms support the "Full RNS" and "double CRT" techniques used to implement lattice cryptography with large modulus without the need for arbitrary-precision arithmetic libraries [24, 8, 32].

Beside subgaussian decomposition, we also provide very efficient algorithms for LWE Decoding and Discrete Gaussian Sampling that improve previous work [37, 23] by supporting arbitrary moduli and Full RNS implementations. (For Discrete Gaussian Sampling, algorithms supporting arbitrary moduli were already provided in [23], but for gadget matrices that do not support Full RNS implementations.)

Taken together, our algorithms provide a complete *lattice gadget toolkit*, offering efficient solutions to the full range of inversion problems encountered in lattice cryptography: Subgaussian Decomposition, LWE Decoding, and Discrete Gaussian Sampling. Our results are not just of theoretical interest, but are also relevant to the implementation and use of advanced lattice cryptography applications.

In order to demonstrate the applicability of our results, we developed an open-source optimized implementation of our algorithms and used it to implement a complex application: a key-policy attribute-based encryption scheme, which utilizes in a single setting most of the new algorithms developed in this paper. Our experimental results strongly support the theoretical analysis, showing both that (with proper algorithms and implementations) the advantages of pythagorean additivity clearly outweigh the modest increase in computational cost over naive bit decomposition, and that the overall impact on the performance of lattice applications can be substantial. Our CPU implementation of homomorphic access policy evaluation for keys and ciphertexts (the most expensive operation in KP-ABE schemes) outperforms the previous CPU implementation (for a comparable level of security) by a factor ranging from 18x to 289x as the number of policy attributes grows from 2 to 16. For higher numbers of attributes, previous CPU implementations were not feasible (only a GPU implementation is known), while we were able to run our implementation within reasonable running times for as many as 128 attributes. Other operations are also faster in our implementation, and memory requirements are also much smaller (by more than a factor of 2x in the simplest case of 2 attributes, and more than one order of magnitude at 16 attributes.) In summary, our results show that using our toolkit gadget inversion is no longer the bottleneck in efficient implementations of lattice cryptography, and it can be profitably used to achieve better performance and scalability both in theory and practice.

While in this paper we focused on the algorithmic core of a general gadget toolkit, and on a specific (but representative) application, our results are applicable to a wide range of other advanced applications in lattice cryptography. These include the use of subgaussian decomposition in GSW-based homomorphic encryption schemes [21, 16, 15], leveled fully homomorphic signatures [28], other forms of ABE [4], obfuscation of finite automata and branching programs using graph-induced encoding [31], and more.

Techniques Our efficient lattice gadget toolkit is based on better algorithmic solutions to known problems, but also on a new class of gadget matrices that enable our algorithmic improvements. While gadget matrices of the type used in [37] and our work are quite common in lattice cryptography, they have never been formally defined. In fact, as different applications and algorithms use the gadget matrices in somehow different ways, it was not even clear if one could meaningfully define gadget matrices as abstract mathematical objects, and most of previous works use the term “gadget” informally to identify specific constructions.

The starting point of our investigation is a simple, intuitive definition of gadget matrix, which turns out to be relevant to the solution of all algorithmic problems studied in this work. For any dimension n and modulus q (typically mandated by the application), a gadget of quality β is a matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times w}$ such that any $\mathbf{u} \in \mathbb{Z}_q^n$ can be represented as $\mathbf{Gx} = \mathbf{u}$ for some small integer vector $\mathbf{x} \in \mathbb{Z}^w$ of norm $\|\mathbf{x}\| \leq \beta$. This definition is directly motivated by the Digit Decomposition problem, but as we show in Section 3, it is already enough to enable theoretically efficient solutions to all of the algorithmic problems discussed above. (See Theorem 3.1 and Corollary 3.1.) The generic solutions obtained from Corollary 3.1 are not suited for practice, both in terms of algorithmic complexity and output quality. Still, a generic definition of gadget is useful to delimit a design space which extends well beyond the simplest (and perhaps most natural) construction of decomposition gadget $[1, 2, 2^2, \ldots, 2^{k-1}]$ corresponding to the the standard binary representation of a number as a sequence of bits. Other gadgets used in our work are digit decomposition gadgets $[1, b, b^2, \ldots, b^{k-1}]$ with a larger base $b > 2$, the CRT gadget $[g_1, \ldots, g_k]$ where $g_i = ((q/q_i)^{-1} \bmod q_i) \cdot (q/q_i)$ [11] (for composite moduli with relatively prime factorization $q = \prod_i q_i$) as well as hybrids between the two approaches where each g_i is replaced by an appropriate multiple of a vector of the form $[1, b, b^2, \ldots, b^{k-1}]$.

Our subgaussian decomposition algorithms use ideas and techniques from recent work on discrete gaussian sampling for arbitrary modulus [23]. In particular, we use the **SD** matrix factorization for the lattice defined by $\mathbf{G}$, and then perform subgaussian decomposition with respect to matrix $\mathbf{D}$, which is sparse and triangular, and admits much faster algorithms. A solution to the original problem is obtained using $\mathbf{S}$ as a linear transformation. Naturally, the details of our subgaussian decomposition algorithm for $\mathbf{D}$ are quite different from the algorithms in [23], as that paper solves a different problem (discrete gaussian sampling.) But, as our algorithm can sample an output vector $\mathbf{x}$ with respect to an arbitrary (not necessarily isotropic) subgaussian distribution, there is no need to apply gaussian correction terms (as done in [23]), and our algorithm is much simpler and more efficient. Our efficient LWE decoding algorithm is analogous. Specifically, the decoding problem can be seen as decoding an input to a lattice basis $q(\mathbf{SD})^{-t} = \mathbf{S}^{-t}(q \cdot \mathbf{D}^{-t})$. Now we can solve the decoding problem by first using $\mathbf{S}^t$ as a linear transformation, then by decoding the transformed input to the lattice generated by $q \cdot \mathbf{D}^{-t}$. This lattice is efficiently decodable since it, too, has a sparse, triangular basis. Our toolkit implementation focuses on providing full ring and RNS support for all gadget algorithms because ring multiplications can be efficiently computed via NTTs and large integer operations can be efficiently performed using native arithmetic in RNS. Full RNS / double CRT constructions based on power-of-two cyclotomic rings provide the best performance for the majority of known lattice cryptography primitives, as illustrated by our experimental results for subgaussian sampling and key-policy attribute-based encryption.

Related Work The first use of subgaussian decomposition appears in [5] in a theoretical form, not optimized for implementations. While the use of CRT gadgets for digit decomposition in the implementation of FHE schemes [8, 32] or even the foundation of Ring LWE [35] is not new, their applicability in the context of Gaussian or subgaussian sampling is, to the best of our knowledge, novel. Our CRT gadget algorithms can be seen as an extension of [37, 23, 11]. The CRT-like gadget proposed in [31] can be considered as a special case of ours when $b_i = p_i$ (assuming that $q_i = p_i^e$), which implies the gagdet noise width is larger than p_i. In our CRT gadget, b_is can be chosen independently from CRT moduli q_i, enabling significantly more efficient implementations in the ring setting. Another related work is a deterministic, balanced digit decomposition for the "double CRT/RNS" gadget [24] in the "LoL" library [18, 40] (initially unknown to the authors).

In the ring setting, one method [20] to achieve better-than-generic, $n \log^2 q$, efficiency of "power-of-b" gadget discrete Gaussian sampling is to use the FFO style of discrete Gaussian samplers from [22] and [23, Section 4]. This incurs a logarithmic slowdown, $\log n$, in time and space compared to using [23, Section 3] on the coefficients independently, which has $n \log q$ time and space efficiency. Further, Section 6 of this work extends [23, Section 3] to the "double-CRT" setting [24], freeing implementations using discrete Gaussian gadget sampling from multi-precision numbers when the modulus is over 64 bits.

Organization The rest of the paper is organized as follows. In Section 2 we review some preliminary material. In Section 3 we present our general definition of gadget matrices. Next, in Section 4 and Section 5 we present our core gadgets and algorithms for subgaussian decomposition and LWE decoding with arbitrary modulus. In Section 6 we extend these algorithms to large composite moduli to allow efficient operations in CRT form without the need of multiprecision integer arithmetic. Section 7 and 8 we present our implementation and experimental results. The generic subgaussian version of Babai's nearest plane is described in the Appendix.

2 Preliminaries

We indicate numbers with lowercase letters, such as $z \in \mathbb{Z}$, vectors as bold lowercase letters, $\mathbf{z} \in \mathbb{Z}^n$, and matrices as uppercase bold letters, $\mathbf{M} \in \mathbb{R}^{n \times n}$. The default norm used is the l_2 norm of a vector unless stated otherwise, though we will often use the max, or l_∞, norm. For a real number r, denote $\lceil r \rfloor$ as the deterministic rounding function to a nearest integer of r. Rounding a real vector is applied analogously, entry-wise. Many computations will be done over the integers modulo q, $\mathbb{Z}_q$. We view $\mathbb{Z}_q$ through its balanced coset representatives in $(-q/2, q/2]$ unless stated otherwise. For a positive integer base b and a non-negative integer $u < b^k$, u's b-ary decomposition is a vector $[u]_b^k = (u_0, \cdots, u_{k-1}) \in \{0, \cdots, b-1\}^k$ and satisfies $\sum_i b^i u_i = u$. When $b = 2$, this is simply u's binary decomposition. Recall the Chinese Remainder Theorem for modular arithmetic. Let q be a positive integer with a prime factorization of $q = p_1^{e_1} \cdots p_l^{e_l} = q_1 \cdots q_l$. Then by the Chinese Remainder Theorem (CRT), we have $\mathbb{Z}_q \cong \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_l}$. The isomorphism $\phi(\cdot)$ is given by $\phi(a) = (a \mod q_1, \cdots, a \mod q_l)$ and its inverse is $\phi^{-1}(a_1, \cdots, a_l) = \sum_i (a_i) q_i^* \hat{q}_i$ where $q_i^* := \frac{q}{q_i}$ and $\hat{q}_i := (q_i^*)^{-1} \mod q_i$.

For a probability distribution χ, we denote $e \leftarrow \chi$ to mean e is sampled from χ. When χ is trivial (often over a number x), we will use $e \leftarrow x$ to be variable assignment as well. We will need the following, known as the *Geršgorin Circle Theorem.*

Theorem 2.1 (Geršgorin) *Let $\mathbf{M}$ be an $n \times n$ matrix with complex entries. For each row i, let r_i be the sum of its non-diagonal entries' magnitudes: $r_i = \sum_{j \neq i} |\mathbf{M}(i,j)|$. Then, the eigenvalues of $\mathbf{M}$ are all in $\bigcup_i \{z \in \mathbb{C} : |z - M(i,i)| \leq r_i\}$.*

2.1 Subgaussian Random Variables

A random variable X over $\mathbb{R}$ is *subgaussian* [35, 42] with parameter $\alpha > 0$ if its (scaled) moment generating function satisfies $\mathbb{E}[\exp(2\pi t X)] \leq \exp(\pi \alpha^2 t^2)$ for all $t \in \mathbb{R}$. Scaling a subgaussian X by any $c \in \mathbb{R}$ to $c \cdot X$ yields a subgaussian random variable with parameter $|c|\alpha$. If X is subgaussian with parameter α, then its tails are dominated by a Gaussian parameterized by α, $\Pr\{|X| \geq t\} \leq 2\exp(-\pi t^2/\alpha^2)$. Any B-bounded centered ($\mathbb{E}[X] = 0$) random variable X is subgaussian with parameter $B\sqrt{2\pi}$. When X is subgaussian with parameter α and Y conditioned on X taking any value is subgaussian with parameter β, $X + Y$ is subgaussian with parameter $\sqrt{\alpha^2 + \beta^2}$. This property is called *Pythagorean additivity.* The proof of the following Lemma is derived by expanding $\mathbb{E}[\exp(2\pi t(X + Y))]$.

Lemma 2.1 *Let X, Y be discrete random variables over $\mathbb{R}$ such that X is subgaussian with parameter α and Y conditioned on X taking any value is subgaussian with parameter β. Then, $X + Y$ is subgaussian with parameter $\sqrt{\alpha^2 + \beta^2}$.*

Proof: Expanding the moment generating function gives the result:

$$\begin{aligned}\mathbb{E}[\exp(2\pi t(X+Y))] &= \sum_z \sum_\chi \Pr\{Y = z - \chi | X = \chi\} \Pr\{X = \chi\} \exp(2\pi t z) \\ &= \sum_\chi \Pr\{X = \chi\} \exp(2\pi t \chi) \mathbb{E}[\exp(2\pi t Y) | X = \chi] \\ &\leq \exp(\pi t^2(\alpha^2 + \beta^2)).\end{aligned}$$

□

A random vector $\mathbf{x}$ over $\mathbb{R}^n$ is *subgaussian* with parameter $\alpha > 0$ if $\langle \mathbf{x}, \mathbf{u} \rangle$ is subgaussian with parameter α for all unit vectors $\mathbf{u}$. Using a similar calculation to the above, one can show that if each coefficient of a random vector is subgaussian with parameter α conditioned on the previous coefficients taking any values, then the vector is subgaussian with parameter α. The slightly more general fact below is needed for our algorithms. Its proof is analogous to the proof of Lemma 2.1.

Lemma 2.2 *Let $\mathbf{x}$ be a discrete random vector over $\mathbb{R}^n$ such that each coordinate x_i is subgaussian with parameter α_i given the previous coordinates take any values. Then, $\mathbf{x}$ is a subgaussian vector with parameter $\max_i\{\alpha_i\}$.*

Proof: As before, we expand the moment generating function:

$$\begin{aligned}\mathbb{E}[\exp(2\pi t \langle \mathbf{x}, \mathbf{u} \rangle)] &= \sum_\chi \Pr\{\mathbf{x} = (\chi_1, \cdots, \chi_n)\} \exp(2\pi t \langle \chi, \mathbf{u} \rangle) \\ &\leq \exp(\pi t^2 \sum_i \alpha_i^2 u_i^2) \\ &\leq \exp(\pi t^2 \max_i {\alpha_i}^2 \|\mathbf{u}\|^2).\end{aligned}$$

The jump to the inequalities skips the straightforward calculations (nearly the same calculations as in Lemma 2.1). □

We emphasize this fact, for without it one is left with an unnecessary $\sqrt{n}$ term in the subgaussian parameter of subgaussian vectors. Now, that the sum of independently generated random vectors $\mathbf{x}$ and $\mathbf{y}$ subgaussian with parameters α and β is a subgaussian vector with parameter $\sqrt{\alpha^2 + \beta^2}$ immediately follows.

A main algorithm presented in this paper will rely on a linear transformation of a discrete subgaussian vector.

Lemma 2.3 (Simplified [35, Corollary 2.3]) *Let $\mathbf{x}$ be a subgaussian random vector with parameter α and let $\mathbf{M}$ be a linear transformation. Then, $\mathbf{Mx}$ is a subgaussian vector with parameter $\alpha\lambda_{max}(\mathbf{M}\mathbf{M}^T)^{1/2}$ where $\lambda_{max}(\cdot)$ is the largest eigenvalue.*

2.2 Lattices

A lattice is a discrete subgroup of $\mathbb{R}^n$. Equivalently, a lattice Λ can be represented as the set of all integer combinations of a basis $\mathbf{B} = [\mathbf{b}_1, \cdots, \mathbf{b}_k] \in \mathbb{Z}^{n\times k}$, $\Lambda = \{\sum_1^k z_i \mathbf{b}_i : z_i \in \mathbb{Z}\} = \mathcal{L}(\mathbf{B})$. Notice that any permutation of basis vectors is another lattice basis. We only consider full-rank lattices ($k = n$). A lattice is an integer lattice if it is a sublattice of $\mathbb{Z}^n$. The dual lattice of Λ, denoted as Λ^*, is the set $\Lambda^* = \{\mathbf{z} \in \mathbb{R}^n : \langle \mathbf{z}, \Lambda\rangle \subseteq \mathbb{Z}\}$. Given a basis $\mathbf{B}$ for Λ, its dual basis is $\mathbf{B}^{-t}$ which is also a basis for Λ^*. We will consider direct sums of lattices, $\Lambda = \Lambda_1 \oplus \cdots \oplus \Lambda_l$ and their dual lattices $\Lambda^* = \Lambda_1^* \oplus \cdots \oplus \Lambda_l^*$. The number $\lambda_i(\Lambda)$ is the radius of the smallest ball containing i linearly independent lattice vectors.

Given a basis $\mathbf{B} = [\mathbf{b}_1, \cdots, \mathbf{b}_n]$ for a lattice Λ, its Gram-Schmidt orthogonalization (GSO) is the set of vectors $\widetilde{\mathbf{B}} = [\widetilde{\mathbf{b}}_1, \cdots, \widetilde{\mathbf{b}}_n]$ where $\widetilde{\mathbf{b}}_i$ is the component of $\mathbf{b}_i$ orthogonal to span$(\mathbf{b}_1, \cdots, \mathbf{b}_{i-1})$. The GSO is not another basis for the lattice in general, but it gives us a tiling of $\mathbb{R}^n$ given by $\mathbb{R}^n = \cup_{\mathbf{x}\in\Lambda}(\mathbf{x} + \mathcal{P}_{1/2}(\widetilde{\mathbf{B}}))$ where $\mathcal{P}_{1/2}(\widetilde{\mathbf{B}}) := \widetilde{\mathbf{B}} \cdot (-1/2, 1/2]^n$. Note that the GSO depends on the order of the vectors given. We define the reverse order GSO analogously. The algorithms presented in this paper will all be instantiations of Babai's greedy decoding algorithm known as the *nearest plane algorithm* [7].

Theorem 2.2 *There is an algorithm which given $\mathbf{B}, \widetilde{\mathbf{B}}, \mathbf{t} \in \mathbb{R}^n$ returns the unique lattice point in $\mathbf{t}+\mathcal{P}_{1/2}(\mathbf{B}^*)$ in time $O(n^2)$ and memory $O(n^3)^\dagger$.*

Discrete Gaussians Let $A \subset \mathbb{R}^n$ be a discrete set, and let the (spherical) Gaussian function with width s and center $\mathbf{c} \in \mathbb{R}^n$ be $\rho_{s,\mathbf{c}}(\mathbf{x}) = \exp(-\pi\|\mathbf{x}-\mathbf{c}\|^2/s^2)$. Let $\rho_{s,\mathbf{c}}(A) = \sum_{\mathbf{y}\in A} \rho_{s,\mathbf{c}}(\mathbf{y})$. The smoothing parameter of a lattice [38] for some $\varepsilon > 0$, is dentoted as $\eta_\varepsilon(\Lambda)$, and it is defined as the minimum $s > 0$ such that $\rho(s \cdot \Lambda^*) \leq 1+\epsilon$. When $s = 1$ and $\mathbf{c} = \mathbf{0}$, we denote this as $\rho(\cdot)$. Then, the discrete Gaussian distribution has probability $\rho_{s,\mathbf{c}}(\mathbf{x})/\rho_{s,\mathbf{c}}(A)$ for each $\mathbf{x} \in A$. This distribution is denoted as $D_{A,s,\mathbf{c}}$. Polynomial time discrete Gausisan sampling algorithms for general lattices and their cosets, with width above the GSO length of the input basis (times a small factor, $\omega(\sqrt{\log n})$ or $O(\sqrt{\log n})$), are given in [25, 13].

q-ary Lattices Throughout this paper we will mostly be concerned with *q-ary lattices.* These are full-rank integer lattices with $q \cdot \mathbb{Z}^k$ as a sublattice. Fix an integer $q > 0$ to be used as a modulus and let $m > w > n$. A matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ is *primitive* if $\mathbf{A}\mathbb{Z}_q^m = \mathbb{Z}_q^n$. Given an $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, we define the following lattices: $\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{z} \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{0} \mod q\}$, and $\Lambda_q(\mathbf{A}) = \{\mathbf{v} \in \mathbb{Z}^m : \exists\ \mathbf{s} \in \mathbb{Z}^n,\ \mathbf{v}^t = \mathbf{s}^t\mathbf{A} \mod q\}$. These lattices satisfy the following duality relation: $\Lambda_q^\perp(\mathbf{A})^* = q \cdot \Lambda_q(\mathbf{A})$. Further, the cosets of $\Lambda_q^\perp(\mathbf{A})$, $\Lambda_\mathbf{u}^\perp(\mathbf{A}) := \{\mathbf{z} \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{u} \mod q\}$, are in bijection with $\mathbb{Z}_q^n$ when $\mathbf{A}$ is primitive. Let $\mathbf{G}$ be an arbitrary, primitive matrix over $\mathbb{Z}_q$. The following sampling problem, defined on the integer cosets of $\Lambda_q^\perp(\mathbf{G})$, is needed for many advanced lattice crypto-schemes.

Definition 2.1 *For a primitive $\mathbf{G} \in \mathbb{Z}_q^{n\times w}$, the subgaussian decomposition problem with parameter α for $\mathbf{G}$ is to sample vectors $\mathbf{x} \in \mathbb{Z}^w$ subgaussian with parameter α such that $\mathbf{u} = \mathbf{G}\mathbf{x} \mod q$ for arbitrary $\mathbf{u}$ given as input.*

Another name for this problem is subgaussian sampling. A generic adaptation of Babai's algorithm (analyzed in the Appendix, called the subgaussian nearest plane algorithm) is used in [5] (AP14) to achieve subgaussian decomposition for a specific $\mathbf{G}$. In general, this generic algorithm runs in time $O(k^2)$, and uses space $O(k^3)$. Another, related problem is the discrete Gaussian sampling problem.

Definition 2.2 *For a primitive $\mathbf{G} \in \mathbb{Z}_q^{n\times w}$, the discrete Gaussian sampling problem with width s for $\mathbf{G}$ is to sample vectors $\mathbf{x} \in \mathbb{Z}^w$ distributed as $D_{\mathbb{Z}^w,s}$ conditioned on $\mathbf{G}\mathbf{x} \mod q = \mathbf{u}$ for arbitrary $\mathbf{u}$ given as input.*

Efficient solutions with small s for commonly used $\mathbf{G}$'s are given in [37, 23]. Both of the above sampling problems have polynomial time solutions using randomized versions of Babai's algorithm. In addition, we will consider decoding the q-ary code defined by $\mathbf{G}$ for an arbitrary, primitive $\mathbf{G}$.

†This assumes the GSO has entries each described in $O(n)$ bits

Definition 2.3 *For a primitive* $\mathbf{G} \in \mathbb{Z}_q^{n\times w}$*, the LWE decoding problem with tolerance* δ *on* $\mathbf{G}$ *is to return* $\mathbf{s}$ *given* $\mathbf{s}^t\mathbf{G} + \mathbf{e}^t \mod q$ *for an error* $\|\mathbf{e}\|_\infty < \delta$*.*

Specifically, we want to efficiently decode $\mathbf{G}$ while maximizing $\delta \in [0, q/2)$. An efficient LWE decoding algorithm for a specific, commonly used $\mathbf{G}$ ($b = 2$ in the paragraph below) with tolerance $q/4$ is provided in [37].

A $\mathbf{G}$ commonly used in lattice crypto-schemes is defined as follows. Fix an integer $b \in (1, q)$, known as the *base*, and let $k = \lceil \log_b q \rceil$. The block-diagonal gadget matrix is $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g}^t$ with blocks $\mathbf{g}^t := (1, b, \cdots, b^{k-1})$. A common basis for $\Lambda_q^\perp(\mathbf{g}^t)$ [23] $\mathbf{S}_q$ has a sparse, triangular factorization $\mathbf{S}_q = \mathbf{SD}$ [23] (restated in Section 4.2 in this paper).

3 Gadget Matrices

In order to guide our search for gadget matrices with efficient inversion and sampling algorithms, we give a simple general definition of gadget. The definition is modeled after the properties required by the digit decomposition problem, perhaps the simplest and most natural application of gadgets. But, as we will see, this simple characterization is enough to guarantee (theoretical) solutions to all problems that arise in the application of gadgets in lattice cryptography.

Definition 3.1 *For any finite additive group* A*, an* A*-gadget of size* w *and quality* β *is a vector* $\mathbf{g} \in A^w$ *such that any group element* $u \in A$ *can be written as an integer combination* $u = \sum_i g_i \cdot x_i$ *where* $\mathbf{x} = (x_1, \ldots, x_w)$ *has norm at most* $\|\mathbf{x}\| \le \beta$*.*

We are primarily interested in gadgets for $A = \mathbb{Z}_q^n$, in which case the gadget is conveniently represented as a matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times w}$ such that for any $\mathbf{u} \in \mathbb{Z}_q^n$ there is a vector $\mathbf{x} \in \mathbb{Z}^w$ of length $\|\mathbf{x}\| \le \beta$ such that $\mathbf{Gx} = \mathbf{u}$ (mod q). We defined gadgets in terms of abstract groups to emphasize that the dimension n and modulus q should be thought of as part of the problem specification (typically mandated by the target application), while the w and β describe the size and quality of the solution. In particular, for any given n and q, one may consider multiple gadgets achieving different values of w and β. Naturally, smaller w and β are preferable, but as we will see there is a natural tradeoff between these two values, and one may increase β in order to reduce w and vice versa.

Before establishing a formal connection between the above definition and the notion of gadget informally defined in previous work, we make some important observations.

- The matrix $\mathbf{G}$ is necessarily primitive, i.e., $\mathbf{G}\mathbb{Z}_q^w = \mathbb{Z}_q^n$. Moreover, any primitive matrix is a $\mathbb{Z}_q^n$-gadget for a sufficiently large $\beta = \max_{\mathbf{u}} \min\{\|\mathbf{x}\| : \mathbf{Gx} = \mathbf{u} \pmod q\}$.
- If $\mathbf{g} \in \mathbb{Z}^k$ is a $\mathbb{Z}_q$-gadget of quality β, then $\mathbf{G} = \mathbf{I} \otimes \mathbf{g}^t \in \mathbb{Z}_q^{n\times w}$ is a $\mathbb{Z}_q^n$-gadget of size $w = kn$ and quality $\sqrt{n}\beta$.
- All definitions and constructions are easily adapted to ideal lattices (as used in the Ring-SIS and Ring-LWE problems) simply by considering "structured gadgets" of the form $\mathbf{G} \otimes [\alpha_1, \ldots, \alpha_n]$ where $[\alpha_1, \ldots, \alpha_n]$ is an appropriate $\mathbb{Z}$-basis of the underlying ring.

Based on the above observations, constructions may focus on the case $n = 1$, i.e., gadget vectors $\mathbf{g} \in \mathbb{Z}_q^w$, and then extend the solution to larger n (and possibly to the ring setting) using general techniques. In fact, this is how larger gadgets are built in all applications we are aware of. However, all the results in this section hold for arbitrary matrices, not necessarily with this tensor structure. So, for the sake of generality, we use matrix notation.

In order to justify our abstract definition of gadget, we show that it guarantees all other properties of gadgets used by lattice cryptography: it maps the gaussian distribution to an almost uniform vector $\mathbf{G}D_{\mathbb{Z},s}^w \approx \mathbb{Z}_q^n$ (as needed by the trapdoor generation algorithm of [37]), and it supports efficient algorithms to invert the LWE function $g_{\mathbf{G}}(\mathbf{x}, \mathbf{e})$, for discrete gaussian sampling on $f_{\mathbf{G}}^{-1}(\mathbf{u})$, and for subgassian decomposition with respect to $\mathbf{G}$. All these properties are proved by bounding the relevant parameters of the lattice $\Lambda_q^\perp(\mathbf{G})$ defined by $\mathbf{G}$.

Theorem 3.1 *For any gadget matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times w}$ of quality β, the lattice $L = \Lambda_q^\perp(\mathbf{G})$ has a basis $\mathbf{S}$ with orthogonalized length $\|\tilde{\mathbf{S}}\| \leq 2\beta + \sqrt{w}$, successive minima $\lambda_1(L), \ldots, \lambda_w(L) \leq 2\beta + \sqrt{w}$ and smoothing parameter $\eta(L) \leq (2\beta + \sqrt{w})\omega(\sqrt{\log n})$.*

Proof: We first bound the covering radius $\mu(L)$. Let $\mathbf{x} \in \mathbb{R}^w$ be arbitrary, and let $\mathbf{y} = \lfloor \mathbf{x} \rceil$ be a nearest point in $\mathbb{Z}^w$ to $\mathbf{x}$. There exists some integer vector $\mathbf{z}$ of norm at most β such that $\mathbf{Gz} = -\mathbf{Gy} \mod q$. Therefore, the vector $\mathbf{y}+\mathbf{z}$ is in $\Lambda_q^\perp(\mathbf{G})$ and is at distance at most $\beta + \sqrt{w}/2$ from $\mathbf{x}$ by two applications of the triangle inequality.

The other bounds immediately follow from general relations (satisfied by any lattice) $\lambda_w(L) \leq 2\mu(L)$ and $\eta(L) \leq \lambda_n(L)\omega(\sqrt{\log n})$. Finally, any lattice has a basis with orthogonalized length $\|\tilde{\mathbf{S}}\| \leq \lambda_w(L)$. □

Note, the proof and theorem easily generalizes to any finite abelian group. Using the bound on the smoothing parameter, and the short (orthogonalized) basis $\mathbf{S} \in \mathbb{Z}^{w\times w}$, we immediately get the following applications. (E.g., for the subgaussian decomposition algorithm see the Appendix.)

Corollary 3.1 *For any gadget matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times w}$ of quality β and $s \geq (2\beta + \sqrt{w})\sqrt{\omega(\log n)}$, the distribution $\mathbf{G}D_{\mathbb{Z},s}^w$ is statistically close to the uniform distribution over $\mathbb{Z}_q^n$. Moreover, there are polynomial-time algorithms for the following problems:*

- *Discrete Gaussian Sampling for the function $f_{\mathbf{G}}(\mathbf{x}) = \mathbf{Gx} \pmod q$ and input distribution $D_{\mathbb{Z},s}^w$ with $s \geq (2\beta + \sqrt{w})\sqrt{\omega(\log n)}$.*
- *Subgaussian Decomposition w.r.t $\mathbf{G}$ with parameter $s \geq (2\beta + \sqrt{w}) \cdot \sqrt{2\pi}$.*
- *LWE decoding of $g_{\mathbf{G}}(\mathbf{s}, \mathbf{e})$ for any $\mathbf{s} \in \mathbb{Z}_q^n$ and $\|\mathbf{e}\|_\infty \leq q/2 \cdot (2\beta + \sqrt{w})$.*

We remark that the general solutions provided by this corollary are of theoretical interest, and not suitable for practice. They are provided here only as a general feasibility result, in order to identify classes of good gadget matrices. The rest of the paper is dedicated to showing that by carefully choosing the gadget vector $\mathbf{g}$, one can obtain constructions and algorithms that are not only theoretically efficient, but also easy to implement and extremely fast.

4 Subgaussian Gadget Decomposition

In this section we present our main algorithms for the problem of *subgaussian gadget decomposition*, defined in Section 2.2, using the gadget matrix $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g}^t$. Since this decomposition $\mathbf{G}^{-1}(\mathbf{u}) = (\mathbf{g}^{-1}(u_i))_{i=1}^n$ can be computed one component at a time (even in-parallel!) we restrict our attention to efficiently computing the subgaussian function $\mathbf{g}^{-1} : \mathbb{Z}_q \to \mathbb{Z}^k$ in the one-dimensional case, i.e., for $n = 1$.

The gadgets and algorithms in this section are parametrized by a "base" integer b, which we consider as fixed throughout the section, but can be used to achieve different efficiency/quality trade-offs. We distinguish two cases, depending on whether the modulus is a power $q = b^k$ of the base b, or an arbitrary integer $q < b^k$. In either case, no assumption is made about the factorization of the modulus q. Later, in Section 6, we will extend the gadgets and algorithms from this section to provide optimized treatment of large moduli with useful co-prime factorization $q = \prod_i q_i$, where the input $u \in \mathbb{Z}_q$ is given in CRT form ($u \bmod q_1, \ldots, u \bmod q_l$).

All algorithms in this section use the same gadget $\mathbf{g}^t := (1, b, \cdots, b^{k-1})$, for $k = \lceil \log_b q \rceil$, but with different subgaussian decomposition procedures depending on the whether q is a power of b. Notice that $\mathbf{g}^t$ is a $\mathbb{Z}_q$-gadget of size k and quality $\beta = \sqrt{k}(b/2)$.

The main result of this section is summarized in the following theorem.[‡]

Theorem 4.1 *For any integer base $b > 1$, integer modulus $q > 1$, $k = \lceil \log_b q \rceil$ and gadget $\mathbf{g}^t = [1, b, \cdots, b^{k-1}]$, there is a subgaussian decomposition algorithm $\mathbf{g}^{-1}$ as follows:*

- *If $q = b^k$, the algorithm runs in linear $O(k)$ time (and space), uses $\log_2 q$ random bits, and achieves subgaussian parameter at most $(b-1)\sqrt{2\pi}$.*
- *If $q \neq b^k$, the algorithm runs in linear $O(k)$ time (and space), uses at most $k \log_2 q$ random bits, and achieves subgaussian parameter at most $(b+1)\sqrt{2\pi}$,*

Notice how the generic solution obtained by applying Theorem 3.1 to our gadget $\mathbf{g}$ only implies a polynomial time inversion algorithm with subgaussian parameter $(b+1)\cdot\sqrt{2k\pi}$, and quadratic $O(k^2)$ time complexity (after a cubic time $O(k^3)$ preprocessing). Depending on implementation details, this generic solution would also require the use of high precision floating point numbers[§] and a substantial amount of randomness for high precision sampling. (For completeness, we provide a more detailed analysis of the generic solution in the Appendix.) By contrast, the solution described in Theorem 4.1 is much more efficient (linear time and space, with no need for preprocessing) and also achieves a smaller subgaussian parameter by a factor of $\sqrt{k}$. Moreover, our specialized algorithms use a relatively small (almost optimal) number of random bits, and can be implemented without the need for high-precision floating-point arithmetic.

A proof of Theorem 4.1 is given by the algorithms presented and analyzed in the next two subsections for the two separate cases $q = b^k$ and $q < b^k$.

4.1 Power-of-Base Case

Here we consider the subgaussian decomposition problem for the gadget $\mathbf{g} = (1, b, \ldots, b^{k-1})$ when $q = b^k$, and the input is given as a positive coset representative $u \in \{0, 1, \cdots, q-1\}$. Conceptually, our solution to this problem is just a specialized/optimized version of the randomized-rounding variant of Babai's nearest plane algorithm [7, 5]. The general algorithm uses the Gram-Schmidt orthogonalization of a basis for the lattice $\Lambda_q^\perp(\mathbf{g}^t)$ associated to the gadget $\mathbf{g}$. The optimization is based on the observation (from [37]) that for our gadget $\mathbf{g}$ and modulus $q = b^k$, the lattice $\Lambda_q^\perp(\mathbf{g}^t)$ has a very simple basis $\mathbf{S}$, and an even simpler GSO $\tilde{\mathbf{S}}$:

$$\mathbf{S} = \begin{pmatrix} b & & & \\ -1 & \ddots & & \\ & \ddots & b & \\ & & -1 & b \end{pmatrix}, \qquad \widetilde{\mathbf{S}} = b \cdot \mathbf{I}.$$

Using this special structure, there is no need to explicitly compute and store the GSO, and the randomized-rounding nearest-plane algorithm can be implemented in linear time and space $O(k)$. The specialized algorithm is best illustrated when $b = 2$, in which case it computes a randomized "bit" decomposition of u as follows:

1. For $i = 0, \cdots, k-1$:

   (a) if u is even, then set $x_i \leftarrow 0$,

   (b) if u is odd, then choose $x_i \leftarrow \{-1, +1\}$ uniformly at random

   Update $u \leftarrow (u - x_i)/2$.

[‡] This theorem is most relevant when q is a relatively small modulus (say $q < 2^{64}$), so that arithmetic operations modulo q can be performed with unit cost. For larger moduli, the theorem will be used as a building block for a more complex algorithm described in Section 6 using RNS/CRT representation for the elements of $\mathbb{Z}_q$.

[§] For a general integer basis $\mathbf{B}$, the GSO can have numbers with denominators as large as $\prod_i \|\mathbf{b}_i\|^2$.

```
Algorithm 1: $\mathbf{g}^{-1}(u)$ for $q = b^k$.
  Input: $u \in \{0, 1, \cdots, q-1\}$
  Output: subgaussian $\mathbf{x} \in \Lambda_u^{\perp}(\mathbf{g}^t)$ with parameter $(b-1)\sqrt{2\pi}$
    Let $\mathbf{x} \leftarrow \mathbf{0}$
    for $i \leftarrow 0, \cdots, k-1$ do
        Let $y \leftarrow u \mod b \in \{0, \cdots, b-1\}$.
        if $y = 0$ then
            $x_i \leftarrow 0$.
        else
            with probability $y/b$, $x_i \leftarrow y - b$, and $x_i \leftarrow y$ otherwise.
        $u \leftarrow (u - x_i)/b$.
    return $\mathbf{x}$
```

2. Return $\mathbf{x} = (x_0, x_1, \cdots, x_{k-1})$.

This is essentially the same as the standard (deterministic) bit decomposition algorithm, except that when the bit is 1, we use a random ±1 digit. Since ±1 have the same parity modulo 2, the algorithm works as expected, with the only difference that now each digit is a zero-mean random variable, and the final output is subgaussian with parameter $\sqrt{2\pi}$.

We can modify this algorithm to an arbitrary base b as follows. Let $y := u \mod b \in \{0, \cdots, b-1\}$ for an input $u \in \mathbb{Z}_q$. Then, at each step, we pick the coset representative (of u with respect to $\mathbb{Z}_b$) with expectation 0 from the set $\{y - b, y\}$. The resulting algorithm is given in Figure 1. One can verify that this is the subgaussian nearest plane algorithm (given in the Appendix) applied to the lattice $\mathcal{L}(\mathbf{S}) = \Lambda_q^{\perp}(\mathbf{g}^t)$, so the correctness of the algorithm is straightforward. Efficiency is also easily analyzed by inspection. Notice that the algorithm is randomness efficient as it needs only one random number in $\mathbb{Z}_b$ for every interation, for a total of $k \cdot \log_2(b) = \log_2(q)$ random bits.

We remark that a similar algorithm is analyzed in [4], though with a loose bound on its subgaussian parameter (there is an unnecessary $\sqrt{k}$ factor in their subgaussian analysis). This section's main contribution is how to generalize the algorithm to arbitrary modulus q, as described in the next subsection.

4.2 Arbitrary Modulus, Arbitrary Base

Unfortunately, the (randomized) nearest plane algorithm $\Lambda_q^{\perp}(\mathbf{g}^t)$ does not specialize well when the modulus q is not a power of b. The reason is that, while we can still use the same gadget $\mathbf{g} = (1, b, \ldots, b^{k-1})$, the corresponding lattice $\Lambda_q^{\perp}(\mathbf{g}^t)$ has a slightly different basis $\mathbf{S}_q$ whose GSO is not diagonal, and not sparse. Our solution uses a technique developed in [23] for the discrete Gaussian sampling problem. Specifically, we use the fact that $\mathbf{S}_q$ admits a sparse, triangular factorization

$$\mathbf{S}_q = \begin{pmatrix} b & & & q_0 \\ -1 & \ddots & & \vdots \\ & \ddots & b & q_{k-2} \\ & & -1 & q_{k-1} \end{pmatrix} = \begin{pmatrix} b & & \\ -1 & \ddots & \\ & \ddots & b \\ & -1 & b \end{pmatrix} \begin{pmatrix} 1 & & d_0 \\ & \ddots & \vdots \\ & 1 & d_{k-2} \\ & & d_{k-1} \end{pmatrix} = \mathbf{SD} \tag{1}$$

where $(q_0, \cdots, q_{k-1})$ are the (base b) digits of q, and the last column of $\mathbf{D}$ is defined by the simple recurrence $d_i = \frac{d_{i-1} + q_i}{b}$ with initial condition $d_{-1} = 0$. (Note that $b^{i+1} d_i = q \mod b^{i+1} \in \{0, \cdots, b^{i+1} - 1\}$.)

Then, on input $u \in \{0, 1, \cdots, q-1\}$, we proceed as follows:

1. Compute an arbitrary element $\mathbf{u} \in \mathbb{Z}^k$ of the lattice coset $\Lambda_u^{\perp}(\mathbf{g}^t)$, for example $\mathbf{u} = (u, 0, \ldots, 0)$.

2. Map $\mathbf{u}$ to $\mathbf{t} = \mathbf{S}^{-1}\mathbf{u}$ by solving a sparse system of linear equations $\mathbf{St} = \mathbf{u} \pmod{q}$.

```
Algorithm 2: $\mathbf{g}^{-1}(u)$
  Input: $u \in \{0, 1, \cdots, q-1\}$
  Output: subgaussian $\mathbf{x} \in \Lambda_u^\perp(\mathbf{g}^t)$ with parameter $(b+1)\sqrt{2\pi}$
    Let $\mathbf{u} \leftarrow [u]_b^k$, $\mathbf{x}, \mathbf{y} \leftarrow \mathbf{0}$
    $\mathbf{x} \leftarrow \mathbf{0}, \mathbf{q} = [q]_b^k$.
    set $x_{k-1} \leftarrow 0$ with probability $(q-u)/q$ and $x_{k-1} \leftarrow -1$ otherwise.
    for $i = k-2, \cdots, 0$ do
        $u \leftarrow u - u_{i+1} b^{i+1}, q \leftarrow q - q_{i+1} b^{i+1}$.
        Let $c \leftarrow -(u + x_{k-1} q)$.
        if $c < 0$ then
            $p \leftarrow (c + b^{i+1})$, $z \leftarrow -1$.
        else
            $p \leftarrow c$, $z \leftarrow 0$.
        set $x_i \leftarrow z + 1$ with probability $p/b^{i+1}$ and $x_i \leftarrow z$ otherwise.
    for $i \in \{0, \cdots, k-2\}$ do
        $y_i \leftarrow b \cdot x_i - x_{i-1} + x_{k-1} \cdot q_i + u_i$.
    $y_{k-1} \leftarrow -x_{k-2} + x_{k-1} \cdot q_{k-1} + u_{k-1}$.
    return $\mathbf{y}$.
```

3. Pick a subgaussian sample from the lattice coset $\mathcal{L}(\mathbf{D}) + \mathbf{t}$.
4. Apply the (sparse) linear transformation $\mathbf{S}$ to the sample, to obtain a subgaussian sample from $\Lambda_u^\perp(\mathbf{g}^t)$.

Here the (randomized) nearest plane algorithm admits a simple and efficient specialization because it is applied to a basis, $\mathbf{D}$, which has a diagonal GSO. The linear transformations $\mathbf{S}^{-1}$ and $\mathbf{S}$ can also be computed in linear time because $\mathbf{S}$ is sparse and triangular. As a result, the algorithm runs in linear time $O(k)$ and does not require any pre-processing. Finally, we get an output with subgaussian parameter $(b+1)\sqrt{2\pi}$ since $\mathbf{S}$ has small spectral norm.

The actual algorithm is given in Algorithm 2. The algorithm directly implements the outline given above, but it is specialized/optimized to avoid the explicit computation of the sparse matrices $\mathbf{S}, \mathbf{D}$, and to use only integer numbers (avoids floating point numbers). Details about the correctness and analysis of the algorithm are provided in the rest of this section.

Lemma 4.1 *The first loop of Algorithm 2 performs the subgaussian nearest plane algorithm (described generically in the Appendix) on the lattice generated by* $\mathbf{D}$ *around target* $\mathbf{t} := -\mathbf{S}^{-1}[u]_b^k$.

Proof: Let $\mathbf{d}$ be the last column of $\mathbf{D}$. The last entry of $\mathbf{t}$ is $t_{k-1} = -u/b^k$ and the last entry of $\mathbf{d}$ is $d_{k-1} = q/b^k$. Therefore, we are randomly rounding x_{k-1} around the center $\left\langle \mathbf{t}, \tilde{\mathbf{d}} \right\rangle / \|\tilde{\mathbf{d}}\|^2 = -u/q \in (-1, 0]$.

For the remainder of the loop, we note that $\mathbf{t} = -\mathbf{S}^{-1} \cdot \mathbf{u}$ has entries $t_i = -(\sum_{j=0}^{i} u_j \cdot b^j)/b^{i+1}$, represented by the recurrence relation $t_i = t_{i-1}/b + u_i/b$, $t_0 = -u_0/b$. This matches the recurrence relation for $\mathbf{d}$, $d_i = (\sum_{j=0}^{i} q_j \cdot b^j)/b^{i+1}$ since $\mathbf{d} = \mathbf{S}^{-1}[q]_b^k$, so we can compute the remaining centers for the nearest plane algorithm by these recurrences. Specifically, we are performing a randomized rounding around the centers $c_i = t_i - x_{k-1} d_i = -(\sum_{l=0}^{i} u_l \cdot b^l + x_{k-1} \cdot \sum_{j=0}^{i} q_j \cdot b^j)/b^{i+1} \in (-1, 1)$. These centers are stored as c in the pseudocode. The variable z represents the two parallel planes (copies of $\mathcal{L}([\mathbf{d}_1, \cdots, \mathbf{d}_{i-1}])$ shifted by integer multiples of $\mathbf{d}_i$) separated by $\tilde{\mathbf{d}}_i$. The lemma follows. □

By storing $\mathbf{d} = \mathbf{S}^{-1}[q]_b^k$ in-advance, one can change the code to sample the first $k-1$ coordinates of $\mathbf{x}$ in-parallel since $\mathcal{L}(\mathbf{d}_0, \cdots, \mathbf{d}_{k-2}) = \mathbb{Z}^{k-1} \oplus \{0\}$. The proof of Theorem 4.1 follows below.
Proof: For the case $q = b^k$, Algorithm 1 returns a subgaussian sample $\mathbf{x} \in \Lambda_u^\perp(\mathbf{g}^t)$ with parameter $(b-1)\sqrt{2\pi}$ in time and space $O(\log_b q)$ while consuming $\log_2(q)$ of random bits by inspection, and Lemma 2.2.

Alternatively, let $q \neq b^k$. Now by Lemma 4.1, $\mathbf{x}$ after the first loop is so that $\mathbf{Dx}$ is the output of subgaussian nearest plane algorithm on $\mathbf{D}$ centered around $-\mathbf{S}^{-1}\mathbf{u}$. By Lemma 2.3, $\mathbf{S}_q\mathbf{x}+\mathbf{u}$ is a subgaussian vector with parameter $\sqrt{\lambda_{max}(\mathbf{S}\cdot\mathbf{S}^t)}\sqrt{2\pi}$, where $\lambda_{max}(\mathbf{S}\cdot\mathbf{S}^t)$ is the maximum eigenvalue of $\mathbf{S}\cdot\mathbf{S}^t$. A routine calculation for $\mathbf{S}\cdot\mathbf{S}^t$'s entries and the Geršgorin Circle Theorem (Theorem 2.1) imply $\lambda_{max}(\mathbf{S}\cdot\mathbf{S}^t) \le (b+1)^2$. Since during each iteration in the first loop we draw a random number in $\mathbb{Z}_{b^i}$ to represent p, the algorithm consumes exactly $\log_2 b(1+2+\cdots+k) = \log_2 b\cdot(k^2+k)/2$ random bits. □

5 Gadget Decoding

Here we discuss our main algorithm for the problem of *LWE gadget decoding*, defined in Section 2.2, on the gadget matrix $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g}^t$ with entries in $\mathbb{Z}_q$, for an arbitrary modulus q. Given a $\mathbf{v}^t = \mathbf{s}^t\mathbf{G} + \mathbf{e}^t \in \mathbb{Z}_q^{nk}$ as input, we can break the vector into n components of length k, then decode (in-parallel) each component with respect to $\mathbf{g}^t$. Therefore, we focus on decoding $\mathbf{g}^t$ as a gadget for $\mathbb{Z}_q$.

Our algorithm and its respective gadgets are parameterized by an integer "base" b. We consider b as fixed in this section, though varying b for a fixed modulus q yields efficiency/quality trade-offs for these gadgets. Later, in Section 6 we present a CRT gadget that can be used to efficiently decode an input given in CRT form.

Let $k = \lceil \log_b q \rceil$ and the gadget be $\mathbf{g}^t = (1, b, \cdots, b^{k-1})$. The vector $\mathbf{g}^t$ is a size k gadget of quality $(b/2)\sqrt{k}$ for $\mathbb{Z}_q$. The results in this section are summarized in the following theorem.

Theorem 5.1 *For every modulus q, and gadget $\mathbf{g}^t = (1, b, \cdots, b^{k-1})$, there is a time and space $O(k)$ algorithm decoding $\mathbf{g}^t$ with tolerance $q/2(b+1)$.*

A proof of Theorem 5.1 is given by the algorithm presented in this section. Note, Theorem 3.1 implies a polynomial time decoding algorithm for $\mathbf{g}^t$ with error tolerance $\|\mathbf{e}\|_\infty \le q/2\sqrt{k}(b+1)$. Our decoding algorithm is more efficient and has a higher error tolerance by a factor $\sqrt{k}$ than the general gadgets decoding guarantee given by Theorem 3.1.

An optimized, linear time and space $O(k)$, decoding algorithm is given in [37] for the case $q = b^k$. The reason for this algorithm's efficiency is that the commonly used basis for $\Lambda_{b^k}(\mathbf{g}^t)$ results in a linear time nearest plane algorithm. In more detail, a basis for $\Lambda_{b^k}(\mathbf{g}^t)$ in this case is the triangular matrix $\mathbf{B}_{b^k} = b^k\cdot\mathbf{S}^{-t}$, where $\mathbf{S}$ is the commonly used basis for $\Lambda_{b^k}^{\perp}(\mathbf{g}^t)$ presented in Section 2.2, and this basis has a GSO of $(q/b)\cdot\mathbf{I}$.

However, the simple decoding idea presented in [37] fails when $q \neq b^k$. Because $\Lambda_q(\mathbf{g}^t)$'s commonly used basis has a dense GSO, Babai's nearest plane algorithm takes time $O(k^2)$ and space $O(k^3)$ when naively applied on $\Lambda_q^{\perp}(\mathbf{g}^t)$.

Efficient Decoding Algorithm The intuition for our algorithm is best initially viewed through the case when $q = b^k$. Given an input $\mathbf{v}$, another way to decode the lattice $\Lambda_{b^k}(\mathbf{g}^t)$ is to use $\mathbf{S}^t$ as a linear transformation, decode $\mathbf{S}^t\mathbf{v}$ to the lattice $b^k\cdot\mathbb{Z}^k$ with the nearest plane algorithm, then map the nearest point in $b^k\cdot\mathbb{Z}^k$ back to $\Lambda_{b^k}(\mathbf{g}^t)$. This leads to a slightly stronger condition on the noise vector $\mathbf{e}$ since we now need $\mathbf{S}^t\mathbf{e} \in \mathcal{P}_{1/2}(q\cdot\mathbf{I})$, which is satisfied if $\|\mathbf{e}\|_\infty < q/2(b+1)$. Though there is no need to do this given the algorithm in [37], this is essentially what we will do in the case when $q \neq b^k$.

Overview The overview of our efficient decoding algorithm for an arbitrary modulus is as follows. First recall the sparse, triangular factorization of $\Lambda_q^{\perp}(\mathbf{g}^t)$'s commonly used basis given in section 2.2, $\mathbf{S}_q = \mathbf{SD}$. The duality relation for q-ary lattices, $\Lambda_q(\mathbf{g}^t) = q\cdot\Lambda_q^{\perp}(\mathbf{g}^t)^*$, dictates that a basis for $\Lambda_q(\mathbf{g}^t)$ is $q\cdot\mathbf{S}_q^{-t} = \mathbf{S}^{-t}(q\cdot\mathbf{D}^{-t})$. Luckily, the matrix $\mathbf{D}^{-t}$ is sparse with a diagonal GSO, and $\mathcal{P}_{1/2}(q\cdot\tilde{\mathbf{D}}^{-t}) \supseteq \mathcal{P}_{1/2}(q\cdot\mathbf{I})$ (meaning we can decode as long as $\|\mathbf{e}\|_\infty < q/2(b+1)$). Therefore, we can decode $\mathbf{g}^t$ by the following.

1. Given $\mathbf{v}$, first apply $\mathbf{S}^t$ as a linear transformation.
2. Then, decode the vector $\mathbf{S}^t\mathbf{v}$ to the lattice generated by $q\mathbf{D}^{-t}$ using the nearest plane algorithm.

Algorithm 3: DecodeG($\mathbf{v}, b, \mathbf{r}[q]_b^k$)

```
Input: $\mathbf{v} \in \mathbb{Z}^k$, $b$, and $\mathbf{q} = [q]_b^k$.
Output: $s \in \mathbb{Z}_q$ where $\mathbf{v} = s\mathbf{g}^t + \mathbf{e}^t$ as long as $\|\mathbf{e}\|_\infty < q/2(b+1)$.
    for $i \leftarrow 0, \cdots, k-2$ do
        $v_i \leftarrow bv_i - v_{i+1}$.
    $v_{k-1} \leftarrow b \cdot v_{k-1}$.
    Let $\mathbf{x} \leftarrow \mathbf{0}$ and reg $\leftarrow 0$.
    for $i \leftarrow 0, \cdots, k-2$ do
        $x_i \leftarrow \lceil v_i/q \rfloor$ and reg $\leftarrow$ reg$/b + b^{k-1} \cdot q_i$.
        $v_{k-1} \leftarrow v_{k-1} + x_i \cdot$ reg.
    $x_{k-1} \leftarrow \lceil v_{k-1}/b^k \rfloor$.
    Let $s \leftarrow x_{k-1}$ and reg $\leftarrow 0$.
    for $i \leftarrow k-2, \cdots, 0$ do
        reg $\leftarrow b \cdot$ reg $+ q_{i+1}$.
        $s \leftarrow s + x_i \cdot$ reg
    return $s \mod q$.
```

Both steps can be computed in linear time and space, $O(k)$, given the sparsity of $\mathbf{S}$ and $q\mathbf{D}^{-t}$, and $q\mathbf{D}^{-t}$'s diagonal GSO.

The pseudocode for our algorithm is shown in DecodeG. In short, the algorithm has three components, where each is represented by a loop in the pseudocode. These components are to first compute the linear transformation on the input $\mathbf{v} \leftarrow \mathbf{S}^t\mathbf{v}$, then to run the nearest plane algorithm on the lattice generated by $q \cdot \mathbf{D}^{-t}$, and finally to return s represented as the first entry of the nearest lattice point in $\Lambda_q(\mathbf{g}^t)$ modulo q. The proof of Theorem 5.1 follows from Lemmas 5.1 and 5.2 below.

Lemma 5.1 *The second loop in* DecodeG *is an instantiation of Babai's nearest plane algorithm on the lattice $q \cdot \mathbf{D}^{-t}$ given target $\mathbf{S}^t\mathbf{v}$, running in time and space $O(k)$.*

Proof: Recall the structure of $\mathbf{D}$ from section 2.2, $\mathbf{D} = [\mathbf{M}|\mathbf{d}]$ where $\mathbf{M}^t = [\mathbf{I}_{k-1}|\mathbf{0}]$ and $\mathbf{d}$ has entries $d_i = (q \mod b^{i+1})/b^{i+1}$, with $q \mod b^{i+1} \in \{0, 1, \cdots, b^{i+1}-1\}$. Then, it follows that $q\mathbf{D}^{-t}$ has a similar triangular, sparse structure. This is given by $q \cdot \mathbf{D}^{-t} = \begin{pmatrix} q\mathbf{I}_{k-1} & \mathbf{0} \\ \mathbf{c}^t & b^k \end{pmatrix}$ and the vector $\mathbf{c} \in \mathbb{Z}^{k-1}$ has entries $c_i = -b^{k-1-i} \cdot (q_0 + bq_1 + \cdots + b^i q_i) = -b^{k-1-i} \cdot (q \mod b^{i+1}) \in [-q, 0]$. Further, the entries of $\mathbf{c}$ satisfy the recurrence relation $-c_i = \frac{-(c_{i-1})}{b} + b^{k-1}q_i$ with the initial condition $-c_0 = b^{k-1}q_0$. The variable reg in DecodeG stores c_i, and it is updated using the recurrence relation for $\mathbf{c}$. The vector $\mathbf{x}$ in the pseudocode stores the coefficients of the nearest lattice point expressed in the basis $q\mathbf{D}^{-t}$. The Lemma follows by inspection. □

Lemma 5.2 *The last loop in* DecodeG *computes $s \mod q$ in time and space $O(k)$.*

Proof: Represent the first row of $\mathbf{B} = \mathbf{S}^{-t}q\mathbf{D}^{-t}$ as $\mathbf{h}$, and note $\langle \mathbf{h}, \mathbf{x} \rangle = s \mod q$. A careful analysis of $q\mathbf{D}^{-t}$ and $\mathbf{S}^{-t}$ gives us an expression for $\mathbf{h}$'s entries: $h_i = q_{i+1} + bq_{i+2} + \cdots + b^{k-i-2}q_{k-1} = \frac{q - (q \mod b^{i+1})}{b^{i+1}}$ for $i \in \{0, 1, \cdots, k-2\}$ and $h_{k-1} = 1$. All but the last entry of $\mathbf{h}$ satisfy the recurrence relation $h_i = q_{i+1} + b \cdot h_{i+1}$ for $i \in \{0, \cdots, k-2\}$, with an initial value of q_{k-1} (which is *not* the actual value of $\mathbf{h}$'s last entry). We use this recurrence relation to compute $\mathbf{h}$'s entries one at a time in the last loop, stored in the variable reg. The Lemma follows by inspection. □

6 Gadgets for the CRT Representation

Many applications of lattice gadgets require a large modulus that, for secure and functional sets of parameters, surpasses the native 64-bit integer arithmetic in a modern machine's hardware. One common method to circumvent the use of multi-precision numbers is to pick a modulus of the form $q = \prod q_i$ with each q_i less than 64 bits. Then, one can store an element $u \in \mathbb{Z}_q$ as its *Chinese Remainder* representation (CRT form[¶]) as $(u \mod q_1, \cdots, u \mod q_l)$ and perform computations via the Chinese Remainder Theorem, utilizing the ring isomorphism $\mathbb{Z}_q \cong \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_l}$. Simple forms of the gadget matrix (e.g. power of two matrix) are not compatible with this representation because the binary digits of a number cannot be easily recovered from the CRT components without a costly reconstruction phase involving large numbers modulo q.

In this section, we discuss a gadget for the CRT form. As usual, the gadget admits a compact (implicit) representation, and does not need to be computed and stored explicitly. Most importantly, the gadget allows us to use the algorithms in Sections 4 and 5 in order to perform subgaussian decomposition, discrete Gaussian sampling, and LWE gadget decoding all given input represented in CRT form. This has several theoretical and practical advantages: (1) the algorithms can be directly used by efficient applications that already store their numbers in CRT form, (2) our algorithms can be easily parallelized as they operate on each CRT component independently, (3) all algorithms only require arithmetic on small numbers (at most $\max_i q_i$) even if the modulus $q = \prod_i q_i$ may be very big. (Efficient solutions to Discrete Gaussian Sampling for the individual moduli q_i, as needed by our CRT DGS algorithm, are given in [37, 23].) We remark that a balanced, deterministic digit decomposition is provided in [18, 40], and an LWE decoding algorithm for a CRT/RNS hybrid gadget for general rings is given in the library's code[‖] (without an analysis). Our results are summarized in the following theorem. We emphasize the analysis below assumes integer operations, including reductions modulo q_i, are done in constant time. This is because our algorithms are best implemented when each q_i is less than 64 bits, avoiding the use of multi-precision numbers.

Theorem 6.1 *Let q have factorization $q = \prod_{i=1}^{l} q_i$ into coprime factors $\{q_i\}$, $(b_i)_{i=1}^{l}$ be an l-tuple of bases with $b_i < q_i$ for all i, and let $k = \sum k_i$ where $k_i = \lceil \log_{b_i} q_i \rceil$. There exists a gadget, $\mathbf{g}_{CRT}^t$, for $\mathbb{Z}_q$ of size k and quality $\max_i b_i/2$. Further, the gadget satisfies the following properties:*

- *Subgaussian decomposition can be performed in-parallel with l processors, each using time and space $O(k_i)$, consuming less than $k_i \log_2 q_i$ random bits (($\lceil\log_2(q_i)$ random bits if $q_i = b_i^{k_i}$)) and with parameter at most $(\max_i(b_i) + 1)\sqrt{2\pi}$.*

- *For any $\epsilon > 0$, discrete Gaussian sampling can be performed in-parallel with l processors, each in time and space $O(k_i)$ with width $s \geq O(b_j^{1.5})\eta_\varepsilon(\mathbb{Z}^{k_j})$ for index j maximizing $\sqrt{2b_j}(b_j + 1) \cdot \eta_\varepsilon(\mathbb{Z}^{k_j})$.*

- *$\mathbf{g}_{CRT}^t$ is decodable in-parallel with l processors in time and space $O(k_i)$ with tolerance $q/(2\max_i(b_i)+1)$.*

As expected, each processor gets slightly more efficient whenever $q_i = b_i^{k_i}$. The algorithms are represented in Figure 1.

The CRT Gadget For each coprime factor q_i, fix the *base-b_i* gadget vector as $\mathbf{g}_i^t := (1, b_i, \cdots, b_i^{k_i - 1})$ where $k_i = \lceil \log_{b_i}(q_i) \rceil$. Let $k = \sum_i k_i$, $q_i^* = q/q_i$, and $\hat{q}_i = (q_i^*)^{-1} \mod q_i$. Consider the gadget vector, which we call the *general CRT gadget*, $\mathbf{g}_{CRT}^t = (q_1^* \hat{q}_1 \cdot \mathbf{g}_1^t, \cdots, q_l^* \hat{q}_l \cdot \mathbf{g}_l^t) \mod q \in \mathbb{Z}_q^{1 \times k}$. This is a generalization of the gadgets (or implicit in algorithms) used in [11, 31, 32, 8]. As before, the gadget matrix is the block-diagonal matrix $\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}_{CRT}^t$. Theorem 6.1 follows from the fact $\Lambda_q^\perp(\mathbf{g}_{CRT}^t) = \Lambda_{q_1}^\perp(\mathbf{g}_1^t) \oplus \cdots \oplus \Lambda_{q_l}^\perp(\mathbf{g}_l^t)$, Theorem 4.1, and Proposition 3.1 in [23]. The parallel decoding algorithm is obtained by a slight adaptation to DecodeG presented in Section 5, and is analyzed in the Section 6.1. We prove the direct sum decomposition of $\Lambda_q^\perp(\mathbf{g}_{CRT}^t)$ below.

Proof: For the inclusion $\supseteq$, let $\mathbf{x}_i \in \Lambda_{q_i}^\perp(\mathbf{g}_i^t)$ be arbitrary with $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_l)$ as their concatenation. Then, $\langle \mathbf{x}_i, \mathbf{g}_i^t \rangle = aq_i \in q_i \cdot \mathbb{Z}$ and $\langle \mathbf{x}, \mathbf{g}_{CRT} \rangle \mod q = \sum_{i=1}^{l} q_i^* \hat{q}_i \langle \mathbf{x}_i, \mathbf{g}_i \rangle \mod q = 0 + \cdots + 0 \mod q$. We

[¶] This is also known as the residue number system (RNS) in previous works.

[‖] https://github.com/cpeikert/Lol/blob/master/lol/Crypto/Lol/Gadget.hs

```
Algorithm 4: Sampling in CRT form.
  Input: $(u_1, \cdots, u_l)$
  Output: $\mathbf{g}_{CRT}^{-1}(u_1, \cdots, u_l)$.
    for $i \in \{1, \cdots, l\}$ do
        $\mathbf{x}_i \leftarrow \mathbf{g}_i^{-1}(u_i)$.
    return $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_l)$.
```

```
Algorithm 5: Decoding in CRT form.
  Input: $\mathbf{v}^t = s \cdot \mathbf{g}_{CRT} + \mathbf{e}^t \mod q$
  Output: $(s_1, \cdots, s_l)$.
    Let $\mathbf{v} = (\mathbf{v}_1, \cdots, \mathbf{v}_l)$ for each $\mathbf{v}_i \in \mathbb{Z}_q^{k_i}$.
    for $i \in \{1, \cdots, l\}$ do
        $s_i \leftarrow \textsc{DecodeCRT}(\mathbf{v}_i)$
    return $(s_1, \cdots, s_l)$.
```

Figure 1: Pseudocode for the parallel algorithms given in Theorem 6.1. We let $\mathbf{g}_i^{-1}(\cdot)$ denote either the subgaussian decomposition algorithm given in Section 4 or a discrete Gaussian sampler. The subroutine DecodeCRT is a variation of the decoding algorithm given in Section 5 and is described in Section 6.1.

```
Algorithm 6: $\textsc{DecodeCRT}(\mathbf{v}_i, b_i, \mathbf{t} = [q_i]_{b_i}^{k_i}, q, q_i^*)$
  Input: $\mathbf{v}_i \in \mathbb{Z}^{k_i}$, $b_i$, $q_i^*$, $q$, and $\mathbf{t} = [q_i]_{b_i}^{k_i}$.
  Output: $s \mod q_i$ where $\mathbf{v} = s\mathbf{g}^t + \mathbf{e}^t \mod q$ as long as $\|\mathbf{e}\|_\infty < q/2(b_i + 1)$.
    for $j \leftarrow 0, \cdots, k_i - 1$ do
        $v_j \leftarrow b_j v_j - v_{j+1}$.
    Let $\mathbf{x} \leftarrow \mathbf{0}$.
    for $j \in \{0, \cdots, k_i - 2\}$ do
        $x_j \leftarrow \lceil v_j/q \rfloor$.
    $x_{k-1} \leftarrow \lceil (v_{k-1} - \langle \mathbf{c}, \mathbf{x}_0^{k-2} \rangle)/(q_i^* {b_i}^{k_i}) \rfloor$.
    Let $s_i \leftarrow x_{k-1}$ and reg $\leftarrow 0$.
    for $j \leftarrow k_i - 2, \cdots, 0$ do
        reg $\leftarrow b \cdot$ reg $+ t_{j+1} \cdot q_i^*$.
        $s_i \leftarrow s_i + x_j \cdot$ reg
    return $s_i \mod q_i$.
```

prove the converse by inducting on l, the number of q's coprime factors. The base case is routine. Now consider $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_l) \in \Lambda_q^\perp(\mathbf{g}_{CRT}^t)$ with $\mathbf{x}_i \in \Lambda_{q_i}^\perp(\mathbf{g}_i^t)$ for $i = 0, \cdots, l-1$ and $\mathbf{x}_l \in \mathbb{Z}^{k_l}$. By the inductive hypothesis, $\langle \mathbf{x}, \mathbf{g}_{CRT} \rangle \mod q = q_l^* \hat{q}_l \cdot \langle \mathbf{x}_l, \mathbf{g}_l^t \rangle = 0 \mod q$. Viewing this equation in $\mathbb{Z}$ and dividing both sides by q_l^* implies $\hat{q}_l \cdot \langle \mathbf{x}_l, \mathbf{g}_l \rangle \mod q_l = 0$. Finally, we conclude $\langle \mathbf{x}_l, \mathbf{g}_l \rangle \mod q_l = 0$ since $\hat{q}_l$ is a multiplicative unit in $\mathbb{Z}_{q_l}$. □

6.1 Decoding the CRT Gadget

Here we show how the efficient gadget decoding algorithm from Section 5 adapts to the general CRT gadget described in Section 6. Recall the decomposition of $\mathbf{g}^t$'s lattice, $\Lambda_q^\perp(\mathbf{g}^t) = \Lambda_{q_1}^\perp(\mathbf{g}_1^t) \oplus \cdots \oplus \Lambda_{q_l}^\perp(\mathbf{g}_l^t) = \mathcal{L}(\mathbf{S}_{q_1}) \oplus \cdots \oplus \mathcal{L}(\mathbf{S}_{q_l})$. The duality relation for q-ary lattices yields $\Lambda_q(\mathbf{g}^t) = q \cdot (\Lambda_q^\perp(\mathbf{g}^t))^* = q \cdot \big(\bigoplus_i \mathcal{L}(\mathbf{S}_{q_i}^{-t} \mathbf{D}_{q_i}^{-t})\big) = \big(\bigoplus_i \mathcal{L}(\mathbf{S}_{q_i}^{-t} q_i^* \cdot (q_i \cdot \mathbf{D}_{q_i}^{-t}))\big)$.

Now we have a clear way to decode the general CRT gadget. First, break the input into l blocks, $\mathbf{v}^t = s\mathbf{g}^t + \mathbf{e}^t \mod q = (\mathbf{v}_1^t, \cdots, \mathbf{v}_l^t)$ where $\mathbf{v}_i^t = s \cdot q_i^* \hat{q}_i \mathbf{g}_i^t + \mathbf{e}_i^t \mod q$. Then, we compute the following. First, transform $\mathbf{v}_i$ to $\mathbf{S}_{q_i}^t \mathbf{v}_i$. Then, decode $\mathbf{S}_{q_i}^t \mathbf{v}_i$ to the lattice $q_i^*(q_i \mathbf{D}_{q_i}^{-t})$. Finally, return $s \mod q_i$. The pseudocode is given as the algorithm DecodeCRT. Another change is that we store the vector $\mathbf{c}$ in memory. Recall, $\mathbf{c}$ has $k-2$ entries of the form $c_j = -b_i^{k_i - 1 - j}(q_i \mod b_i^j)$. Note that the correctness condition of our algorithm is still $\|\mathbf{e}^t\|_\infty < q/2(\max_i(b_i) + 1)$.

Decoding in CRT Form Here we describe how DecodeCRT can decode $\mathbf{v} = s\mathbf{g} + \mathbf{e}$ where the input is given in its CRT representation. The ideas sketched here follow from [32]. The linear transformation $\mathbf{v} \to \mathbf{S}^t\mathbf{v}$ is easily computed given the CRT form of $\mathbf{v}$. Really, we are only concerned with divisions and integer rounding. In the second loop, note that $x_j \leftarrow \lceil v_j/q \rfloor = \lceil \sum_{o=1}^{l} [(v \mod q_o) \cdot (\hat{q_o}/q_o)] \rfloor$. Next we consider the line $x_{k-1} \leftarrow \lceil (v_{k-1} + \langle \mathbf{c}, \mathbf{x}_0^{k-2} \rangle)/(q_i^* {b_i}^{k_i}) \rfloor$. First, note that $v_{k-1}/(b_i^{k_i} q_i^*) = b_i^{-k_i} \cdot \sum_{o=1}^{l} (v_{k-1} \mod q_o) \cdot \hat{q}_o(q_i/q_o)$. This should be a small number in nearly all practical instantiations. Lastly, we note that we return s in CRT form, but we can alter the algorithm to return $s \in (-q/2, q/2]$ via a simple change. The s computed in the last loop is actually $s \cdot q_i^* \hat{q}_i$. So, we can remove the $\mod q_i$ in the return statement and sum up the output from the l parallel processors, $\sum_i (s \cdot q_i^* \hat{q}_i) = s \cdot \sum_i (q_i^* \hat{q}_i) = s \cdot 1 \mod q$.

7 Toolkit Implementation and Its Application

7.1 Software Implementation

We implemented most of the algorithms presented in this work in PALISADE [41], a modular open-source lattice cryptography library that includes ring-based implementations of homomorphic encryption, proxy re-encryption, identity-based encryption, attribute-based encryption, and other lattice schemes. More concretely, we added a new lattice gadget toolkit module to PALISADE that implements the following algorithms:

- Subgaussian gadget decomposition (Algorithm 2) for arbitrary moduli and gadget bases.
- Efficient gadget in CRT representation, enabling both trapdoor sampling and subgaussian gadget decomposition in the CRT representation.
- Subgaussian gadget decomposition for cyclotomic rings both in positional and CRT number systems, which wraps around Algorithm 2.

The toolkit module complements/improves the lattice gadget algorithms previously added to PALISADE, such as trapdoor sampling for cyclotomic rings proposed in [23] and implemented in [30, 17]. The full lattice gadget capability will be included in the next major public release of PALISADE.

7.2 Optimized Variant of Key-Policy Attribute-Based Encryption

We use the lattice gadget toolkit algorithms to build and implement a full RNS/CRT variant of the short-secret Key-Policy Attribute-Based Encryption (KP-ABE) scheme originally proposed in [10] and implemented for cyclotomic rings in [19]. The KP-ABE scheme is a complex cryptographic primitive that can be used for attribute-based access control applications, as well as a building block for audit log encryption, targeted broadcast encryption, predicate encryption, functional encryption, and some forms of program obfuscation [10, 27].

7.2.1 Overview

ABE is a public key cryptography primitive that enables the decryption of a ciphertext by a user only if a specific access policy (defined over ℓ attributes) is satisfied. In the key-policy scenario, a message is encrypted using the attribute values as public keys, and a specific access policy is typically defined afterwards. When the access policy becomes known, a secret key for the policy is generated (using trapdoor sampling in our KP-ABE scheme), and the ciphertexts and public keys are homomorphically evaluated over the policy circuit (using a GSW-type homomorphic multiplication in our KP-ABE scheme).

The short-secret KP-ABE scheme is a tuple of functions, namely `Setup`, `Encrypt`, `EvalPK`, `KeyGen`, `EvalCT`, and `Decrypt`, whose definitions are:

- $\textsc{Setup}(1^\lambda, \ell) \to \{\texttt{MPK}, \texttt{MSK}\}$: Given a security parameter λ and the number of attributes ℓ, a trusted private key generator (PKG) generates a master public key MPK and a master secret key MSK. MPK contains the ABE public parameters while MSK includes the trapdoor that is used by PKG to generate secret keys for access policies.

- $\textsc{Encrypt}(\mu, \mathbf{x}, \texttt{MPK}) \to \mathbf{C}$: Using MPK and attribute values $\mathbf{x} \in \{0,1\}^\ell$, sender encrypts the message μ and outputs the ciphertext $\mathbf{C}$.

- $\textsc{EvalPK}(\texttt{MPK}, \mathbf{x}, f) \to \texttt{PK}_f$: Homomorphically evaluate MPK over a policy (Boolean circuit) $f : \{0,1\}^\ell \to \{0,1\}$ to generate a public key $\texttt{PK}_f$ for the policy f.

- $\textsc{KeyGen}(\texttt{MSK}, \texttt{MPK}, \texttt{PK}_f) \to \texttt{SK}_f$: Given MSK, MPK and policy-specific $\texttt{PK}_f$, PKG generates the secret key $\texttt{SK}_f$ corresponding to f. PKG sends $\texttt{SK}_f$ to the receiver that is authorized to decrypt ciphertexts encrypted under f.

- $\textsc{EvalCT}(\mathbf{C}, \mathbf{x}, f) \to \mathbf{C}_f$: Homomorphically evaluate $\mathbf{C}$ over the policy f to generate the ciphertext $\mathbf{C}_f$.

- $\textsc{Decrypt}(\mathbf{C}_f, \texttt{SK}_f) \to \bar{\mu}$: Given the homomorphically computed ciphertext $\mathbf{C}_f$ and corresponding secret key $\texttt{SK}_f$, find $\bar{\mu}$, which is the same as the original message μ if the receiver has the secret key matching the policy f.

The most computationally expensive operations are EvalPK and EvalCT, which homomorphically evaluate a circuit of depth $\lceil \log_2 \ell \rceil$ using the GSW homomorphic multiplication approach. At each level of a Boolean circuit composed of NAND gates (which are used for benchmark evaluation in [19]), the algorithms compute matrix products $\mathbf{B}_{2i}\mathbf{G}^{-1}(-\mathbf{B}_i)$ and $\left(\mathbf{G}^{-1}(-\mathbf{C}_i)\right)^t \mathbf{C}_{2i}$ for public keys and ciphertexts, respectively. Here, $\mathbf{B}_i \in R_q^{1\times m}$, $\mathbf{C}_i \in R_q^m$, $R_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$, and $m = \lceil \log_b q \rceil + 2$ (the latter corresponds to the Ring-LWE trapdoor construction). Note that that the gadget $\mathbf{G}$ is extended in this case to m by adding two zero entries to the decomposed digits.

The work [19] presents a CPU implementation of the ring variant of the KP-ABE scheme along with an efficient GPU implementation for policy evaluation and encryption. The CPU implementation was done for a binary gadget base and used the conversion from CRT to the positional number system for digit decomposition both in trapdoor sampling and gadget decomposition. To avoid the linear noise growth $O(nm)$ in gadget decomposition, the authors used a balanced digit decomposition, namely the binary non-adjacent form (NAF), that replaces digits in (0,1) with a zero-centered representation in (-1,0,1). Although this approach allows one to achieve a heuristic growth close to $O(\sqrt{nm})$ in the case of the KP-ABE scheme, the noise properties depend on the randomness of the input, i.e., this approach is deterministic.

The CPU runtimes for policy evaluation and encryption operations in [19] were far from practical (the CPU results only for ℓ up to 8 are presented), and hence the authors developed an efficient GPU implementation for these operations.

For detailed algorithms of the KP-ABE scheme, the reader is referred to [19].

7.2.2 Our Optimizations

We present a full CRT/RNS ring variant of the KP-ABE scheme that leverages the lattice gadget toolkit to significantly (by more than one order of magnitude) speed up the policy evaluation operations. In particular, our implementation includes the following optimizations as compared to [19]:

- The subgaussian gadget decomposition in CRT representation to minimize the noise growth instead of the NAF decomposition with the conversion from CRT representation to positional system. This provides a theoretical guarantee of the square-root noise growth. To achieve the repeatability of randomized decomposition in EvalPK and EvalCT, we use the same seed for the random operations in subgaussian gadget decomposition. The seed is treated as part of the master public key.

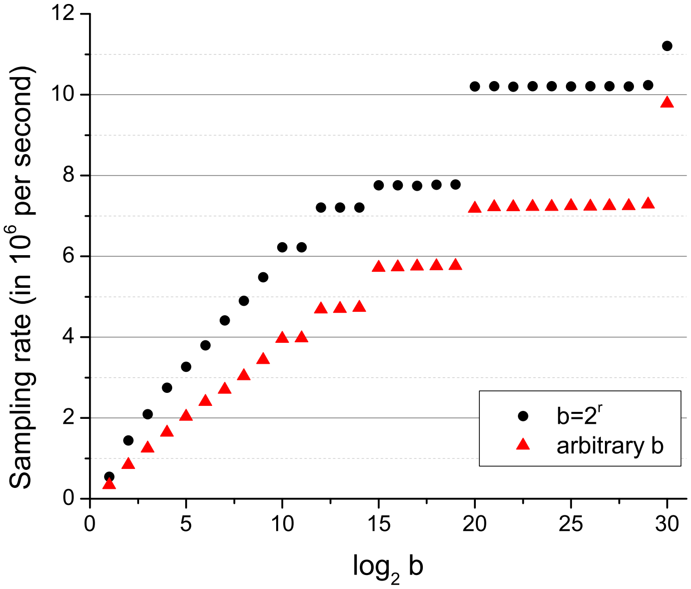

Figure 2: Runtime baseline of subgaussian sampling rate for native uniformly random integers (w.r.t a 60-bit modulus). When $b = 2^r$, the modulo reduction in digit decomposition is performed by simple bit shifting. When b is arbitrary, the slower hardware modulo operation is used. The plateaus correspond to the same number of digits, i.e., the same value of $\lceil 60/\log_2 b \rceil$.

- The CRT variant of trapdoor sampling using the gadget decomposition technique discussed in this paper in contrast to the multiprecision digit decomposition in [19].
- The RNS/CRT scaling proposed in [32] for decryption in constrast to the multiprecision scaling.
- Increased gadget base b (both in trapdoor and subgaussian gadget decomposition) instead of the binary base.

7.2.3 Parameter Selection

As the correctness constraint in [19] was derived for the classical binary-base gadget decomposition, we provide here a modified version incorporating the effect of a larger gadget base for the case of subgaussian gadget decomposition:

$$q > 4C_1 s\sigma\sqrt{mn}\left(b\sqrt{mn}\right)^d, \tag{2}$$

where $C_1 = 128$, $s = C \cdot \sigma^2(b+1) \cdot (\sqrt{n \log_b q} + \sqrt{2n} + 4.7)$, $C = 1.8$, $\sigma \approx 4.578$, and $d = \lceil \log_2 \ell \rceil$. Here, C and C_1 are empirical parameters chosen the same way as in [19].

The differences compared to [19] are the b factor in the exponentiation base (as the digits vary between $-b$ and b in subgaussian gadget decomposition) and a $(b+1)$ factor in the expression for s (contributed by Gaussian sampling; see [23, 17] for a more detailed discussion of the Gaussian distribution parameter for arbitrary gadget bases).

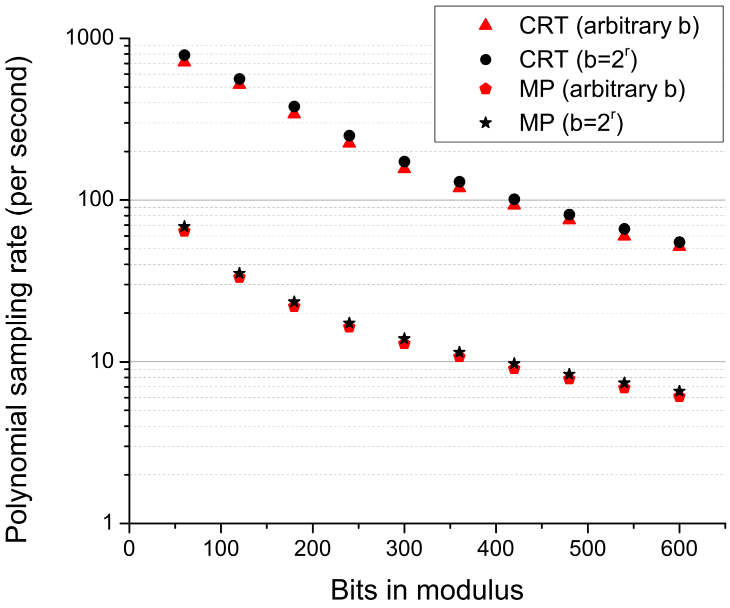

Figure 3: Comparison of sampling rates for CRT and multiprecision (MP) variants of subgaussian gadget decomposition for ring elements with 4096 coefficients and 60-bit CRT moduli at $r = \lceil \log_2 b \rceil = 20$. The MP variant requires converting from CRT representation to positional number system followed by digit decomposition w.r.t. a large integer.

8 Experimental Results

We ran the experiments in PALISADE version 1.2, which includes NTL version 10.5.0 and GMP version 6.1.2. The evaluation environment was a commodity desktop computer system with an Intel Core i7-3770 CPU with 4 cores rated at 3.40GHz and 16GB of memory, running Linux CentOS 7. The compiler was g++ (GCC) 5.3.1.

8.1 Subgaussian Gadget Decomposition

The experiments described in this section were all performed in the single-threaded mode. The goal of these results is to provide the performance baselines for subgaussian gadget decomposition, demonstrate the benefits of the efficient gadget in CRT representation, and illiustrate the effect of subgaussian sampling on the noise growth in GSW-type products.

Figure 2 shows the dependence of subgaussian gadget decomposition rate (per decomposed integer) on the gadget base for native (64-bit) integers. The results are shown both for a power-of-two base, which supports fast modulo reduction by bit shifting, and an arbitrary base, which requires a division-based modulo operation on x86 architectures. In our implementation, the native arithmetic is a building block for performing operations in CRT representation for integers that are larger than 60 bits, and, therefore, these results can be used to estimate the runtimes for larger CRT-represented integers. Figure 2 illustrates that the sampling rate increases in a discrete manner as we raise the gadget base because the number of digits is determined by $\lceil 60/\log_2 b \rceil$. The runtime is dominated by the randomized operations (as the difference between a power-of-two-base and arbitrary-base scenarios is relatively small), thus limiting the advantages of choosing the faster power-of-two bases. This suggests that a CRT representation in terms of powers of primes, where the primes are used as the residue bases, might be preferred in some instances (where an

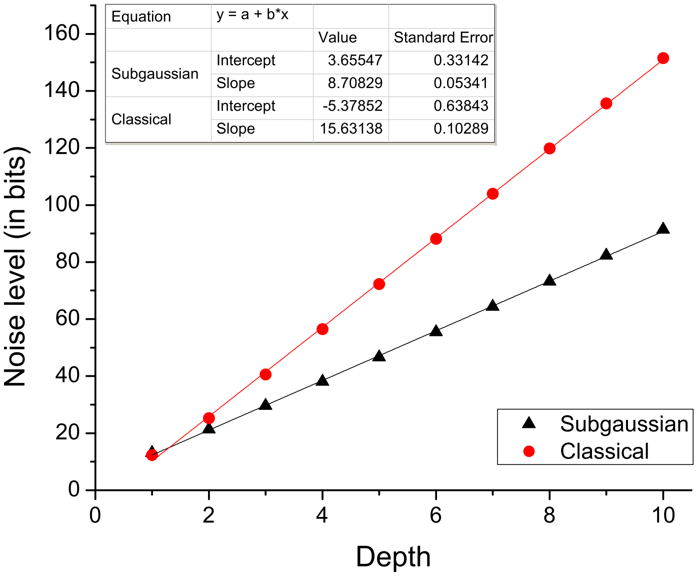

Figure 4: Noise growth for GSW-type multiplication in the ring-based KP-ABE variant ($k = 180$, $n = 1024$, $b = 2$). The base in the exponentiation is $(mn)^{\beta}$, where $m = k + 2 = 182$ and β describes the rate of noise growth. The slope of the linear interpolation is $\beta \log_2(mn)$. The values of β are 0.497 and 0.893 for subgaussian and classical gadget decomposition, respectively.

efficient implemention of arithmetic over prime powers is available) over power-of-two bases.

Figure 3 illustrates the benefits of using the efficient gadget in CRT representation when working with cyclotomic rings. The conversion from CRT representation to the positional system followed by digit decomposition w.r.t a large modulus slows down subgaussian gadget decomposition rate by almost one order of magnitude. We also observe that the difference in performance between a power-of-two base and an arbitrary base is relatively small for both cases.

Figure 4 demonstrates the differences in the noise growth of GSW-type products using the subgaussian and classical binary gadget decomposition methods. For this experiment, we generated an error vector in R^m and iteratively multiplied it by $\mathbf{G}^{-1}(\mathbf{U}_i)$, where $\mathbf{U}_i$ is a vector of unformly random ring elements in R_q^m at level i. We applied the tree multiplication approach (rather than a sequential evaluation in a right-associative manner, which reduces the noise when dealing with a chained product of fresh encryptions in GSW [14, 5]) to emulate the noise growth in evaluating a Boolean policy circuit in the KP-ABE scheme. We considered both the cases when the same $\mathbf{U}$ was used at all levels (correlated ciphertexts) and different $\mathbf{U}_i$ at each level. The results were approximately the same for both scenarios because the classical gadget decomposition matrix is centered at 0.5 (see [19] for a more detailed discussion of the classical gadget decomposition case).

Figure 4 suggests that the noise growth in the subgaussian gadget decomposition case has a square-root dependence on mn ($\beta \approx 0.5$) while the classical gadget decomposition approach results in almost linear noise growth ($\beta \approx 0.9$). Note that the intercept is lower for classical gadget decomposition because the infinity norm of digits is 1 (only 0 or 1 are possible) vs. 2 in the case of subgaussian decomposition (the allowed integer values are in the range from -2 to 2). However, this advantage does not propagate to the second level of the circuit as the square-root dependence of subgaussian gadget decomposition already plays a more dominant role here.

Table 1: Comparison of performance results for our KP-ABE variant (in bold) vs. the implementation in [19] (in parentheses). EvalCT* = EvalPK + EvalCT corresponds to the scenario when the policy evaluation of public keys and ciphertexts is done at the same time.

ℓ	k	$\log_2 n$	r	KeyGen	Encrypt	EvalCT*	EvalPK	Decrypt	RAM
				[ms]	[ms]	[s]	[s]	[ms]	[MB]
2	50 (44)	11 (11)	5 (1)	**40** (126)	**7** (33)	**0.023** (0.44)	**0.021** (0.42)	**1.8** (3.0)	**19** (58.5)
4	100 (52)	12 (12)	20 (1)	**64** (143)	**15** (57)	**0.072** (1.76)	**0.064** (1.68)	**3.9** (3.5)	**36.4** (86.3)
8	120 (60)	13 (13)	15 (1)	**151** (317)	**56** (222)	**0.59** (10.8)	**0.53** (10.4)	**8.9** (7.7)	**94.1** (255)
16	180 (70)	13 (13)	20 (1)	**177** (419)	**157** (1,483)	**1.68** (429)	**1.48** (427)	**11.5** (18.1)	**230** (2,867)
32	180	13	15	**206**	**414**	**5.67**	**5.0**	**13.46**	**508**
64	204	13	17	**226**	**1,052**	**13.1**	**11.2**	**16.39**	**1,229**
128	300	14	25	**568**	**6,454**	**98.3**	**85.5**	**45.43**	**7,024**

8.2 Key-Policy Attribute-Based Encryption

Table 1 shows the performance results for our implementation along with the corresponding results for the implementation in [19]. The first three rows for the results in [19] were obtained using native (64-bit) integer arithmetic and the last row used a multiprecision backend in PALISADE based on NTL/GMP. The experiments were run for 4 threads on a commodity desktop system, i.e., Intel Core i7-3770 CPU with 4 cores at 3.40GHz and 16GB of memory running CentOS 7. Both variants were implemented in PALISADE v1.2.

To choose the ring dimension n for both implementations, we ran the LWE security estimator** (commit 560525) [3] to find the lowest security levels for the uSVP, decoding, and dual attacks following the standard homomorphic encryption security recommendations [2]. We selected the least value of the number of security bits λ for all 3 attacks on classical computers based on the estimates for the BKZ sieve reduction cost model. All results are presented for at least 128 bits of security.

Table 1 suggests there is a speed-up of 2.1x to 3.2x for key generation, where the lattice trapdoor sampling subroutine is called. The speed-up for encryption is 3.8x to 9.5x, which is mostly attributed to the use of a larger gadget base. The speed-ups for the main botteneck operations of homomorphic public key and ciphertext evaluation are in the range from 18x to 289x, which is a combined effect of subgaussian gadget decomposition in CRT and a larger gadget base. The decryption runtimes are comparable, and already fast for both implementations. The memory requirements for our optimized variant are 2.4x to 12.5x smaller.

Note that the peformance of the KP-ABE variant in [19] dramatically degrades after switching from the native arithmetic (when $k \leq 60$ bits) to the multiprecision backend (for gadget decomposition), which is observed for $\ell = 16$ in Table 1. This implies the efficient gadgets in CRT representation are critical for supporting deeper Boolean circuits with CPU systems.

We also profiled the contributions of subgaussian gadget decomposition and the number theoretic transforms (NTT) of the digit-decomposed matrix (needed for matrix multiplication) to the runtimes for homomorphic policy evaluation of ciphertexts (EvalCT*). The contribution of subgaussian gadget decomposition was in the range from 15% to 22% w.r.t. the total homomorphic policy evaluation runtime. The contribution of the related NTTs was between 47% and 63%, suggesting that the latter is the main bottleneck of homomorphic circuit evaluation in our KP-ABE variant.

References

[1] M. Ajtai. Generating hard instances of the short basis problem. In *Automata, Languages and Programming - ICALP'99*, volume 1644 of *LNCS*, pages 1–9, 1999.

** https://bitbucket.org/malb/lwe-estimator

[2] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Cambridge MA, March 2018.

[3] M. Albrecht, S. Scott, and R. Player. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169203, 10 2015.

[4] J. Alperin-Sheriff and D. Apon. Weak is better: Tightly secure short signatures from weak prfs. *IACR Cryptology ePrint Archive*, 2017:563, 2017.

[5] J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In *CRYPTO'14*, volume 8616 of *LNCS*, pages 297–314, 2014.

[6] J. Alwen and C. Peikert. Generating shorter bases for hard random lattices. *Theory Comput. Syst.*, 48(3):535–553, 2011.

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

[8] J. Bajard, J. Eynard, M. A. Hasan, and V. Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *Selected Areas in Cryptography - SAC'16*, volume 10532 of *LNCS*, pages 423–442, 2016.

[9] P. Bert, P. Fouque, A. Roux-Langlois, and M. Sabt. Practical implementation of Ring-SIS/LWE based signature and IBE. In *Post-Quantum Cryptography - PQCrypto'18*, volume 10786 of *LNCS*, pages 271–291, 2018.

[10] D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT'14*, volume 8441 of *LNCS*, pages 533–556. Springer, 2014.

[11] G. Bonnoron, L. Ducas, and M. Fillinger. Large FHE gates from tensored homomorphic accumulator. In *AFRICACRYPT'18*, volume 10831 of *LNCS*, pages 217–251, 2018.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science - ITCS'12*, pages 309–325. ACM, 2012.

[13] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In *Symposium on Theory of Computing - STOC'13*, pages 575–584, 2013.

[14] Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In *Innovations in Theoretical Computer Science - ITCS'14*, pages 1–12, 2014.

[15] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT'16*, volume 10031 of *LNCS*, pages 3–33, 2016.

[16] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *ASIACRYPT'17*, volume 10624 of *LNCS*, pages 377–408, 2017.

[17] D. B. Cousins, G. D. Crescenzo, K. D. Gür, K. King, Y. Polyakov, K. Rohloff, G. W. Ryan, and E. Savas. Implementing conjunction obfuscation under entropic ring LWE. In *Symposium on Security and Privacy - SSP'18*, pages 354–371, 2018.

[18] E. Crockett and C. Peikert. $\Lambda o \lambda$: Functional lattice cryptography. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 993–1005. ACM, 2016.

[19] W. Dai, Y. Doröz, Y. Polyakov, K. Rohloff, H. Sajjadpour, E. Savas, and B. Sunar. Implementation and evaluation of a lattice-based key-policy ABE scheme. *IEEE Trans. Information Forensics and Security*, 13(5):1169–1184, 2018.

[20] R. del Pino, V. Lyubashevsky, and G. Seiler. Lattice-based group signatures and zero-knowledge proofs of automorphism stability. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, pages 574–591. ACM, 2018.

[21] L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT'15*, volume 9056 of *LNCS*, pages 617–640, 2015.

[22] L. Ducas and T. Prest. Fast fourier orthogonalization. In S. A. Abramov, E. V. Zima, and X. Gao, editors, *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016*, pages 191–198. ACM, 2016.

[23] N. Genise and D. Micciancio. Faster gaussian sampling for trapdoor lattices with arbitrary modulus. In *EUROCRYPT'18*, volume 10820 of *LNCS*, pages 174–203, 2018.

[24] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO'12*, pages 850–867, 2012.

[25] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Symposium on Theory of Computing - STOC'08*, pages 197–206, 2008.

[26] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO'13*, volume 8042 of *LNCS*, pages 75–92, 2013.

[27] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Predicate encryption for circuits from LWE. In *CRYPTO'15*, volume 9216 of *LNCS*, pages 503–523, 2015.

[28] S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *Symposium on Theory of Computing - STOC'15*, pages 469–477, 2015.

[29] K. D. Gür, Y. Polyakov, K. Rohloff, G. W. Ryan, H. Sajjadpour, and E. Savas. Practical applications of improved gaussian sampling for trapdoor lattices. *IACR Cryptology ePrint Archive*, 2017:1254, 2017.

[30] K. D. Gür, Y. Polyakov, K. Rohloff, G. W. Ryan, and E. Savas. Implementation and evaluation of improved gaussian sampling for lattice trapdoors. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '18, pages 61–71, 2018.

[31] S. Halevi, T. Halevi, V. Shoup, and N. Stephens-Davidowitz. Implementing bp-obfuscation using graph-induced encoding. In *Computer and Communications Security - CCS'17*, pages 783–798, 2017.

[32] S. Halevi, Y. Polyakov, and V. Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. *IACR Cryptology ePrint Archive*, 2018:117, 2018.

[33] P. N. Klein. Finding the closest lattice vector when it's unusually close. In *Symposium on Discrete Algorithms - SODA'00*, pages 937–941, 2000.

[34] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: A modest proposal for FFT hashing. In *Fast Software Encryption - FSE'08*, volume 5086 of *LNCS*, pages 54–72, 2008.

[35] V. Lyubashevsky, C. Peikert, and O. Regev. A toolkit for ring-lwe cryptography. In *EUROCRYPT'13*, volume 7881 of *LNCS*, pages 35–54, 2013.

[36] C. A. Melchor, J. Barrier, S. Guelton, A. Guinet, M. Killijian, and T. Lepoint. Nfllib: Ntt-based fast lattice library. In *CT-RSA'16*, volume 9610 of *LNCS*, pages 341–356, 2016.

[37] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT'12*, volume 7237 of *LNCS*, pages 700–718, 2012.

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

[39] D. Micciancio and J. Sorrell. Ring packing and amortized FHEW bootstrapping. In *Automata, Languages, and Programming - ICALP'18*, volume 107 of *LIPIcs*, pages 100:1–100:14, 2018.

[40] C. Peikert. personal communication.

[41] Y. Polyakov, K. Rohloff, and G. W. Ryan. PALISADE lattice cryptography library. `https://git.njit.edu/palisade/PALISADE`, Accessed October 2018.

[42] R. Vershynin. Introduction to the non-asymptotic analysis of random matrices. *CoRR*, abs/1011.3027, 2010.

A Subgaussian Nearest Plane

Now we describe the subgaussian nearest plane algorithm, $SG_{NP}(\mathbf{B}, \mathbf{t})$, used in throughout the paper. Clearly, it is a randomized version of Babai's algorithm (Theorem 2.2), and it was first used in a theoretical sense in [5]. The algorithm takes as input a target $\mathbf{t} \in \mathbb{R}^n$, a lattice basis $\mathbf{B}$, and its GSO. It returns a lattice point $\mathbf{x}$ that is randomly chosen so that $\mathbf{x} - \mathbf{t}$ is a subgaussian vector. The subgaussian parameter, as we will see, is $\max_i \|\tilde{\mathbf{b}}_i\|$.

Let $\mathbf{B}_k^*$ be the GSO of the basis $\mathbf{B}_k$. As in the nearest plane algorithm, we partition the lattice $\mathcal{L}(\mathbf{B}_k)$ into parallel planes $\mathcal{L}(\mathbf{B}_k) = \bigcup_{i \in \mathbb{Z}} (\mathcal{L}(\mathbf{B}_{k-1}) + i \cdot \mathbf{b}_k)$, but here we randomly round to the plane j or $j-1$, the planes between which the target lies, with a probability which depends on the target's distance from the nearest plane instead of directly rounding. In more detail, given an input vector $\mathbf{t}$, we perform the following: First project the target orthogonally onto the span of $\mathbf{b}_k^*$ and store the coefficient $t \leftarrow \langle \mathbf{t}, \mathbf{b}_k^* \rangle / \|\mathbf{b}_k^*\|^2$. If $t \in \mathbb{Z}$, set $c \leftarrow t$. Otherwise, pick $c \leftarrow \lfloor t \rfloor + 1$ with probability $t \mod 1$ and $c \leftarrow \lfloor t \rfloor$ otherwise. Finally, return $c \cdot \mathbf{b}_k + SG_{NP}(\mathbf{B}_{k-1}, \mathbf{t} - c \cdot \mathbf{b}_k)$. The following lemma is easily proved by induction through expanding the expectation and the definition of the GSO.

Lemma A.1 *The expected value of $SG_{NP}(\mathbf{B}_k, \mathbf{t})$ is $\mathbf{t}$ projected to* $\mathrm{span}(\mathbf{B}_k)$.

Proof: We prove this by induction on the dimension. For the base case, let the basis and target be $b, t \in \mathbb{R}$, respectively. Let $p = t/b \mod 1$ (or equivalently $t/b - \lfloor t/b \rfloor$). Then the expectation of $SG_{NP}(b, t)$ is

$$\mathbb{E}[SG_{NP}(b, t)] = b[\lfloor t/b \rfloor (1-p) + (\lfloor t/b \rfloor + 1)p] = b(t/b) = t.$$

For the inductive step, we assume $\mathbb{E}[SG_{NP}(\mathbf{B}_{k-1}, \mathbf{t})] = \mathrm{span}_{\mathbf{B}_{k-1}}(\mathbf{t})$. The coefficient c's expectation is $c' := \langle \mathbf{t}, \mathbf{b}_k^* \rangle / \|\mathbf{b}_k^*\|^2$ via the same computation as the base case. The law of iterated expectation gives us

$$\mathbb{E}[SG_{NP}(\mathbf{B}_k, \mathbf{t})] = c'\mathbf{b}_k + \sum_{i=1}^{k-1} \frac{\langle \mathbf{t} - c'\mathbf{b}_k, \mathbf{b}_i^* \rangle}{\|\mathbf{b}_i^*\|^2} \mathbf{b}_i^*.$$

The result follows from the definition of the GSO. □

To see $SG_{NP}(\mathbf{B}_k, \mathbf{t}) - \mathbf{t}$ is subgaussian, we view the space in the basis generated by $\mathbf{B}_k^*$. Since the coefficients of the output vector in this basis are chosen independently and by the definition of the GSO, $SG_{NP}(\mathbf{B}_k, \mathbf{t}) - \mathbf{t}$ is a subgaussian vector with parameter $\max_i \|\mathbf{b}_i^*\|$.

Lemma A.2 *$SG_{NP}(\mathbf{B}_k, \mathbf{t}) - \mathbf{t}$ is a subgaussian vector with parameter* $\max_i \|\mathbf{b}_i^*\|$.

Proof:(Sketch.) The result is seen through expressing $SG_{NP}(\mathbf{B}_k, \mathbf{t}) - \mathbf{t}$ in the GSO basis $\mathbf{B}_k^*$. Each coordinate c_i (coefficient of $\mathbf{b}_i^*$) is subgaussian with parameter $\sqrt{2\pi}$ and is independent of the previous coordinates. As in Lemma A.1's proof, the result follows from the definition of the GSO. □