A Generator for LWE and Ring-LWE Instances

Martin R. Albrecht[1], Daniel Cabarcas[2], Robert Fitzpatrick[3], Florian Göpfert[2] and Michael Schneider[2]

[1] Technical University of Denmark, Kgs. Lyngby
[2] Technische Universität Darmstadt
[3] Royal Holloway, University of London

Abstract. We introduce software for the generation of instances of the LWE and Ring-LWE problems, allowing both the generation of generic instances and also particular instances closely-related to those arising from cryptomania proposals in the literature. Our goal is to allow researchers to attack different instances in order to assess the practical hardness of LWE and Ring-LWE. This will in turn give insight to the practical security of cryptographic systems based on both problems.

Keywords. LWE, Ring-LWE, Lattice-Based Cryptography

1 Introduction

The LWE and Ring-LWE problems (reproduced in Definitions 1 and 2) have received widespread attention from the cryptographic community in recent years. Promising encryption systems have been proposed with LWE and Ring-LWE as security background. So far, algorithms to solve both problems have been rarely tested. For this purpose this work describes a public generator system for LWE and Ring-LWE instances.

Definition 1 (LWE). *Let $n \geq 1$ be a positive integer, q be an odd prime, χ be a probability distribution on $\mathbb{Z}_q$ and $\boldsymbol{s}$ be a secret vector in $\mathbb{Z}_q^n$. We denote by $L_{\mathbf{s},\chi}^{(n)}$ the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ at random, choosing $e \in \mathbb{Z}_q$ according to χ, and returning $(\boldsymbol{a}, c) = (\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.*
Decision-LWE *is the problem of distinguishing $L_{\mathbf{s},\chi}^{(n)}$ from the uniform distribution $\mathcal{U}(\mathbb{Z}_q^n \times \mathbb{Z}_q)$ on $\mathbb{Z}_q^n \times \mathbb{Z}_q$.*
Search-LWE *is the problem of finding $\boldsymbol{s} \in \mathbb{Z}_q^n$ given pairs $\mathbb{Z}_q^n \times \mathbb{Z}_q$ sampled according to $L_{\mathbf{s},\chi}^{(n)}$.*

Under certain conditions on the modulus and noise distribution χ (which we do not discuss here), solving certain (assumed hard) worst-case lattice problems can be reduced to solving (average-case) LWE. In recent work [BLP+13], the authors go beyond the *quantum* reduction of Regev [Reg05], to give a classical reduction, further bolstering confidence in the hardness of LWE.

Definition 2 (Ring-LWE (informal)). *Let $n \geq 1$ be a power of 2 and let $q \equiv 1 \mod 2n$, $q \in \mathrm{poly}(n)$ be a prime modulus. Let $f(x) = x^n + 1 \in \mathbb{Z}[x]$, implying that $f(x)$ is irreducible over $\mathbb{Q}$. Set $R = \mathbb{Z}[x]/\langle f(x) \rangle$, the ring of integer polynomials modulo $f(x)$ and set $R_q = R/\langle q \rangle$. Then the ring-LWE problem can (informally) be defined as: choosing $s \in R_q$ to be a uniformly random ring element and defining an error distribution χ on R such that the 'weight' of χ (in general terms) is concentrated on 'small' elements of R. Similarly to decision-LWE, the (decision) Ring-LWE problem is to distinguish between pairs $(a, a * s + e)$ (where $a \leftarrow \mathcal{U}(R_q)$, $e \leftarrow \chi$ and $*$ denotes multiplication in R_q) and pairs $(a, c) \leftarrow \mathcal{U}(R_q \times R_q)$.*

It should be noted that, despite the significant extra structure (and thus enhanced efficiency in constructions based on Ring-LWE) present in instances of the Ring-LWE problem, no algorithms are currently known which can exploit this structure in a significant manner, thus the present algorithms for the LWE and Ring-LWE problems are essentially the same.

The popularity of these problems as a base for cryptographic constructions is partly due to their remarkable flexibility and partly due to their strong asymptotic hardness guarantees. However, the concrete cost of solving specific instances of these problems has not received widespread attention. Even when reports on the concrete hardness of particular instances are available, these are usually not easily comparable with other results, as no unified set of benchmarks for algorithms solving (Ring-)LWE are available.

This work aims to provide such benchmark instances and to facilitate research on the concrete hardness of LWE instances by making easy-to-use instance generators for the LWE and Ring-LWE problem available. This includes both generic classes for these problems (`LWE` and `RingLWE`) as well as specific generators for various proposals from the literature [Reg09,LP11,CGW13]. Our generators are written for and submitted for inclusion in the Sage mathematics software [S$^+$13]. We start with an example where we construct an LWE oracle following [Reg09]:

```
sage: from lwe import Regev
sage: Regev(n=128)  # Regev's parameters with security parameter n=128
LWE(128, 16411, DiscreteGaussianSamplerRejection(11.809841, 16411, 4), 'uniform', None)
```

Our second example demonstrates how to use the oracle and what the output looks like:

```
sage: from lwe import Regev
sage: lwe = Regev(n=10)  # Regev's parameters with security parameter n=10
sage: lwe()
((5, 40, 37, 23, 62, 29, 29, 75, 81, 88), 51)
```

The output shows the tuple $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s}\rangle + e)$.

2 The Generator

The `LWE` and `RingLWE` constructors accept as parameter a noise distribution $\mathcal{D}$ which, when invoked, returns an element in $\mathbb{Z}$. We provide instances of such noise distribution oracles called "samplers" in our work. Somewhat contrary to intuition, sampling faithfully from Gaussian and (in particular) discretised Gaussian distributions is not a straight-forward task. The approach we employ is simple rejection sampling, in which elements are drawn uniformly at random from the support of a distribution, then rejected with probability inversely proportional to the probability density at the chosen point.

`DiscreteGaussianSamplerRejection` samples element in $\mathbb{Z}$ according to a a discrete Gaussian distribution centred at zero with standard deviation σ. We stress that this sampler is parameterised by the target standard deviation instead of the parameter $s = (\sigma \cdot \sqrt{2\pi})$ often found in the literature. We give an example below:

```
sage: from lwe import DiscreteGaussianSamplerRejection
sage: D = DiscreteGaussianSamplerRejection(3.0)
sage: variance([D() for _ in range(2000)]).sqrt().n()
3.01445160173946
```

Note that rejection sampling is currently the default strategy for sampling from a discrete Gaussian distribution. Hence, the default Gaussian sampler `DiscreteGaussianSampler` points to `Discrete-GaussianSamplerRejection`.

Some recent works (e.g., [GLP12,MP13]) raise the possibility of employing the uniform distribution over a small subset of $\mathbb{Z}$ in place of a discrete Gaussian, thus we include `UniformSampler` which samples uniformly between a lower and an upper bound as illustrated in the following example:

```
sage: from lwe import UniformSampler
sage: D = UniformSampler(-2,2)
sage: L = [D() for _ in range(2000)]
sage: [L.count(i) for i in (-2,-1,0,1,2)]
[399, 409, 380, 395, 417]
```

For each of these classes there also exist analogous polynomial variants for Ring-LWE. Here, a polynomial in R_q is generated by sampling its coefficients independently from Gaussian or uniform distribution.

The basic `LWE` class characterises LWE instances by the dimension n, a modulus q and a noise distribution $\mathcal{D}$ defined over $\mathbb{Z}$. We can construct LWE instances as follows:

```
sage: from lwe import DiscreteGaussianSampler, LWE
sage: D = DiscreteGaussianSampler(3)
sage: LDist = LWE(n=20, q=401, D=D)
sage: LDist
LWE(20, 401, DiscreteGaussianSamplerRejection(3.000000, 53, 4), 'uniform', None)
sage: LDist()
((19, 118, 249, 127, 347, 269, 232, 383, 16, 265, 247, 133, 102, 74, ... , 272), 63)
```

The `LWE` class also accepts a parameter `secret_dist` which may be either "uniform" or "noise". In the later case, the secret is sampled from the same distribution as the noise, namely $\mathcal{D}$. It also accepts a parameter `m` to limit the number samples returned, as is the case in some cryptosystem proposals, for instance [LP11]. After this limit is reached an `IndexError` is raised whenever another sample is requested. We highlight this using `RingLWE`:

```
sage: from lwe import RingLWE, DiscreteGaussianPolynomialSampler
sage: D = DiscreteGaussianPolynomialSampler(euler_phi(8), stddev=3)
sage: rlwe = RingLWE(N=8, q=next_prime(400), D=D, m=5); rlwe
RingLWE(8, 401, DiscreteGaussianPolynomialSamplerRejection(4, 3.000000, 53, 4), x^4 + 1, '
    uniform', 5)
sage: rlwe() # note that samples are tuples of vectors over IntegerModRing(q)
((314, 333, 270, 367), (89, 276, 152, 388))
sage: _ = [ rlwe() for _ in range(4) ]
sage: rlwe()
...
IndexError: Number of available samples exhausted.
```

2.1 Instances from the Literature

We also provide high-level interfaces to generate LWE instances from the literature. In particular, the following instance generators are supported.

1. `Regev` instantiates an LWE oracle given a security parameter n following [Reg09].

```
sage: from lwe import Regev
sage: Regev(128)
LWE(128, 16411, DiscreteGaussianSamplerRejection(11.809841, 16411, 4), 'uniform', None)
```

2. `LindnerPeikert` instantiates an LWE oracle given a security parameter n following [LP11].

```
sage: from lwe import LindnerPeikert
sage: LindnerPeikert(128)
LWE(128, 2053, DiscreteGaussianSamplerRejection(2.705800, 53, 4), 'noise', None)
```

3. `UniformNoiseLWE` instantiates an LWE oracle given a security parameter n following [CGW13].

```
sage: from lwe import UniformNoiseLWE
sage: UniformNoiseLWE(128)
LWE(128, 389164331, UniformSampler(0, 486), 'noise', 177)
```

4. `RingLindnerPeikert` instantiates a Ring-LWE oracle given a security parameter n following a generalisation of [LP11].

   ```
   sage: from lwe import RingLindnerPeikert
   sage: RingLindnerPeikert(128)
   RingLWE(128, 2053, DiscreteGaussianPolynomialSamplerRejection(64, 3.050908, 53, 4), x^64 +
       1, 'noise', None)
   ```

2.2 Utility Functions

The function `samples` provides easy, one-line access to all generators. It accepts a number m requested samples, a security parameter n, an LWE instance generator, a seed for the random number generator and a special parameter "balanced": by default, elements in $\mathbb{Z}_q$ are represented as integers in $x \in [0, q-1]$ by Sage. If, instead, we wish to use the balanced representation $x \in [-\lfloor q/2 \rfloor, \lfloor q/2 \rfloor]$, the parameter "balanced" can be used to apply the balanced representation to scalars and vectors in and over $\mathbb{Z}_q$ (with output given as integers or $\mathbb{Z}$-module elements, respectively).

```
sage: S = samples(20, 10, 'RingLindnerPeikert', seed=1337); S
[((706, 21, 602, 420), (197, 136, 639, 177)),
 ...
 ((709, 855, 682, 57), (504, 166, 894, 963))]
sage: len(S)
20
```

With balanced-representation output:

```
sage: samples(20, 10, 'RingLindnerPeikert', seed=1337, balanced=True)
[((-325, 21, -429, 420), (197, 136, -392, 177)),
 ...
 ((-322, -176, -349, 57), (504, 166, -137, -68))]
```

3 More Information

3.1 Availability

The source code is available at https://www.bitbucket.org/malb/lwe-generator and has also been submitted for inclusion in Sage [S+13].

The generator is also available as an interactive service at:

http://aleph.sagemath.org/?z=eJzLyU9M0VDKKCkpKLbS10_KLEkqTc5OLdHLL0rXz03MSdLPKU_VTU_NSy1KLMkv0i9KLNcvySwAieoVVCpp8nIBWQq2CkGp6allGnm2xgZAoYKizLwSBaAEWFZDEwD2OyF9&lang=sage

3.2 Further Documentation

For further documentation we refer the reader to the module itself and Sage's documentation, available at http://sagemath.org/doc/ .

3.3 Mailing List

We have established a mailing list for the purposes of collecting and sharing experiences with the different LWE instances and also for any questions directed to the authors regarding the project. The list also covers ideas for new LWE instances which should be integrated into the generator. The mailing list can be reached via

http://groups.google.com/group/lwe-challenge-devel.

Acknowledgements

This work was supported by CASED (www.cased.de)

References

BLP$^+$13. Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of Learning with Errors. to appear STOC 2013, 2013.

CGW13. Daniel Cabarcas, Florian Göpfert, and Patrick Weiden. Provably secure LWE-encryption with uniform secret. *IACR Cryptology ePrint Archive*, page 164, 2013.

GLP12. Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES*, volume 7428 of *LNCS*, pages 530–547. Springer, 2012.

LP11. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *CT-RSA*, volume 6558 of *LNCS*, pages 319–339. Springer, 2011.

MP13. Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. *IACR Cryptology ePrint Archive*, 2013:69, 2013.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93. ACM, 2005.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

S$^+$13. W. A. Stein et al. *Sage Mathematics Software*. The Sage Development Team, 2013. http://www.sagemath.org.