Ring-LWE: An Efficient PQC Public Key Encryption Scheme

N.P. Smart

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB.
United Kingdom.

April 23, 2016

Outline

Lattices

Lattice Reduction

Lattices and Rings

Standard and Ring LWE

Basic Ring-LWE Encryption Scheme

Coding Theory

Fourier Transforms

CCA Secure Scheme

Lattices

Lattice Definition

A discrete additive subgroup of $\mathbb{R}^m$.

Let

$$B = \{b_1, \ldots, b_n\}$$

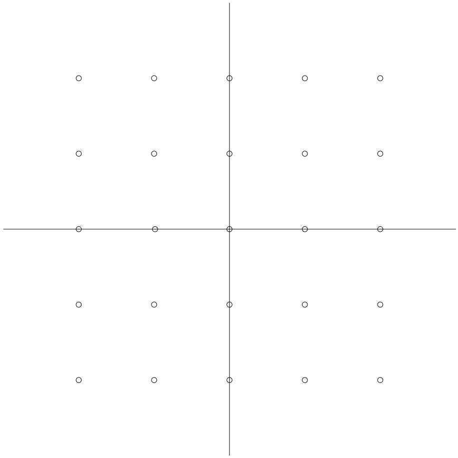
be an ordered set of vectors in $\mathbb{R}^m$. With this we generate a lattice

$$L_B = \left\{ \sum_{i=1}^{n} \lambda_i b_i | \lambda_i \in \mathbb{Z} \right\}.$$

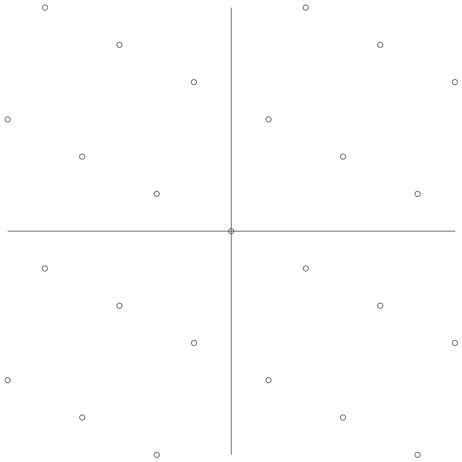
We shall also denote $(b_1, \ldots, b_n)^\top$ by B.

- B is called the basis matrix

Illustrative Example 1: $\mathbb{Z}^2$



Illustrative Example 2:



Lattice Bases

If B_1 and B_2 are bases of the same lattice if and only if

$$B_1 = UB_2$$

for some integer matrix U with determinant 1 or -1.
We define the fundamental parallelepiped associated to B as

$$P(B) = \left\{ \sum_{i=1}^{n} x_i b_i | x_i \in [0, 1) \right\}.$$

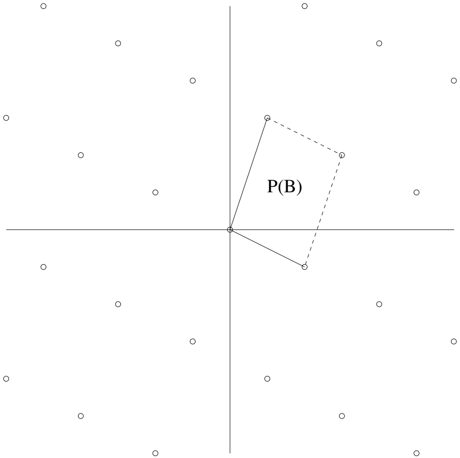
Roughly speaking

$$\{P(B) + v | v \in L_B\}$$

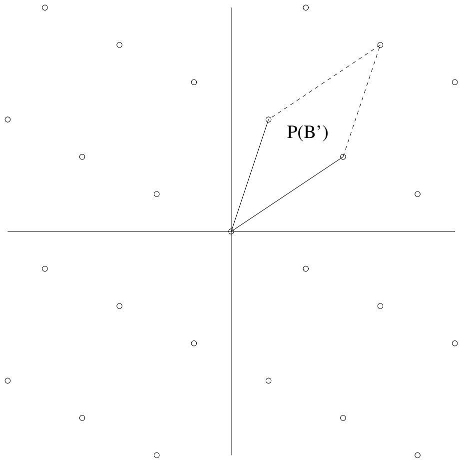
are the regions between the lattice points.

- They partition $\mathbb{R}^m$.

Fundamental Parallelepiped Illustration



Fundamental Parallelepiped Illustration 2



Lattice Determinant

We define a lattice determinant $\det(L_B)$ to be the volume of $P(B)$.

$$\det(L_B) = \mathrm{Vol}(P(B)) = \prod_i \|b_i^*\| = |\det(B^*)| \,.$$

If $m = n$, then

$$\det(L_B) = |\det(B^*)| = |\det(B)| \,.$$

More generally

$$\det(L_B) = \sqrt{|\det(B^\top B)|}.$$

The determinant is well-defined since bases differ by a factor of $\pm U$ where U is unimodular.

SVP and CVP

Given a lattice basis B.

The SVP problem is the problem of finding a vector $v \in L_B \setminus \{0\}$ such that $\|v\|_2$ is minimal.

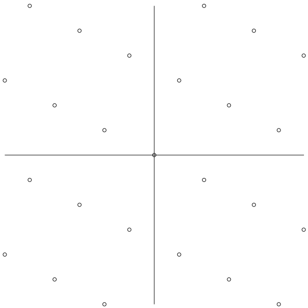
The CVP problem is the problem of given $u \notin L_B$ find a vector $v \in L_B$ such that $\|v - u\|_2$ is minimal

The are known to be hard problems.

- Suspected exponential complexity

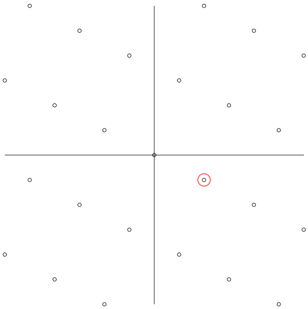
Easy SVP Problem

If we can see the complete lattice, the SVP problem is trivial.



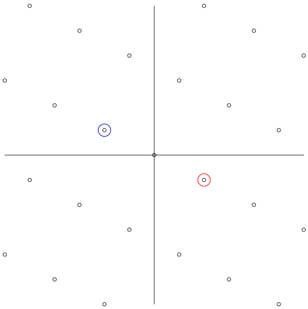
Easy SVP Problem

If we can see the complete lattice, the SVP problem is trivial.



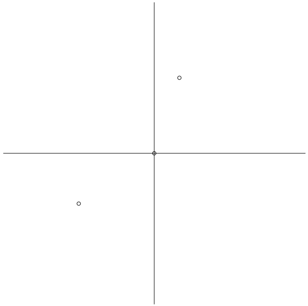
Non-Uniqueness

Note also that the shortest vector is not unique.

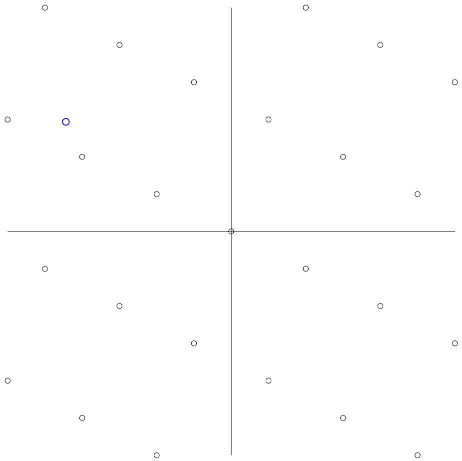


Slightly Harder SVP Problem

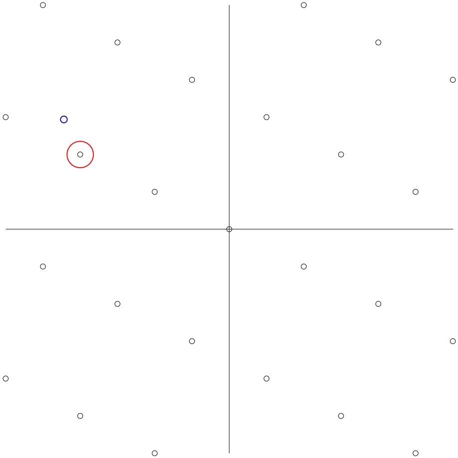
Given only a random basis, the SVP becomes harder.



CVP Illustration



CVP Illustration



Lattice Reduction

Gaussian Reduced Basis

Can often solve CVP/SVP using basis reduction

In two dimensions this is a classical algorithm

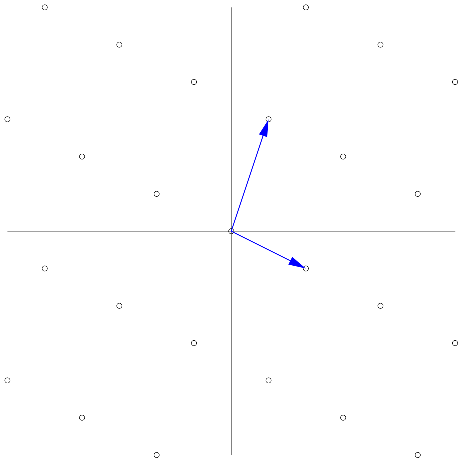
A basis $\{b_1, b_2\}$ is said to be Gaussian Reduced if

1. $|\mu_{1,2}| \leq \frac{1}{2}$
2. $\|b_1\| \leq \|b_2\|$.

where

$$\mu_{1,2} = \frac{\langle b_2, b_1 \rangle}{\langle b_1, b_1 \rangle}$$

Reduced Basis Illustration



Gaussian Reduction Algorithm

DO

1. IF $\|b_1\| > \|b_2\|$ THEN swap b_1 and b_2
2. $\mu_{1,2} = \frac{\langle b_2, b_1 \rangle}{\langle b_1, b_1 \rangle}$
3. $b_2 := b_2 - \lceil \mu_{1,2} \rfloor b_1$

WHILE $\|b_1\| > \|b_2\|$.

RETURN $(b_1, b_2)^\top$.

To obtain a Speed/Approximation Trade-off we introduce $1/2 < \delta < 1$ and replace the second condition.

1. $|\mu_{1,2}| \leq \frac{1}{2}$
2. $\delta \|b_1\| \leq \|b_2\|$.

Generalisation to *n* Dimensions

There are various ways we could attempt to generalise Gaussian Reduction to n dimensions.

Lovasz' solution, named after Lenstra, Lenstra and Lovasz, is the LLL algorithm.

It outputs a roughly orthogonal basis comprising of some short vectors.

It terminates in polynomial time in n.

An LLL Reduced Basis

1. $|\mu_{i,j}| \leq \frac{1}{2}$ for all $1 \leq j < i \leq n$.
2. $\delta \left\| b_i^* \right\|^2 \leq \left\| b_{i+1}^* + \mu_{i+1,i} b_i^* \right\|^2$ for $i = 1 \ldots n - 1$.

- $\mu_{i,j}$ are the standard Gram-Schmidt coefficients.
- We say B is *LLL reduced with respect to* δ, (or $\delta - LLL$ reduced) if both conditions are satisfied.

An LLL Reduced Basis

The first LLL condition produces an approximation to Gram-Schmidt.

- We are forced to approximate since our basis must span the same lattice.
- Makes the basis roughly orthogonal.

The second condition makes the basis vectors small, and roughly in increasing size

- So the first basis vector is an appromixation to the short-vector in the lattice

Lattices and Rings

Size Matters

A major issue in using lattice in cryptography is we seem to need to hold a lot of data.

After all a lattice basis requires storing $n \times n$ elements!

We want to be able to reduce this to n elements.

For this we use rings of polynomials

$$R = \mathbb{Z}[X]/(F(X)),$$

where $\deg(F) = n$.

Polynomial Rings

Clearly an element in a ring of polynomials will require *n* elements to define it

- One for each coefficient.

$$\mathbf{a} \longmapsto a(X).$$

This is the *vector representation* of the ring.

- We can clearly add vectors/polynomials.

Polynomial Rings

We can also, in a polynomial ring, multiply vectors/polynomials to get another polynomial of degree n

$$c = a \cdot b \pmod{F}.$$

We can think of this as vectors by looking at the *matrix representation* of the ring

$$\mathbf{c} = M_a \cdot \mathbf{b}.$$

i.e. M_a is a matrix (depending on a) which acts like multiplication by a on vectors.

Matrix Representation

So what does M_a represent?

All the vectors $\mathbf{c} = M_a \cdot \mathbf{b}$ as $\mathbf{b} \in \mathbb{Z}^n$ define a lattice

This is the lattice of the ideal of R generated by the polynomial a

Finding a good basis for the lattice generated by M_a is equivalent to finding a short generator of the ideal generated by a.

$$\begin{aligned} \langle a \rangle &= \{c(X) = a(X) \cdot b(X) \pmod{F(X)} : b(X) \in R\} \\ &\equiv \{\mathbf{c} = M_a \cdot \mathbf{b} : \mathbf{b} \in \mathbb{Z}^n\}. \end{aligned}$$

Standard and Ring LWE

Linear Algebra With Noise

LWE (Learning With Errors) is basically linear algebra with noise.

In usual linear algebra we try to solve the equation

$$\mathbf{y} = A \cdot \mathbf{x}$$

for some matrix A

In LWE we do not give you $\mathbf{y}$ but a vector with some errors in

$$\mathbf{y}' = A \cdot \mathbf{x} + \mathbf{e},$$

where $\mathbf{e}$ is "small" in some sense.

We can think of this as like a decoding problem for the linear code defined by the matrix A.

Or equivalently this is a CVP problem, as $\mathbf{e}$ is small.

Linear Algebra With Noise

However, using the above definition in crypto is a bit awkward as numbers are unbounded

- All vectors are in $\mathbb{Z}^n$
- We would like to work modulo an integer q, to bound the sizes

So we define our problem as

$$\mathbf{y} = A \cdot \mathbf{x} + \mathbf{e} \pmod{q},$$

i.e. we are given $\mathbf{y} \in \mathbb{Z}_q^n$ and are asked to find $\mathbf{x}$, given $\mathbf{e} \in \mathbb{Z}_q^n$ has "small coefficients"

- Small means when reduced into the interval $(-q/2, q/2)$.

Linear Algebra With Noise

This is actually another lattice problem, for the non square generating matrix $(A \mid q \cdot I_n)$...

$$\mathbf{y} = (A \mid q \cdot I_n) \cdot \mathbf{x} + \mathbf{e}.$$

We can also define a Ring version of LWE (called Ring-LWE)

Given a polynomial a we are asked to solve

$$y = (a \cdot x + e \pmod{F}) \pmod{q}$$

for a polynomial e with small coefficients.

The underlying ring is

$$R_q = \mathbb{Z}_q[X]/(F(X)).$$

C'Mon Feel The Noise

So how do we pick the small noise vector/polynomial?

There are various ways we *could* do this

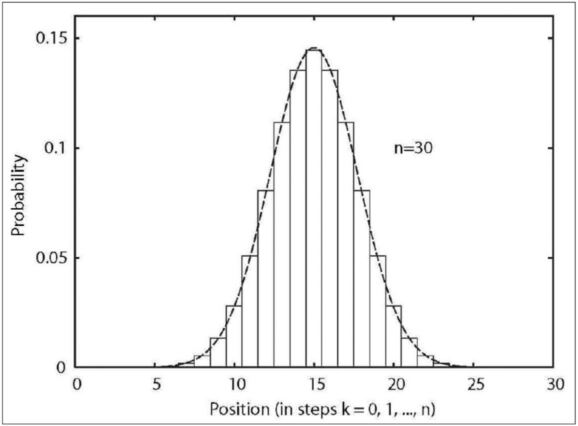
- Sample unformly from $[-b, \ldots, b]$ for a small value of b.
- Sample from $[-b, \ldots, b]$ using a non-uniform distribution.
- Sample (in the ring case) from the "complex embedding" in some way and then "pull back"

Each method has its own advantages/disadvantages

- Ease of implementation.
- Ease of analysis of resulting bounds.
- Worst-case/Average-case results (mainly for theory wonks!).

Gaussian Noise

For simplicity we will use sampling of an approximation to the discrete Gaussian...



Sampling Made Easy

Take a normal Gaussian of standard deviation σ.

The probability of being more than $6 \cdot \sigma$ away from the mean is so small we can ignore it.

So we create a table which approximates the CDF from $-6 \cdot \sigma$ upto $6 \cdot \sigma$

1. Let B be the smallest power of two which is larger than $6 \cdot \sigma$.
2. Initialize DGtable as a table indexed by 0 to $2 \cdot B - 1$.
3. For i from $-B$ to $B - 1$ do
   3.1 $x \leftarrow (x + .5)/(\sigma \cdot \sqrt{2})$.
   3.2 $v \leftarrow (1 + \mathrm{erf}(x))/2$.
   3.3 $j \leftarrow \lceil 2^{32} \cdot v \rfloor$.
   3.4 $\mathrm{DGtable}[i + B] = j$.
4. Return $[B, \mathrm{DGtable}]$.

Sampling Made Easy

To sample we then just pull a uniform random number in the range $[0, \ldots, 2^{32}]$ and do a binary search

1. $x \leftarrow \text{rand}(2^{32})$.
2. $l \leftarrow 0, u \leftarrow 2 \cdot B$.
3. While $(u - l) > 1$ do
   3.1 $m \leftarrow \lfloor (l + u)/2 \rfloor$.
   3.2 If $x > \text{DGtable}[m]$ then $l = w$, else $u = m$.
4. Return $(u - B)$.

The resulting distribution we call χ_σ.

Basic Ring-LWE Encryption Scheme

Key Generation

A public key consists of a Ring-LWE instance (a, b), a private key is the underlying secret polynomial s such that

$$b = a \cdot s + e',$$

where e' is "small"

- We pick however s also to be "small"

So key generation goes as follows:

1. $a \leftarrow R_q$.
2. $s, e' \leftarrow \chi_\sigma^n$.
3. $b = a \cdot s + e'$ in R_q.

Encryption

This works by randomizing the LWE instance given by the public key and then embedding the message $\mu \in \{0,1\}^n$

1. $v, e, d \leftarrow \chi_\sigma^n$.
2. $c_0 = b \cdot v + d + \Delta_q \cdot \mu$.
3. $c_1 = a \cdot v + e$

where

$$\Delta_q = \left\lfloor \frac{q}{2} \right\rfloor .$$

Note

$$\begin{aligned} c_0 - s \cdot c_1 &= (b \cdot v + d + \Delta_q \cdot \mu) - s \cdot (a \cdot v + e) \\ &= ((a \cdot s + e') \cdot v + d + \Delta_q \cdot \mu) - s \cdot (a \cdot v + e) \\ &= \Delta_q \cdot \mu + e' \cdot v + d - e \cdot s \\ &= \Delta_q \cdot \mu + \text{“small”}. \end{aligned}$$

Decryption

Since we have

$$c_0 - s \cdot c_1 = \Delta_q \cdot \mu + \text{“small”}.$$

we decrypt as follows:

1. $f = c_0 - s \cdot c_1$.
2. $\mu = \left\lfloor \left\lceil \frac{2}{q} \cdot f \right\rfloor \right\rfloor$.

BUT this only works if the “small” is small enough.

- We need to fix the parameters so that small is small enough

Three Problems

If we fix the parameters so that “small” is guaranteed to be **always** small enough we get huge parameters

- Like those seen in FHE schemes!

So we need to apply some coding theory to make the parameters smaller.

We do lots of operations like $a \cdot b \pmod{F}$ which are costly, $O(n^2)$.

- Would prefer $O(n)$

If we pick F and q well we can use FFT techniques

The above scheme is not CCA secure it is only CPA secure

- Need a way of transforming to get a CCA secure scheme

We now address these three problems.

Coding Theory

Coding Theory

We could just apply some complex BCH code to get a really excellent encoding scheme

- Good error correction properties.
- Complex to code correctly/well due to complex decoding algorithm.
- Open source good implementations are available, but they are polluted by the GPL.

However, these are overkill as we only need to correct very few bits in practice.

We pick a $(16, 8, 5)$ linear code.

This takes a byte and expands it into two bytes

- Allows us to correct a byte in the presence of two error bits in the 16

The $(16, 8, 5)$ Code

Generator Matrix:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \in \mathbb{F}_2^{8\times 16}$$

The $(16, 8, 5)$ Code

Parity Check Matrix:

$$H = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \in \mathbb{F}_2^{8 \times 16}$$

The $(16, 8, 5)$ Code

To encode we take a byte b and represent it as a bit vector $\mathbf{b} \in \mathbb{F}_2^8$ and then compute

$$\mathbf{c} = \mathbf{b} \cdot G.$$

To decode we compute the syndrome

$$\mathbf{s} = H \cdot \mathbf{c}.$$

We can then look up the resulting decoding value in a simple look up table of size $137 = 1 + 16 + (16 \cdot 15)/2$.

- LUT has one entry for each zero, one and two bit error.
- LUT gives the error vector which we can then add onto $\mathbf{c}$ to recover the actual message.

Coding Theory

If the per-bit error rate is p_0 then the probability we recover a byte correctly using this encoding scheme is

$$p_1 = (1 - p_0)^{16} + 16 \cdot (1 - p_0)^{15} \cdot p + \frac{16 \cdot 15}{2} \cdot (1 - p_0)^{14} \cdot p_0^2.$$

When $p_0 \approx 2^{-18}$ we obtain

$$p_1 \approx 1 - 2^{-44}.$$

Coding Theory

Given a vector $\mathbf{b}$ in $\mathbb{F}_2^n$ where n is a multiple of eight we can apply the encoding scheme $n/8$ times in parallel to obtain a vector of length $2 \cdot n$.

$$\mathbf{c} = \text{Encode}(\mathbf{b}).$$

Likewise we can decode a bit-vector vector of length $2 \cdot n$ to a bit-vector vector of length n via

$$\mathbf{b} = \text{Decode}(\mathbf{c}).$$

If $p_0 \approx 2^{-18}$ probability of transmitting 64 bytes (i.e. 512 bits) correctly is

$$p_2 = p_1^{64} \approx 1 - 2^{-38}.$$

We encode the 64 bytes as 128 bytes.

Fourier Transforms

Fourier Transforms

We pick F and q for our Ring LWE problem in a special way

1. $F = X^N + 1 = X^{2^n} + 1$
2. $q \equiv 1 \pmod{2 \cdot N}$.

This implies that $\mathbb{F}_q$ contains a $(2 \cdot N)$th root of unity.

- Let α denote a mutually agreed one.

Since $F = X^{2^n} + 1$ picking our noise via sampling Gaussian coefficients is essentially the same as sampling in the canonical embedding and pulling back.

Idea is to represent a polynomial $a(X)$ by its evaluations at the roots of unity α^i for odd $i \in [1, \ldots, 2 \cdot N]$.

Fourier Transforms

Evaluating $a(X)$ at the odd powers of α is the same as evaluating the FFT of $a(X)$.

$$\mathbf{a} \leftarrow \mathrm{FFT}(a) \in \mathbb{F}_q^N.$$

Interpolating the vector back to a polynomial is the same as evaluating the inverse FFT of **a**.

$$a \leftarrow \mathrm{FFT}^{-1}(\mathbf{a}) \in R_q.$$

Since N is a power of 2 the FFT is fast

Fourier Transforms

If we define $\oplus$ and $\otimes$ as coordinate wise addition and multiplication modulo q then we have

$$\mathbf{a} \oplus \mathbf{b} = \mathsf{FFT}(a+b),$$
$$\mathbf{a} \otimes \mathbf{b} = \mathsf{FFT}(a \cdot b).$$

This means that operations in our ring can be done in time $O(n)$ as soon as we have passed into the FFT domain.

- Mapping to/from FFT domain takes time $O(n \cdot \log n)$.

CCA Secure Scheme

CCA Parameters

We are going to need a ring size of $N = 1024$ bits

- Want to transmit keys of size 256 bits.
- Padding scheme needs 256 bits of randomness.
- Our encoding scheme doubles this to $2 \cdot (256 + 256) = 1024$.

We pick a Gaussian standard deviation of $\sigma = 3.2$

We therefore pick $q = 765953$

- Gives good security.
- Per bit error rate of $p_0 \approx 2^{-18}$.

We first define a CPA secure scheme which includes the FFT and coding theory modifications...

CPA Secure Scheme: KeyGen

1. $a \leftarrow R_q$.
2. $s, e' \leftarrow \chi_\sigma$.
3. $\mathbf{a} \leftarrow \mathsf{FFT}(a)$.
4. $\mathbf{s} \leftarrow \mathsf{FFT}(s)$.
5. $\mathbf{e}' \leftarrow \mathsf{FFT}(e')$.
6. $\mathbf{b} \leftarrow (\mathbf{a} \otimes \mathbf{s}) \oplus \mathbf{e}'$.
7. $\mathfrak{sk} \leftarrow \mathbf{s}$.
8. $\mathfrak{pk} \leftarrow (\mathbf{a}, \mathbf{b})$.
9. Return $(\mathfrak{pk}, \mathfrak{sk})$

CPA Secure Scheme: $\mathsf{Enc}_1(m, \mathfrak{pk})$

The encryption mechanism takes as input the public key $\mathfrak{pk} = (\mathbf{a}, \mathbf{b})$ and a message $m \in \{0, 1\}^b$, where $b < N/2$.

1. $\mu \leftarrow \mathsf{Encode}(m)$, treat μ as an element in R_2 (this involves applying Encode a total of $\lceil b/8 \rceil$ times).
2. $v, e, d \leftarrow \chi_\sigma$.
3. $\mathbf{v} \leftarrow \mathsf{FFT}(v)$, $\mathbf{e} \leftarrow \mathsf{FFT}(e)$.
4. $x \leftarrow d + \Delta_q \cdot \mu \bmod q$.
5. $\mathbf{x} \leftarrow \mathsf{FFT}(x)$.
6. $\mathbf{c}_0 \leftarrow (\mathbf{b} \otimes \mathbf{v}) \oplus \mathbf{x}$.
7. $\mathbf{c}_1 \leftarrow (\mathbf{a} \otimes \mathbf{v}) \oplus \mathbf{e}$.
8. Return $(\mathbf{c}_0, \mathbf{c}_1)$.

CPA Secure Scheme: $\text{Dec}_1(m, \mathfrak{pk})$

On input of a ciphertext $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ and a secret key $\mathfrak{sk} = \mathbf{s}$ the decryption is performed as follows:

1. $\mathbf{f} \leftarrow \mathbf{c}_0 \ominus (\mathbf{s} \otimes \mathbf{c}_1)$.
2. $f \leftarrow \text{FFT}^{-1}(\mathbf{f})$.
3. Convert f into centered-representation.
4. $\mu \leftarrow \left| \left\lfloor \frac{2}{q} f \right\rceil \right|$. Note μ can be considered as a string of $2 \cdot b$ bits, as $N > 2 \cdot b$ we only take the first $2 \cdot b$ bits of μ and ignore all non-zero trailing bits as "errors".
5. $m \leftarrow \text{Decode}(\mu)$.
6. Return m.

CCA From CPA

To define a CCA scheme from the above CPA scheme we use the Fujisaki-Okamoto transform:

If the original encryption scheme $(\mathrm{Enc}_1, \mathrm{Dec}_1)$ can encrypt messages of b bits in length, this this IND-CCA scheme encrypts messages of $b - 256$ bits in length.

The scheme makes use of a hash function H which outputs at least 256-bit hash values, and takes as inputs bit strings of length b.

CCA Secure Pubic Key Scheme

$\text{Enc}_2(m, \mathfrak{pk})$:

1. $s \leftarrow \{0, 1\}^{256}$.
2. $\mu \leftarrow m \| s$.
3. $r \leftarrow H(\mu)$.
4. $(\mathbf{c}_0, \mathbf{c}_1) \leftarrow \text{Enc}_1(\mu, \mathfrak{pk})$, where all randomness is generated from the seed r.
5. Return $(\mathbf{c}_0, \mathbf{c}_1)$.

$\text{Dec}_2(\mathbf{c}, \mathfrak{sk})$:

1. $\mu \leftarrow\leftarrow \text{Dec}_1(\mathbf{c}, \mathfrak{sk})$.
2. $m \| s \leftarrow \mu$, where s is 256 bits long.
3. $r \leftarrow H(\mu)$.
4. $\mathbf{c}' \leftarrow \text{Enc}_1(\mu, \mathfrak{pk})$, where all randomness is generated from the seed r.
5. If $\mathbf{c} \neq \mathbf{c}'$ then return $\perp$.
6. Return m.

CCA Secure Key Encapsulation

For key-encapsulation in post-quantum schemes we aim to transmit a key of 256 bits in length.

Thus we can easily adapt the previous IND-CCA encryption scheme to do this by selecting a message of size 256 bits at random.

- So $b - 256 = 256$.
- Which implies $b = 512$.
- So $N > 2 \cdot 512 = 1024$.
- This is why we wanted $N = 1024$ for our ring.

CCA Secure Key Encapsulation

Key Encapsulation: This takes as input a public key $\mathfrak{pk}$ and outputs an encapsulation $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$ and the key $\mathfrak{k}$ it encapsulates.

1. $\mathfrak{k} \leftarrow \{0, 1\}^{256}$.
2. $(\mathbf{c}_0, \mathbf{c}_1) \leftarrow \mathsf{Enc}_2(\mathfrak{k}, \mathfrak{pk})$.
3. Return $((\mathbf{c}_0, \mathbf{c}_1), \mathfrak{k})$.

Key Decapsulation: This takes as input a secret key key $\mathfrak{sk}$ and an encapsulation $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$, and outputs the key $\mathfrak{k}$ it encapsulates, or the error symbol $\perp$.

1. $\mathfrak{k} \leftarrow \mathsf{Dec}_2(\mathbf{c}, \mathfrak{sk})$.

Questions?