Attacks on Search-RLWE

No Author Given

No Institute Given

Abstract. We describe a new attack on the Search Ring Learning-With-Errors (RLWE) problem based on the chi-square statistical test, and give examples of RLWE instances in Galois number fields which are vulnerable to our attack. We prove a search-to-decision reduction for Galois fields which applies for any unramified prime modulus q, regardless of the residue degree f of q, and we use this in our attacks. The time complexity of our attack is $O(q^{2f})$, where f is the *residue degree* of q in K.
We also show an attack on the RLWE problem in general cyclotomic rings (non 2-power cyclotomic rings) which works when the modulus is a ramified prime. We demonstrate the attacks in practice by finding many vulnerable instances and successfully attacking them. We include the code for all attacks.

1 Introduction

The Ring Learning-with-Errors (RLWE) problem, proposed in [13], is a variant of the traditional Learning-with-Errors (LWE) problem, and is an active research area in lattice based cryptography. It has drawn increased attention due to the important application to constructing homomorphic encryption schemes ([3,2,4,8,16,12,1]).

Central to an RLWE problem instance is a choice of a number field K and a prime q called the *modulus*. The authors of [13] considered the case where K is some cyclotomic field, and proved a reduction from certain hard lattice problems to the dual variant of RLWE. The hardness for the non-dual variant was proved in [5]. Also in [13], a search-to-decision reduction was proved for RLWE problems for cyclotomic fields and modulus q which splits completely. This reduction was then generalized to general Galois number fields where q splits in [6]. As an auxiliary result in this paper, we generalize this search-to-decision reduction to the case of arbitrary degree primes q.

The authors of [7] proposed an attack on the decision RLWE problem. The attack makes use of ring homomorphisms $\pi : R \to \mathbb{F}_q$, and works when the image of the RLWE error distribution under the map π only takes values in a small subset of $\mathbb{F}_q$ with overwhelming probability. The authors of [7] then gave an infinite family of examples vulnerable to the attack. Unfortunately, the vulnerable number fields in [7] are not Galois. Hence, the search-to-decision reduction theorem does not apply, and the attack can not be directly used to solve the search variant of RLWE for those instances.

In our paper, we generalize the attack of [7] to Galois number fields and moduli of higher degree. As a result, we have an attack on the *Search*-RLWE problem and an implementation of the attack on concrete RLWE instances, including the search-to-decision reduction. Our attack is new in two major ways: first, the attack considers ring homomorphisms from $R \to \mathbb{F}_{q^f}$, for $f > 1$, instead of just homomorphisms from $R \to \mathbb{F}_q$; second, the error distribution is distinguished from random (i.e. from the uniform distribution) using the statistical chi-squared test, instead of relying on the values of the error polynomial to be small or in a small subset. The attack aims at an intermediate problem used in the search-to-decision proof of [13], which is to recover the secret modulo a prime ideal (denoted SRLWE($\mathcal{R}, \mathfrak{q}$); see Definition 8). The time complexity of our attack is $O(q^{2f})$, where f is the residue degree of q in K.

Importantly, we also show an attack on non-2-power cyclotomic rings, which succeeds with high probability and with surprising efficiency when the modulus is a ramified prime. For example, we show that in dimension $n = 808$, we can attack an RLWE instance in the the cyclotomic ring $\mathbb{Q}(\zeta_{809})$ effectively in 35 seconds, where the modulus is 809. This opens up the question of whether general cyclotomic fields are safe for cryptography, depending on whether modulus switching can be used to transfer this attack from the ramified modulus to other larger moduli which are used in practice.

An important difference between this paper and the attacks of [7] is that here we work directly with the RLWE error distribution, and we do not need to work with a polynomial basis for the ring in order for the attack to work. Thus we also eliminate the need for the assumption that the ring is monogenic, which is

helpful in finding Galois fields which are vulnerable. The attacks of [7] succeed for rings R such that the defining polynomial has special roots modulo q, and we do not need such a restrictive condition on the number field and modulus in order for our new attacks to succeed. We give heuristic arguments about what properties are sufficient for a ring to be vulnerable to our new attacks.

Auxiliary results we present include several stand-alone items of possibly independent interest: We prove a search-to-decision reduction for Galois fields which applies for any unramified modulus q, regardless of the residue degree of q. We consider some heuristic arguments as to whether modulus switching techniques are likely to be successfully combined with our attacks. Also, we analyze the vulnerability of cyclotomic fields to our attack, and show that they are in general safe, except for the case when the modulus p is ramified in the cyclotomic field.

1.1 Organization

In Section 2, we recall definitions related to the RLWE problems. In Section 3, we prove a search-to-decision reduction for Galois extensions K and unramified moduli. In Section 4, we introduce an attack on RLWE problems based on the chi-square statistical test, which directly generalizes the attack in [7]. In Section 5, we give examples of subfields of cyclotomic fields vulnerable to our new attack, where the modulus q has residue degree two. In Section 6, we show that our attack works on prime cyclotomic fields when the modulus is the unique ramified prime. In Section 7, we consider the possibility of modulus switching. Finally, in Section 8, we use Fourier analysis to give a heuristic argument, which we then combine with numerical evidence to support the view that cyclotomic extensions with unramified moduli of small residue degree are invulnerable to our attack.

All computations in this paper were performed in Sage [17]. All the relevant code is available and can be found at `https://github.com/haochenuw/GaloisRLWE`.

2 Background

Let K be a number field of degree n with ring of integers R and let $\sigma_1, \cdots, \sigma_n$ be the embeddings of K into $\mathbb{C}$, the field of complex numbers. The *canonical embedding* of K is

$$\begin{aligned} \iota : K &\to \mathbb{C}^n \\ x &\mapsto (\sigma_1(x), \cdots, \sigma_n(x)). \end{aligned}$$

To work with real vector spaces, we define the *adjusted embedding* of K as follows. Let r_1, r_2 denote the number of real embeddings and conjugate pairs of complex embeddings of K. Without loss of generality, assume $\sigma_1, \cdots, \sigma_{r_1}$ are the real embeddings and $\sigma_{r_1+r_2+j} = \overline{\sigma_{r_1+j}}$ for $1 \leq j \leq r_2$. We define

$$\begin{aligned} \tilde{\iota} : K &\to \mathbb{R}^n \\ x &\mapsto (\sigma_1(x), \cdots, \sigma_{r_1}(x), \mathrm{Re}(\sigma_{r_1+1})(x), \mathrm{Im}(\sigma_{r_1+1})(x), \cdots, \mathrm{Re}(\sigma_{r_1+r_2})(x), \mathrm{Im}(\sigma_{r_1+r_2})(x)). \end{aligned}$$

Then $\Lambda_R = \tilde{\iota}(R)$ is a lattice in $\mathbb{R}^n$, and we call it the *embedded lattice of* R.

Let $w = (w_1, \cdots, w_n)$ be an integral basis for R.

Definition 1. *The canonical (resp. adjusted) embedding matrix of w, denoted by A_w (resp. $\widetilde{A_w}$), is the n-by-n matrix whose i-th column is $\iota(w_i)$ (resp. $\tilde{\iota}(w_i)$).*

The two embedding matrices are related in a simple way: let T denote the unitary matrix

$$T = \begin{bmatrix} I_{r_1} & 0 \\ 0 & T_{r_2} \end{bmatrix}, \text{ where } T_s = \frac{1}{\sqrt{2}} \begin{bmatrix} I_{r_2} & I_{r_2} \\ -iI_{r_2} & iI_{r_2} \end{bmatrix},$$

Then we have

$$\widetilde{A_w} = TA_w,$$

and the lattice Λ_R has a basis consisting of columns of $\widetilde{A_w}$.

For $\sigma > 0$, define the Gaussian function $\rho_\sigma : \mathbb{R}^n \to [0, 1]$ as $\rho_\sigma(x) = e^{-||x||^2/2\sigma^2}$ (our σ is equal to $r/\sqrt{2\pi}$ for the parameter r in [13]).

Definition 2. *For a lattice $\Lambda \subset \mathbb{R}^n$ and $\sigma > 0$, the discrete Gaussian distribution on Λ with parameter σ is:*

$$D_{\Lambda,\sigma}(x) = \frac{\rho_\sigma(x)}{\sum_{y\in\Lambda}\rho_\sigma(y)}, \forall x \in \Lambda.$$

Equivalently, the probability of sampling any lattice point x is proportional to $\rho_\sigma(x)$.

2.1 Ring LWE Problems for General Number Fields

We follow [7] in setting up the Ring LWE problem for general number fields. In particular, we do not consider the dual of the ring of integers.

Definition 3. *An RLWE instance is a tuple $\mathcal{R} = (K, q, \sigma, s)$, where K is a number field with ring of integers R, q is a prime, $\sigma > 0$, and $s \in R/qR$ is the secret.*

Definition 4. *Let $\mathcal{R} = (K, q, \sigma, s)$ be an RLWE instance and let R be the ring of integers of K. The error distribution of $\mathcal{R}$, denote by $D_{\mathcal{R}}$, is the discrete Gaussian distribution*

$$D_{\mathcal{R}} = D_{\Lambda_R,\sigma}.$$

As pointed out in [7], when analyzing the error distribution, one needs to take into account the sparsity of the lattice Λ_R, which is measured by its covolume V_R. In light of this, we define a relative version of the standard deviation parameter:

$$\sigma_0 = \frac{\sigma}{V_R^{\frac{1}{n}}}.$$

The notation $x \leftarrow D$ indicates that variable x is distributed according to distribution D.

Definition 5 (RLWE distribution). *Let $\mathcal{R} = (K, q, \sigma, s)$ be an RLWE instance with error distribution $D_{\mathcal{R}}$. We let R_q denote R/qR, then a sample from the RLWE distribution of $\mathcal{R}$ is a tuple*

$$(a, b = as + e \pmod{qR}) \in R_q \times R_q,$$

where the first coordinate a is chosen uniformly at random in R_q, and $e \leftarrow D_{\mathcal{R}}$.

We use the shorthand notation $(a, b) \leftarrow \mathcal{R}$ to represent that (a, b) is sampled from the RLWE distribution of $\mathcal{R}$.

The RLWE problem has two major variants: search and decision.

Definition 6 (Search RLWE). *Let $\mathcal{R}$ be an RLWE instance. The Search Ring-LWE problem, denoted by* SRLWE($\mathcal{R}$)*, is to discover s given access to arbitrarily many independent samples $(a, b) \leftarrow \mathcal{R}$.*

Definition 7 (Decision RLWE). *Let $\mathcal{R}$ be an RLWE instance. The Decision Ring-LWE problem, denoted by* DRLWE($\mathcal{R}$)*, is to distinguish between the same number of independent samples in two distributions on $R_q \times R_q$. The first is the RLWE distribution of $\mathcal{R}$, and the second consists of uniformly random and independent samples from $R_q \times R_q$.*

2.2 Sampling Methods

In practice, there are different ways to approximately sample from the RLWE error distribution $D_{\mathcal{R}}$, and we will consider three sampling methods in our paper. While searching for weak Galois RLWE instances as well as attacking ramified primes, we use the sampling algorithm in [9]; when analyzing the security of cyclotomics, we use the PLWE distribution $P_{m,\tau}$ and another distribution $P'_{m,k}$ to assist the analysis. The efficient sampling algorithm in [14] for cyclotomic fields is related to the dual version of RLWE, so we will not use it in our paper.

3 Search-to-Decision Reduction

In [6], the search-to-decision reduction of [13] is extended to Ring-LWE for Galois number fields, where q is an unramified prime of degree one. The approach is via an intermediate problem, denoted $\mathfrak{q}_i$-LWE in [13]. In this section, we extend this result to primes $\mathfrak{q}$ of arbitrary residue degree. Our intermediate problem, which we denote by SRLWE$(\mathcal{R}, \mathfrak{q})$, is the same as $\mathfrak{q}_i$-LWE, and it amounts to find the secret modulo the prime $\mathfrak{q}$. The Galois group allows us to bootstrap this piece of information to discover the full secret.

The attack in Section 4 targets SRLWE$(\mathcal{R}, \mathfrak{q})$ and hence, by the results of this section, will solve Search Ring-LWE. In Section 5, we demonstrate the attack on Search Ring-LWE in practice.

Definition 8. *Let $\mathcal{R} = (K, q, \sigma, s)$ be an RLWE instance and let $\mathfrak{q}$ be a prime of K lying above q. The problem* SRLWE$(\mathcal{R}, \mathfrak{q})$ *is to determine $s \pmod{\mathfrak{q}}$, given access to arbitrarily many independent samples $(a, b) \leftarrow \mathcal{R}$.*

We recall some facts from algebraic number theory in the following lemma.

Lemma 9. *Let $K/\mathbb{Q}$ be a finite Galois extension with ring of integers R, and let q be a prime unramified in K. Then there exists a unique divisor g of n and a set of g distinct prime ideals $\mathfrak{q}_1, \cdots, \mathfrak{q}_g$ of R such that:*

1. *$qR = \prod_{i=1}^{g} \mathfrak{q}_i$,*
2. *the quotient $R/\mathfrak{q}_i$ is a finite field of cardinality q^f for each i, where $f = \frac{n}{g}$,*
3. *there is a canonical isomorphism of rings*

$$R_q \cong R/\mathfrak{q}_1 \times \cdots \times R/\mathfrak{q}_g, \tag{1}$$

4. *the Galois group acts transitively on the ideals $\mathfrak{q}_1, \ldots, \mathfrak{q}_g$ and this action descends to an action on R_q which permutes the corresponding factors in (1) in the same way.*

The number f in the above lemma is called the *residue degree* of q in K. Note that the prime q splits completely in K if and only if its residue degree is one.

Theorem 10. *Let $\mathcal{R} = (K, q, \sigma, s)$ be an RLWE instance with $K/\mathbb{Q}$ Galois of degree n and q unramified in K with residue degree f. Let $\mathscr{A}$ be an oracle which solves* SRLWE$(\mathcal{R}, \mathfrak{q})$ *using a list of m samples modulo $\mathfrak{q}$. Let S be a set of m RLWE samples in $R_q \times R_q$. Then the problem* SRLWE$(\mathcal{R})$ *can be solved using S by n/f calls to the oracle $\mathscr{A}$, $2mn/f$ reductions $R_q \to R/\mathfrak{q}$, and $2mn/f$ evaluations of a Galois automorphism on R_q.*

Proof. The Galois group $G = \mathrm{Gal}(K/\mathbb{Q})$ acts on the set $\{\mathfrak{q}_1, \cdots, \mathfrak{q}_g\}$ transitively. Hence for each i, there exists $\sigma_i \in \mathrm{Gal}(K/\mathbb{Q})$, such that $\sigma_i(\mathfrak{q}) = \mathfrak{q}_i$, Then we call the oracle $\mathscr{A}$ on the input $(\sigma_i^{-1}(S) \pmod{\mathfrak{q}}, \mathfrak{q})$. The algorithm will output $\sigma_i^{-1}(s) \pmod{\mathfrak{q}}$, from which we can recover $s \pmod{\mathfrak{q}_i}$ using σ_i. We do this for all $1 \leq i \leq g$ and use (1) of Lemma 9 to recover s. □

In particular, if the number of samples m is polynomial in n and the time taken to evaluate Galois automorphisms on a single sample is also polynomial in n, then Theorem 10 gives a polynomial time reduction from SRLWE$(\mathcal{R})$ to SRLWE$(\mathcal{R}, \mathfrak{q})$.

Remark 11. For a proper runtime analysis of the reduction, one must examine the implementation, in particular with regards to Galois automorphisms. The runtime for evaluating an automorphism depends rather strongly on the instance and on the way ring elements are represented. For example, for subfields of cyclotomic fields represented with respect to normal integral bases, the Galois automorphisms are simply permutations of the coordinates, so the time needed to apply these automorphisms is trivial.

The search-to-decision reduction will follow from the lemma below.

Lemma 12. *There is a probabilistic polynomial time reduction from* SRLWE$(\mathcal{R}, \mathfrak{q})$ *to* DRLWE$(\mathcal{R})$.

Proof. This is a rephrasing of [13, Lemma 5.9 and Lemma 5.12]. □

Corollary 13. *Suppose $\mathcal{R}$ is an RLWE instance where K is Galois and q is an unramified prime in K. Then there is a probabilistic polynomial-time reduction from* SRLWE$(\mathcal{R})$ *to* DRLWE$(\mathcal{R})$.

4 The Chi-square Attack

In this section, we extend the $f(1) \equiv 0 \pmod{q}$ attack of [6] and the root-of-small-order attack of [7]. These attacks can be viewed as examples of a more general attack principle, as follows. Suppose one has a ring homomorphism

$$\phi : R_q \to F$$

where F is a finite field, and where two properties hold:

1. F is small enough that its elements can be examined exhaustively; and
2. the error distribution on R_q, transported by ϕ to F, is detectably non-uniform.

Then the attack on DRLWE on R_q is as follows:

1. Transport the samples (a, b) in $R_q \times R_q$ to $F \times F$ via ϕ.
2. Loop through possible guesses for the image of the secret, $\phi(s)$, in F.
3. For each guess g, compute the distribution of $\phi(b) - \phi(a)g$ on the available samples (this is $\phi(e)$ if the guess is correct).
4. If the samples are RLWE samples with secret s and $g = \phi(s)$, then this distribution will follow the error distribution, which will look non-uniform.
5. If all such distributions look uniform, then the samples were uniform, not RLWE, samples.

The fact that ϕ is a ring homomorphism is essential in guaranteeing that for the correct guess, the distribution in question is the image of the error distribution. The only ring homomorphisms from R_q to a finite field are given by reduction modulo a prime ideal $\mathfrak{q}$ lying above q in R.

4.1 Chi-square Test for Uniform Distribution

We briefly review the properties and usage of the chi-square test for uniform distributions over a finite set S. We partition S into r subsets $S = \bigsqcup_{j=1}^{r} S_j$, called *bins*. Suppose there are M samples $y_1, \ldots, y_M \in S$. For each $1 \le j \le r$, we compute the expected number of samples in the j-th subset: $c_j := \frac{|S_j|M}{|S|}$. Then we compute the actual number of samples in S_j, i.e., $t_j := |\{1 \le i \le r : y_i \in S_j\}|$. Finally, the χ^2 value is computed as

$$\chi^2(S, y) = \sum_{j=1}^{r} \frac{(t_j - c_j)^2}{c_j}.$$

Suppose the samples are drawn from the uniform distribution on S. Then the χ^2 value follows the chi-square distribution with $(r-1)$ degrees of freedom, which we denote by χ^2_{r-1}. Let $\mathcal{F}_{r-1}(x)$ denote its cumulative distribution function. For the chi-square test, we choose a confidence level parameter $\alpha \in (0, 1)$ and compute $\delta = \mathcal{F}^{-1}_{r-1}(\alpha)$. Then we reject the hypothesis that the samples are drawn from the uniform distribution if $\chi^2(S, y) > \delta$.

If P, Q are two probability distributions on the set S, then their *statistical distance* is defined as

$$d(P, Q) = \frac{1}{2} \sum_{t \in S} |P(t) - Q(t)|.$$

For convenience, we also define the l_2 *distance* between P and Q as $d_2(P, Q) = (\sum_{t \in S} |P(t) - Q(t)|^2)^{\frac{1}{2}}$. We have the inequality $d(P, Q) \le \frac{\sqrt{|S|}}{2} d_2(P, Q)$.

4.2 The Chi-square Attack on SRLWE($\mathcal{R}$, $\mathfrak{q}$)

Let $\mathcal{R}$ be an RLWE instance with error distribution $D_{\mathcal{R}}$ and $\mathfrak{q}$ be a prime ideal above q. The basic idea of our attack relies on the assumption that the distribution $D_{\mathcal{R}} \pmod{\mathfrak{q}}$ is distinguishable from the uniform distribution on the finite field $F = R/\mathfrak{q}$. More precisely, the attack loops through all q^f possible values $\bar{s} = s \pmod{\mathfrak{q}}$, and for each guess s', it computes the values $\bar{e}' = \bar{b} - \bar{a}s' \pmod{\mathfrak{q}}$ for every sample $(a, b) \in S$. If

the guess is wrong, or if the samples are taken from the uniform distribution in $(R_q)^2$, the values $\bar{e}'$ would be uniformly distributed in F and it is likely to pass the chi-square test. On the other hand, if the guess is correct, then we expect the test on the errors $\bar{e}'$ to reject the null hypothesis. Let $N = q^f$ denote the cardinality of F. We remark that one needs at least $\Omega(N)$ samples for the test to work effectively.

For the attack to be successful, we need the $(N-1)$ tests corresponding to wrong guesses of $s \pmod{\mathfrak{q}}$ to pass, and the one test corresponding to the correct guess to be rejected. For this purpose, we need to choose the confidence level α to be close enough to one (a reasonable choice is $\alpha = 1 - \frac{1}{10N}$). The detailed attack is described in Algorithm 1. Let $\mathcal{F}_{N-1}(x)$ denote the cumulative distribution function of χ^2_{N-1}.

Algorithm 1 chi-square attack of $SRLWE(\mathcal{R}, \mathfrak{q})$

Input: $\mathcal{R} = (K, q, \sigma, s)$ – an RLWE instance; R – the ring of integers of K; $\mathfrak{q}$ – a prime ideal in K above q; $F = R/\mathfrak{q}$ – the residue field of $\mathfrak{q}$; N – the cardinality of F; $\mathcal{S}$ – a collection of M ($M = \Omega(N)$) RLWE samples from $\mathcal{R}$; $0 < \alpha < 1$ – the confidence level.

Output: a guess of the value $s \pmod{\mathfrak{q}}$, or **NOT-RLWE**, or **INSUFFICIENT-SAMPLES**

```
δ ← F_{N-1}^{-1}(α), G ← ∅.
for s in F do
    E ← ∅.
    for a, b in S do
        ā, b̄ ← a (mod q), b (mod q).
        ē ← b̄ − ās.
        add ē to E.
    end for
    χ²(E) ← Σ_{j=1}^{N} (|{c∈E : c=j}| − M/N)² / (M/N).
    if χ²(E) > δ then
        add s to G.
    end if
end for
if G = ∅ then
    return NOT-RLWE
else if G = {g} then
    return g
else
    return INSUFFICIENT-SAMPLES
end if
```

Remark 14. For simplicity of exposition, we use N bins in Algorithm 1, that is one element per bin. In some situations, it might be advantageous to choose the bins differently.

The time complexity of the attack is $O(N^2)$ since there are N possible values for $s \pmod{\mathfrak{q}}$ and the number of samples needed is $O(N)$. The correctness of the attack is captured in Theorem 15 below. We use $D_{\mathcal{R},\mathfrak{q}}$ as a shorthand notation for $D_{\mathcal{R}} \pmod{\mathfrak{q}}$. For $\lambda \in \mathbb{R}$ and $d \in \mathbb{Z}$, we use $\mathcal{F}_{d,\lambda}(x)$ to denote the cumulative distribution function of the noncentral chi-square distribution with degree of freedom d and parameter λ.

Theorem 15. *Let $\mathcal{R} = (K, q, s, \sigma)$ be an RLWE instance. Suppose $\mathfrak{q}$ be a prime ideal in K above q, and let Δ denote the statistical distance between the distribution $D_{\mathcal{R},\mathfrak{q}}$ and the uniform distribution on $R/\mathfrak{q}$. Let M be the number of samples used in Algorithm 1, and let $\lambda = 4M\Delta^2$. Let $0 < \alpha < 1$ and let $\delta = \mathcal{F}^{-1}_{N-1}(\alpha)$. If p is the probability of success of the attack in Algorithm 1, then*

$$p \geq \alpha^{N-1}(1 - \mathcal{F}_{N-1;\lambda}(\delta)).$$

Proof. It is a standard fact (see [15], for example) that the chi-square value on samples from $D_{\mathcal{R},\mathfrak{q}}$ follows the noncentral chi-square distribution with $(N-1)$ degrees of freedom and parameter λ_0 given by

$$\lambda_0 = d_2(D_{\mathcal{R},\mathfrak{q}}, U(R/\mathfrak{q}))^2 \cdot MN.$$

Note that we have $\lambda_0 \geq (2d(D_{\mathcal{R},\mathfrak{q}}, U(R/\mathfrak{q}))/\sqrt{N})^2 MN = 4M\Delta^2 = \lambda$. Recall that our attack succeeds if the "error" set $\mathcal{E}$ from each of the $(N-1)$ wrong guesses of $s \pmod{\mathfrak{q}}$ passes the test, and the true reduced errors fails the test. We assume that the results of these tests are independent of each other. Then the first event happens with probability α^{N-1}, whereas the second event has probability $(1 - \mathcal{F}_{N-1;\lambda_0}(\delta))$. Since this is an increasing function in λ_0, we replace λ_0 by λ and the theorem follows. □

Remark 16. One could choose the value of α in Theorem 15 to suit the specific instance. The probability of success will change accordingly. When we expect the statistical distance Δ to be large, it is preferable to choose a larger α to increase the probability of success. For example, if we choose $\alpha = 1 - \frac{1}{10N}$, then $\alpha^{N-1} \geq e^{-1/10} = 0.904\cdots$.

Figure 1 shows a plot of p versus Δ for various choices of N, made according to Theorem 15, where we fix the number of samples to be $M = 5N$ and fix $\alpha = 1 - \frac{1}{10N}$.

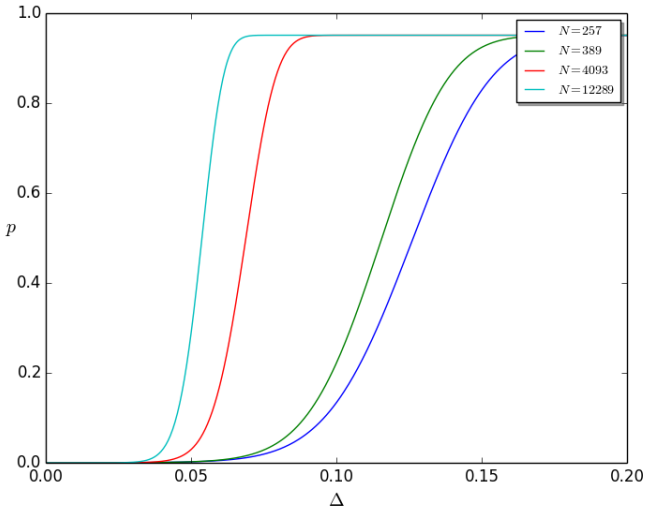

Fig. 1. Success probability versus statistical distance

5 Vulnerable Instances among Subfields of Cyclotomic Fields

We searched for instances of RLWE vulnerable to the chi-square attack. For this purpose, we restricted attention to subfields of cyclotomic fields $\mathbb{Q}(\zeta_m)$, where we assume m is *odd and squarefree*. The Galois group $\mathrm{Gal}(\mathbb{Q}(\zeta_m)/\mathbb{Q})$ is canonically isomorphic to $G = (\mathbb{Z}/m\mathbb{Z})^*$. For each subgroup H of G, let $K_{m,H} = \mathbb{Q}(\zeta_m)^H$ be the subfield of elements fixed by H. Then the extension $K_{m,H}/\mathbb{Q}$ is Galois with degree $n = \frac{\varphi(m)}{|H|}$. Also, the residue degree of a prime q in $K_{m,H}$ is equal to the order of $[q]$ in the quotient group G/H. Moreover, $K_{m,H}$ has canonical *normal integral basis*, whose embedding matrix is easy to compute. More precisely, let C denote a set of coset representatives of the coset space G/H. If c is an integer coprime to m, we use $[c]$ to denote its coset in G/H. For each $[c] \in C$, set

$$w_{[c]} = \sum_{h \in H} \zeta_m^{hc}.$$

Then $w := (w_{[c]})_{[c]\in C}$ is a $\mathbb{Z}$-basis of R. (For a proof of this fact, see [10, Proposition 6.1]). Setting $\zeta = \exp(2\pi i/m)$, the canonical embedding matrix of w is

$$(A_w)_{[i],[j]} = \sum_{h \in H} \zeta^{hij}, \text{ for } [i], [j] \in C.$$

Lemma 17. *Suppose $\mathcal{R}$ is an RLWE instance such that the underlying field K is a Galois number field and q is unramified in K. Then the reduced error distribution $D_{\mathcal{R},\mathfrak{q}}$ (that is, $D_{\mathcal{R}}(\pmod{\mathfrak{q}})$) is independent of the choice of prime ideal $\mathfrak{q}$ above q.*

Proof. From Lemma 9, we may switch from a prime $\mathfrak{q}$ to $\mathfrak{q}'$ via $\mathrm{Gal}(K/\mathbb{Q})$. On the other hand, the Galois group acts on the embedded lattice Λ_R by permuting the coordinates. Hence we have a group homomorphism

$$\phi : \mathrm{Gal}(K/\mathbb{Q}) \to \mathrm{Aut}(\Lambda).$$

Since permutation matrices are orthogonal, the Galois group action on Λ_R given by ϕ is distance-preserving. In particular, it preserves any spherical discrete Gaussian distribution on Λ_R. □

5.1 Searching

Algorithm 1 allows us to search for vulnerable instances among fields of the form $K_{m,H}$ by generating actual RLWE samples and running the attack. Success of the attack will indicate vulnerability of the instance. Note that our field searching requires sampling efficiently from a discrete Gaussian $D_{\Lambda,\sigma}$, for which we use the efficient algorithm of [9].

In Table 1, we list some instances on which the attack has succeeded. The columns of Table 1 are as follows. The first two columns specify m and the generators of H, where H is represented as a subgroup of $(\mathbb{Z}/m\mathbb{Z})^*$; the column labeled f is the residue degree of q. The last column consists of either the runtime for an actual attack which succeeded, or an estimation of the runtime. Note that we omitted our choice of prime ideal $\mathfrak{q}$, since due to Lemma 17 the choice of $\mathfrak{q}$ is irrelevant to our attack. The parameters σ_0 in Table 1 represent the boundary of the power of our attack, i.e., we tried higher σ_0 and the attack failed. Note that although σ_0 is relatively small, in practice it still provides exponentially many error vectors. Intuitively, when $\sigma_0 = 1$, our σ is equal to the geometric mean of the lengths of a Gram-Schmidt basis of Λ_R. In practice, the lengths of these basis vectors do not differ by a lot, so we still expect to get at least $\Omega(2^n)$ error vectors.

The rows of Table 1 with "estimated" runtime mean the following. First, we ran the chi-square test on the correct reduced errors to obtain an estimate $\hat{\Delta}$ of the statistical distance Δ. We then chose α according to $\hat{\Delta}$ and obtained an estimation $\hat{p}$ of the success probability of our attack, using the formula in Theorem 15. The corresponding rows in the table all have $\hat{p} > 1 - 2^{-10}$, suggesting that the attack is very likely to succeed. Finally, we ran a few chi-square tests on samples obtained from a few randomly chosen incorrect guesses to compute the average time t for running one chi-square test. We set the estimated runtime for the attack to be tN.

Table 1. Attacked sub-cyclotomic RLWE instances

m	generators of H	n	q	f	σ_0	no. samples	runtime (in hours)
2805	[1684, 1618]	40	67	2	1	22445	3.49
15015	[12286, 2003, 11936]	60	43	2	1	11094	1.05
15015	[12286, 2003, 11936]	60	617	2	1.25	8000	228.41 (estimated)
90321	[90320, 18514, 43405]	80	67	2	1	26934	4.81
255255	[97943, 162436, 253826, 248711, 44318]	90	2003	2	1.25	15000	1114.44 (estimated)
285285	[181156, 210926, 87361]	96	521	2	1.1	5000	75.41 (estimated)
1468005	[312016, 978671, 956572, 400366]	100	683	2	1.1	5000	276.01 (estimated)
1468005	[198892, 978671, 431521, 1083139]	144	139	2	1	4000	5.72

5.2 Discussion

We searched for vulnerable instances where the modulus has residue degree one or two. It turns out that all vulnerable instances we found and listed in Table 1 have a modulus of degree two. We have a heuristic

explanation for the existence of examples of higher degree. Let K be a Galois number field and suppose q is a prime of degree f in K. Suppose we have found a short basis $w_1, \cdots, w_n$ of R with respect to the adjusted embedding. Fix a prime ideal $\mathfrak{q}$ above q. Then the images of the basis under the reduction modulo $\mathfrak{q}$ map are elements of $F = R/\mathfrak{q}$. Now if for some index i, the element w_i lies inside some proper subfield K' of K, and if q has residue degree $f' < f$ in K', then $w_i \pmod{\mathfrak{q}}$ will lie in a proper subfield of F. If this occurs for a large number of the basis elements w_i, then we could expect the reduced error distribution $D_{\mathcal{R},\mathfrak{q}}$ to take values in a proper subfield of F more frequently. This would allow us to distinguish it from the uniform distribution on F.

In practice, we found out that the above scenario is more likely to happen when the field K has a subfield K' of index 2 such that q splits completely in K', while q has degree 2 in K. Since the ring of integers of K' is a subring of the ring of integers of K, one has at least $n/2$ vectors in Λ_R with the desired property, i.e., their reduction modulo some prime $\mathfrak{q}$ above q lie inside $\mathbb{F}_q$ instead of $\mathbb{F}_{q^2}$.

5.3 A Detailed Example

In order to illustrate our discussion above together with the search-to-decision reduction, we present a vulnerable Galois RLWE instance in detail, where we generated RLWE samples, performed the attack, and used the search-to-decision reduction to recover the entire secret s.

Let $m = 3003$ and H be the subgroup of $(\mathbb{Z}/m\mathbb{Z})^*$ generated by 2276, 2729 and 1123. Then $K = K_{m,H}$ is a Galois number field of degree $n = 30$. We take the modulus to be $q = 131$, a prime of degree two in K, and take $\sigma_0 = 1$. We generate the secret s from the discrete Gaussian $D_{\Lambda_R,\sigma}$. There are 15 prime ideals in K lying above q, which we denote by $\mathfrak{q}_1, \cdots, \mathfrak{q}_{15}$. We then generate 1000 RLWE samples and use Algorithm 1 and Theorem 10 to recover $s \pmod{\mathfrak{q}_i}$ for each $1 \leq j \leq 15$. Then we use Chinese remainder theorem to recover s. The attack succeeded in 32.8 hours. The code for this attack is in Section 9.5.

6 Attacking Prime Cyclotomic Fields when the Modulus is the Ramified Prime

Let p be an odd prime and let $K = \mathbb{Q}(\zeta_p)$ be the p-th cyclotomic field. Then K has degree $(p-1)$ and discriminant p^{p-2}. In addition, the prime p is totally ramified in K. There is a unique prime ideal $\mathfrak{p} = (1-\zeta_p)$ above p, and the reduction map $\pi : R/pR \to \mathbb{F}_p$ satisfies

$$\pi(\zeta_p^i) = 1, \quad \forall i \in \mathbb{Z}.$$

Writing an RLWE error as $e = \sum e_i \zeta_m^i$, we have $e \pmod{\mathfrak{p}} = \sum_i e_i$. Since the coefficients e_i tend to be small, it may be that $e \pmod{\mathfrak{p}}$ takes on small values with higher probability, making the instance vulnerable to our chi-square attack. Table 2 contains data of some actual attacks we have done. Note that the parameters σ_0 represent the boundary of the power of our attack, i.e., we tried higher σ_0 and the attack failed.

Table 2. Attacked instances of DRLWE for $K = \mathbb{Q}(\zeta_p)$

$q\ (= p)$	n	σ_0	runtime (in seconds)
251	250	0.5	2.62
503	503	0.575	12.02
809	808	0.61	34.38

7 Can Modulus Switching be Used?

The modulus switching procedure is a technique to reduce noise in RLWE samples, and has been discussed extensively in [2] and [11]. We recap the basic ideas of modulus switching. Let $\mathcal{R} = (K, q, \sigma, s)$ be an RLWE

instance. Choose another prime p less than q as the new modulus and consider the instance $\mathcal{R}' = (K, p, \sigma', s)$ for some $\sigma' > \sigma$. We can "switch modulus" if there exists a map

$$\pi_{q,p} : R_q \to R_p,$$

which takes RLWE samples with respect to $\mathcal{R}$ to RLWE samples with respect to $\mathcal{R}'$. In what follows, we give a heuristic argument that our attack will not work in combination with modulus switching under a naïve implementation, and isolate the key characteristics a successful implementation of the attack would require.

One example of a map $\pi_{q,p}$ being used in practice is as follows. Let $\alpha = \frac{p}{q}$ and fix a small positive number τ. For an equivalence class $[a]$ in R_q, we sample a vector a' from the "shifted discrete Gaussian" $D_{\Lambda_R,\tau,\alpha a}$, defined as follows. For a lattice Λ and a vector $c \in \mathbb{R}^n$,

$$D_{\Lambda,\tau,c}(x) = \frac{\rho_\tau(x-c)}{\sum_{y\in\Lambda}\rho_\tau(y-c)}, \forall x \in \Lambda.$$

Finally, we set $\pi_{q,p}([a]) = a' \pmod{pR}$. Note that the definition of $\pi_{q,p}([a])$ is independent of the choice of representative a, as follows. Suppose we choose another representative a_1, then $a_1 = a + \lambda q$ for some $\lambda \in R$, hence $\alpha a_1 = \alpha a + \lambda p$. Finally, observe that the shifted discrete Gaussian behaves well under translating by a lattice point, i.e., we have $D_{\Lambda,\tau,c+u} = D_{\Lambda,\tau,c} + u$ for any $u \in \Lambda$.

Put loosely, the map $\pi_{q,p}$ scales a by p/q and then rounds back into the lattice. It is a natural question then to ask whether modulus switching can be combined with our attack, to switch from a "strong" modulus to a "weak" modulus. However, a heuristic argument shows that the naive combination of our attack with modulus switching will not work.

Let $a'' = \alpha a - a'$. By construction, we expect a'' to be a short vector in $\mathbb{R}^n$, and the point a' can be viewed as a "rounding" of the point αa to the lattice Λ_R.

We will make two heuristic assumptions:

1. That $\pi_{q,p}$ takes the uniform distribution on R_q to an almost uniform distribution on R_p.
2. The distribution of a'' and $(sa)''$ is independent modulo $\mathfrak{q}$, for $s \neq \pm 1$.

Proposition 18. *Under the assumption that $\pi_{q,p}$ takes the uniform distribution on R_q to an almost uniform distribution on R_p, the reduction of a'' modulo $\mathfrak{p}$ will be almost uniformly distributed in $R/\mathfrak{p}R$.*

Proof. The reduction map $R \to R/\mathfrak{p}$ is a ring homomorphism that can be extended to a homomorphism of additive groups $\phi : \frac{1}{q}R \to R/\mathfrak{p}$ by the following chain of maps:

$$\frac{1}{q}R \xrightarrow{(\mathrm{mod}\ \mathfrak{p}\frac{1}{q}R)} \frac{1}{q}R\Big/\mathfrak{p}\frac{1}{q}R \xrightarrow{\times q} R/\mathfrak{p}R \xrightarrow{\times[q]^{-1}} R/\mathfrak{p}R.$$

Then the relation $a''+a' = \alpha a$ is preserved by this map. However, $\phi(\alpha a) = 0 \pmod{\mathfrak{p}}$, so that $\phi(a'') \equiv -\phi(a')$. □

Suppose we have a sample $(a, b) \leftarrow \mathcal{R}$ and the switched sample $(a', b') = (\pi_{q,p}(a), \pi_{q,p}(b))$. Consider the error $e' := b' - a's$. Suppose $b = as + e + \lambda q$ for some $\lambda \in R$. Then

$$\begin{aligned} e' &= b' - a's \\ &= \alpha(b - as) - b'' + a''s. \\ &= \alpha e + \lambda p - b'' + a''s. \end{aligned}$$

and therefore, considering this as an additive relation in $\frac{1}{q}R$ and applying the map of the proof above,

$$e' \equiv -b'' + a''s \pmod{\mathfrak{p}}.$$

By the Proposition above, a'' and b'' are uniformly distributed modulo $\mathfrak{p}$. Hence, if we assume the a'' and b'' are independent, then the reduced rounding errors $a'' \pmod{\mathfrak{p}}$ and $b'' \pmod{\mathfrak{p}}$ are also independent, and the new reduced errors $e' \pmod{\mathfrak{p}}$ would follow the uniform distribution. So our chi-square attack will fail on these modulus-switched samples, even though p might be a "weak" modulus.

Therefore, the best hope of attack is if one of our two assumptions is violated by a map $\pi_{q,p}$. The second is the most likely target. Note that a'' and b'' are the rounding errors when we try to round αa and αb to the lattice Λ_R. However, Λ_R is a n-dimensional lattice, so there are $\Omega(2^n)$ options of rounding a vector in $\mathbb{R}^n$ to a moderately close lattice point. Even in the scenario with zero error, i.e., $e = 0$, an attacker will face the task of finding a "nice" rounding algorithm, so that the roundings of the two vectors αa and $\alpha b = \alpha as$ are somehow related.

So far, we are not aware of any such algorithm, unless the secret s is trivial, e.g., $s = 1$, in which case αa is almost equal to αb, and one expects that a'' is close to b''.

8 Invulnerability of General Cyclotomic Extensions for Unramified Primes

In this section we provide some numerical evidence that for cyclotomic fields, the image of the RLWE error distribution modulo an unramified prime ideal $\mathfrak{q}$ of residue degree one or two is practically indistinguishable from uniform, implying that the cyclotomics are protected against the family of attacks in this paper. For simplicity of analysis, we will define two error distributions that approximate the RLWE error distribution (the PLWE error distribution and the modified PLWE error distribution). The advantage of these simpler distributions is the relative accessibility of a formula for a bound on the statistical distance between these distributions and the uniform distribution. This eases computation and allows for heuristic arguments. Then, we generate the actual RLWE samples, run our chi-square attack, and confirm that the errors modulo $\mathfrak{q}$ are indeed uniform.

Let $m \geq 1$ be an integer and let $K = \mathbb{Q}(\zeta_m)$ be the m-th cyclotomic field. Let q be a prime such that $q \equiv 1 \pmod{m}$, so q is unramified in K. Finally, let $\mathfrak{q}$ be a prime ideal above q.

First, we introduce the PLWE error distribution on cyclotomic fields, which is commonly used in practice for homomorphic encryption schemes as a substitute for the RLWE error distribution. Let $n = \varphi(m)$ be the degree of K.

Definition 19. *Let $\tau > 0$. A sample from the PLWE distribution $P_{m,\tau}$ is*

$$e = \sum_{i=0}^{n-1} e_i \zeta_m^i,$$

where the e_i are sampled independently from the discrete Gaussian $D_{\mathbb{Z},\tau}$.

Next, with the aim of simplifying our analysis, we introduce a class of "shifted binomial distributions" indexed by even integers $k \geq 2$, which approximate discrete Gaussians over $\mathbb{Z}$.

Definition 20. *For an even integer $k \geq 2$, let $\mathcal{V}_k$ denote the distribution over $\mathbb{Z}$ such that for every $t \in \mathbb{Z}$,*

$$\text{Prob}(\mathcal{V}_k = t) = \begin{cases} \frac{1}{2^k}\binom{k}{t+\frac{k}{2}} & \text{if } |t| \leq \frac{k}{2} \\ 0 & \text{otherwise} \end{cases}$$

We will abuse notation and also use $\mathcal{V}_k$ to denote the reduced distribution $\mathcal{V}_k \pmod{q}$ over $\mathbb{F}_q$, and let ν_k denote its probability density function. Figure 2 shows a plot of the function v_8.

Definition 21. *Let $k \geq 2$ be an even integer. Then a sample from the modified PLWE error distribution $P'_{m,k}$ is*

$$e' = \sum_{i=0}^{n-1} e'_i \zeta_m^i,$$

where the coefficients e'_i are sampled independently from $\mathcal{V}_k$.

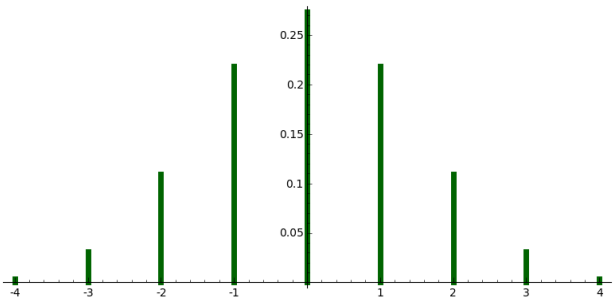

Fig. 2. Probability density function of $\mathcal{V}_8$

8.1 Bounding the Distance from Uniform

We recall the definition and key properties of Fourier transform over finite fields. Suppose f is a real-valued function on $\mathbb{F}_q$. The *Fourier transform* of f is defined as

$$\widehat{f}(y) = \sum_{a \in \mathbb{F}_q} f(a)\overline{\chi_y}(a),$$

where $\chi_y(a) := e^{2\pi i a y/q}$.

Let u denote the probability density function of the uniform distribution over $\mathbb{F}_q$, that is $u(a) = \frac{1}{q}$ for all $a \in \mathbb{F}_q$. Let δ denote the characteristic function of the one-point set $\{0\} \subseteq \mathbb{F}_q$. Recall that the convolution of two functions $f, g : \mathbb{F}_q \to \mathbb{R}$ is defined as $(f * g)(a) = \sum_{b \in \mathbb{F}_q} f(a-b)g(b)$. We list without proof some basic properties of the Fourier transform.

1. $\widehat{\delta} = qu$; $\widehat{u} = \delta$.
2. $\widehat{f * g} = \widehat{f} \cdot \widehat{g}$.
3. $f(a) = \frac{1}{q} \sum_{y \in \mathbb{F}_q} \widehat{f}(y)\chi_y(a)$ (the Fourier inversion formula).

The following is a standard result.

Lemma 22. *Suppose the random variables F, G are independent random variables with values in $\mathbb{F}_q$, having probability density functions f and g. Then $h = f * g$. In general, suppose $F_1, \cdots, F_n$ are mutually independent random variables in $\mathbb{F}_q$, with probability density functions $f_1, \cdots, f_n$. Let f denote the density function of the sum $F = \sum F_i$, then $f = f_1 * \cdots * f_n$.*

The Fourier transform of ν_k has a nice closed-form formula, as below.

Lemma 23. *For all even integers $k \geq 2$, $\widehat{\nu_k}(y) = \cos\left(\frac{\pi y}{q}\right)^k$.*

Proof. We have

$$\begin{aligned}
2^k \cdot \widehat{\nu_k}(y) &= \sum_{m=-\frac{k}{2}}^{\frac{k}{2}} \binom{k}{m+\frac{k}{2}} e^{2\pi i y m/q} \\
&= e^{-\pi i y k/q} \sum_{m=-\frac{k}{2}}^{\frac{k}{2}} \binom{k}{m+\frac{k}{2}} e^{2\pi i y (m+k/2)/q} \\
&= e^{-\pi i y k/q} \sum_{m'=0}^{k} \binom{k}{m'} e^{2\pi i y m'/q} \\
&= e^{-\pi i y k/q} (1 + e^{2\pi i y/q})^k \\
&= (e^{-\pi i y/q} + e^{\pi i y/q})^k \\
&= (2\cos(\pi y/q))^k.
\end{aligned}$$

Dividing both sides by 2^k gives the result. □

Next, we concentrate on the reduced distributions $P_{m,\tau} \pmod{\mathfrak{q}}$ and $P'_{m,k} \pmod{\mathfrak{q}}$. Note that there is a one-to-one correspondence between primitive m-th roots of unity in $\mathbb{F}_q$ and the prime ideals above q in $\mathbb{Q}(\zeta_m)$. Let α be the root corresponding to our choice of $\mathfrak{q}$. Then a sample from $P_{m,\tau} \pmod{\mathfrak{q}}$ (resp. $P'_{m,k} \pmod{\mathfrak{q}}$) is of the form

$$\sum_{i=0}^{n-1} \alpha^i e_i \pmod{q},$$

where e_i are independent variables under the distribution $D_{\mathbb{Z},\tau}$ (resp. $\mathcal{V}_k$). We use e_α and e'_α to denote their probability density functions. Then

Lemma 24.

$$\widehat{e'_\alpha}(y) = \prod_{i=1}^{n} \cos\left(\frac{\alpha^i \pi y}{q}\right)^k.$$

Proof. This follows directly from Lemma 23 and the basic properties of Fourier transform. □

Now we are able to bound the difference using the Fourier inversion formula.

Proposition 25. *Let $f : \mathbb{F}_q \to \mathbb{R}$ be a function such that $\sum_{a\in\mathbb{F}_q} f(a) = 1$. Then for all $a \in \mathbb{F}_q$,*

$$|f(a) - 1/q| \le \frac{1}{q} \sum_{y\in\mathbb{F}_q, y\neq 0} |\hat{f}(y)|. \tag{2}$$

Proof. For all $a \in \mathbb{F}_q$,

$$\begin{aligned} f(a) - 1/q &= f - u(a) \\ &= \frac{1}{q} \sum_{y\in\mathbb{F}_q} (\hat{f}(y) - \widehat{u}(y))\chi_y(a) \\ &= \frac{1}{q} \sum_{y\in\mathbb{F}_q} (\hat{f}(y) - \delta(y))\chi_y(a) \\ &= \frac{1}{q} \sum_{y\in\mathbb{F}_q, y\neq 0} \hat{f}(y)\chi_y(a). \qquad (\text{since } \hat{f}(0) = 1) \end{aligned}$$

Now the result follows from taking absolute values on both sides, and noting that $|\chi_y(a)| \le 1$ for all a and all y. □

Taking $f = e_\alpha$ or $f = e'_\alpha$ in Proposition 25, we immediately obtain

Theorem 26. *The statistical distance between e_α and u satisfies*

$$d(e_\alpha, u) \le \frac{1}{2} \sum_{y\in\mathbb{F}_q, y\neq 0} |\widehat{e_\alpha}(y)|.$$

Similarly,

$$d(e'_\alpha, u) \le \frac{1}{2} \sum_{y\in\mathbb{F}_q, y\neq 0} |\widehat{e'_\alpha}(y)|. \tag{3}$$

Now let $\epsilon'(m, q, k, \alpha)$ denote the right hand side of (3), i.e.,

$$\epsilon'(m, q, k, \alpha) = \frac{1}{2} \sum_{y\in\mathbb{F}_q, y\neq 0} \prod_{i=0}^{n-1} \cos\left(\frac{\alpha^i \pi y}{q}\right)^k.$$

To take into account all prime ideals above q, we let α run through all primitive m-th roots of unity in $\mathbb{F}_q$ and define

$$\epsilon'(m,q,k) := \max\{\epsilon'(m,q,k,\alpha) : \alpha \text{ has order } m \text{ in } \mathbb{F}_q\}.$$

If $\epsilon'(m,q,k)$ is negligibly small, the distribution $P'_{m,k} \pmod{\mathfrak{q}}$ will be computationally indistinguishable from uniform.

We can run the same analysis for the PLWE distribution, with the only difference being that there is no obvious closed-form formula for the density function d of $D_{\mathbb{Z},\tau} \pmod{q}$. Nonetheless, we could numerically approximate this probability density function, using the formula

$$d(a) = \frac{\sum\limits_{\substack{z\in\mathbb{Z} \\ z\equiv a \bmod q}} e^{-|z|^2/2\tau}}{\sum\limits_{z\in\mathbb{Z}} e^{-|z|^2/2\tau}}, \quad \forall a \in \mathbb{F}_q.$$

Since the sums in the definition of $d(a)$ converge rapidly, we could obtain good approximations of d by truncating the sums and then evaluating. Then we compute numerically the Fourier transform $\hat{d}$, and obtain

$$\widehat{e_\alpha}(y) = \prod_{i=0}^{n-1} \hat{d}(\alpha^i y)$$

Finally, we compute $\epsilon(m,q,\tau) = \frac{1}{2}\sum_{y\in\mathbb{F}_q, y\neq 0} \prod_{i=0}^{n-1} \widehat{d}(\alpha^i y)$. Then $\epsilon(m,q,\tau)$ is an upper bound of the statistical distance between the distribution e_α and the uniform distribution over $\mathbb{F}_q$.

8.2 Numerical Distance from Uniform

We have computed $\epsilon'(m,q,k)$ and $\epsilon(m,q,k)$ for various choices of parameters. Smaller values imply that the error distribution looks uniform when transfered to $R/\mathfrak{q}$, rendering the instance of RLWE invulnerable to the attacks suggested in this paper.

The following is a table of data. Note that we chose $k = 2$ and $\tau = 1$. For each instance in the table, we also generated the actual RLWE samples (where we fixed $\sigma_0 = 1$) and ran the chi-square attack using the confidence level $\alpha = 0.99$. The column labeled "χ^2" contains the χ^2 values we obtained, and the column labeled "uniform?" indicates whether the reduced errors are uniform. We can see from data how the practical situation agrees with our analysis on the approximated distributions.

Table 3. Values of $\epsilon'(m,q,2)$ and $\epsilon(m,q,1)$ and the χ^2 values

m	n	q	$-[\log_2(\epsilon'(m,q,2))]$	$-[\log_2(\epsilon(m,q,1))]$	χ^2	uniform?
96	32	193	35	38	231.6	yes
55	40	331	44	51	308.8	yes
160	64	641	55	79	658.0	yes
101	100	1213	177	203	1254.4	yes
244	120	1709	230	248	1721.2	yes
256	128	3329	194	253	3350.0	yes
197	196	3547	337	410	3475.2	yes
512	256	10753	431	511	10732.8	yes

The data in Table 3 shows that when $n \geq 100$ and the size of the modulus q is polynomial in n, the statistical distances between $P'_{m,k} \pmod{\mathfrak{q}}$ (or $P_{m,\tau} \pmod{\mathfrak{q}}$) and the uniform distribution are both negligibly small. Also, note that we fixed $k = 2$ and $\tau = 1$, and the epsilon values becomes even smaller when k and τ increases.

It is possible to generalize our discussion in this section to primes of arbitrary residue degree, in which case the Fourier analysis will be performed over the corresponding extension field. The only change in the definitions would be $\chi_y(a) = e^{\frac{2\pi i Tr(ay)}{q}}$. Similarly, we have

$$\widehat{e'_\alpha}(y) = \prod_{i=1}^{n} \cos\left(\frac{\pi Tr(\alpha^i y)}{q}\right)^k.$$

Table 4 contains some data for primes of degree two.

Table 4. Values of $\epsilon'(m, q, 2)$ for primes of degree two

m	n	q	$-[\log_2(\epsilon'(m, q, 2))]$
64	32	383	31
63	36	881	33
55	40	109	48
53	52	211	61
512	256	257	263

8.3 Heuristics

There is a heuristic argument as to why one expects $\epsilon'(m, q, k, \alpha)$ to be small. Each term in the summand is a product of form $\prod_{i=0}^{n-1} \cos\left(\frac{\alpha^i \pi y}{q}\right)^k$. For each $0 \neq y \in \mathbb{F}_q$, if one assumes the elements α^i are distinct and uniformly distributed in $\mathbb{F}_q$, it is very likely that $\alpha^i y$ is close to $q/2$ for at least some values of i, making the product of cosines small.

References

1. Bos, J.W., Lauter, K., Loftus, J., Naehrig, M.: Improved security for a ring-based fully homomorphic encryption scheme. In: Cryptography and Coding, pp. 45–64. Springer (2013)
2. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 309–325. ACM (2012)
3. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Advances in Cryptology–CRYPTO 2011, pp. 505–524. Springer (2011)
4. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. SIAM Journal on Computing 43(2), 831–871 (2014)
5. Ducas, L., Durmus, A.: Ring-lwe in polynomial rings. In: Public Key Cryptography–PKC 2012, pp. 34–51. Springer (2012)
6. Eisenträger, K., Hallgren, S., Lauter, K.: Weak instances of plwe. In: Selected Areas in Cryptography–SAC 2014, pp. 183–194. Springer (2014)
7. Elias, Y., Lauter, K., Ozman, E., Stange, K.: Provably weak instances of ring-lwe. In: Advances in Cryptology – CRYPTO 2015, Lecture Notes in Comput. Sci., vol. 9215, pp. 63–92. Springer, Heidelberg (2015)
8. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Advances in Cryptology–EUROCRYPT 2012, pp. 465–482. Springer (2012)
9. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the fortieth annual ACM symposium on Theory of computing. pp. 197–206. ACM (2008)
10. Johnston, H.: Notes on galois modules. Notes accompanying the course Galois Modules given in Cambridge in (2011)
11. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Designs, Codes and Cryptography 75(3), 565–599 (2014)
12. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the forty-fourth annual ACM symposium on Theory of computing. pp. 1219–1234. ACM (2012)
13. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM (JACM) 60(6), 43 (2013)

14. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-lwe cryptography. In: Advances in Cryptology–EUROCRYPT 2013, pp. 35–54. Springer (2013)
15. Ryabko, B.Y., Stognienko, V., Shokin, Y.I.: A new test for randomness and its application to some cryptographic problems. Journal of statistical planning and inference 123(2), 365–376 (2004)
16. Stehlé, D., Steinfeld, R.: Making ntru as secure as worst-case problems over ideal lattices. In: Advances in Cryptology–EUROCRYPT 2011, pp. 27–47. Springer (2011)
17. Stein, W., et al.: Sage Mathematics Software (Version 6.4). The Sage Development Team (2014), http://www.sagemath.org

9 Appendix: Code

9.1 SubgroupModm.sage

This file contains the object needed for manipulating subgroups H of $(\mathbb{Z}/m\mathbb{Z})^*$.

```
class SubgroupModm:
    """
    a subgroup of (Z/mZ)^*
    """

    def __init__(self,m, gens, elements = None):
        self.m = m
        self.phim = euler_phi(m)
        self.Zm = Integers(m)

        newgens = []
        for a in gens:
            a = self.Zm(a)
            if not a.is_unit():
                raise ValueError('the generator %s must be a unit in the ambient group.'%a)
            newgens.append(a)

        self.gens = newgens

        if elements is None:
            print 'computing group elements...'
            t = cputime()
            self.H1 = self.compute_elements()
            print 'Time = %s'%cputime(t)
            sys.stdout.flush()
        else:
            self.H1 = elements

        self.order = len(self.H1)
        print 'group order = %s'%self.order
        sys.stdout.flush()

        self._degree = ZZ(self.phim // self.order)

        print 'computing coset representatives...'
        t = cputime()
        self.cosets = self.cosets()
        print 'Time = %s'%cputime(t)
        sys.stdout.flush()

        self._is_totally_real = self.is_totally_real()

        if not self._is_totally_real:
            merged_cosets = []
            for c in self.cosets:
                if not any([-c/d in self.H1 for d in merged_cosets]):
                    merged_cosets.append(ZZ(c))
            newcosets = merged_cosets + [-a for a in merged_cosets]
            self.cosets = newcosets

    def __repr__(self):
        return "subgroup of (Z/%sZ)^* of order %s generated by %s"%(self.m, self.order, self.gens)

    def is_totally_real(self):
        """
        The fixed field Q(zeta_m)^H is totally real if and only if -1 mod m \in H.
        """
        return self.Zm(-1) in  self.compute_elements()
```

```
def compute_elements(self):
    """
    core function. Gives all the group elements
    """
    gens = self.gens
    result = [self.Zm(1)]
    for gen in gens:
        if gens != self.Zm(1):
            order = gen.multiplicative_order()
            pows = [gen**j for j in range(order)]
            result = set([a*b for a in result for b in pows])
    return result

def cosets(self):
    """
    another core function, assuming we have elements, this shouldn't be hard.
    """
    Zm = self.Zm
    elts = self.H1
    m = self.m
    result =[]
    explored = []

    for a in range(m):
        if gcd(a,m) == 1 and a not in explored:
            for h in elts:
                explored.append(h*a)
            result.append(Zm(a))
        if euler_phi(m) ==  len(result)*len(elts): # already have enough cosets
            return result

@cached_method
def coset(self, a):
    """
    elt -- an integer
    returns the coset representative for this element
    """
    Zm = self.Zm
    for bb in self.cosets:
        if Zm(a)/Zm(bb) in set(self.H1):
            return bb
    raise ValueError('did not find a coset.')

def extension_degree(self,vec):
    """
    vec -- a vector indexed by cosets of self, representing an element z in K.
    return the degree of the extension QQ(z)/QQ.
    """
    try:
        vec = list(vec)
    except:
        raise ValueError('input can not be turned into a list. Please debug.')
    C = self.cosets
    ele_dict = dict([(a,b) for a,b in zip(C,vec) if b != 0])
    fixGpLen = 0
    for ll in C:
        fixed = True
        for a in ele_dict.keys():
            lla = self.coset(ll*a)
            try:
                coef = ele_dict[lla]
            except:
                fixed = False
                break
            if coef != ele_dict[a]:
                fixed = False
                break
        if fixed:
            fixGpLen += 1
    return self._degree // fixGpLen


def _check_cosets(self):
    """
    sanity check that the cosets has been computed correctly.
    """
    H1 = self.H1
    cosets = self.cosets
    from itertools import combinations
    return not any([c[1]*c[0]**(-1) in H1 for c in combinations(cosets, 2)])
```

```
    def __hash__(self):
        return hash((self.m,tuple(self.gens)))


    def _associated_characters(self):
        """
        Definition: a Dirichlet character chi of modulus m is associated to
        a subgroup H <= Z/mZ)^* if chi|_H = 1.

        return all the associated characters of self.
        """
        m, Zm = self.m, self.Zm
        G = DirichletGroup(m)
        H1 = Set(self.compute_elements())

        result =[]
        for chi in G:
            ker_chi = Set([Zm(a) for a in chi.kernel()]) # a list of integers
            if H1.issubset(ker_chi):
                result.append(chi)
        return result

    def multiplicative_order(self, a):
        """
        return the multiplicative order of [a] in the quotien group G/H
        """
        m = self.m
        Zm = self.Zm
        if gcd(m,a) != 1:
            raise ValueError
        a = Zm(a)
        o = self._degree
        for dd in o.divisors()[:-1]:
            if a**dd in self.H1:
                return dd
        return o

    def discriminant(self):
        """
        return, up to sign, the discriminant of the fixed field of self as a subfield of Q(zeta_m).
        """
        return prod([chi.conductor() for chi in self._associated_characters()])

    def intersection(self, other):
        """
        intersection of two subgroups of the same m.
        """
        if self.m != other.m:
            raise ValueError('the underlying m of self and other must be same.')
        H1 = self.H1
        H1other = other.H1
        Hnew = Set(H1).intersection(Set(H1other))
        print 'size of intersection = %s'%len(Hnew)
        Hnew_reduced = _reduce_gens(self.m,Hnew)
        print 'reduced gens for intersection  = %s'%Hnew_reduced
        sys.stdout.flush()
        return SubgroupModm(self.m, Hnew_reduced, elements = Hnew)

def _reduce_gens(m,H1):
    """
    given a full group, get a short list of generators.
    """
    Zm = Integers(m)
    gens = set([])
    gensSpan = set([Zm(1)])
    for a in H1:
        if Zm(a) not in gensSpan:
            sys.stdout.flush()
            ordera = Zm(a).multiplicative_order()
            alst  = [Zm(a)**j for j in range(1, ordera)]
            newelts = set([cc*aa for cc in gensSpan for aa in alst])
            gensSpan  |=  newelts
            gens.add(a)
        if len(gensSpan) == len(H1):
            # found enough generators.
            return list(gens)
    raise ValueError('did not find enough generators.')
```

9.2 MyLatticeSampler.sage

This file allows sampling from discrete lattice Gaussian distributions using the algorithm in [9]. It took the current implementation in sage and modified it slightly to fix some issues. The authors claim no originality of any code in this file.

```
from sage.stats.distributions.discrete_gaussian_integer import DiscreteGaussianDistributionIntegerSampler
def _fpbkz(A, K = 10**20, block = 8, delta = 0.75):
    """
    including a transpose operation.
    """
    print 'blocksize for bkz = %s'%block
    At = A.transpose()
    RF = A[0][0].parent()
    AA = Matrix(ZZ, [[ZZ(round(K*a)) for a in row] for row in list(At)])
    F = FP_LLL(AA)
    F.BKZ(block_size = block, delta= delta)
    B = F._sage_()
    T = B*AA**(-1)
    B1 = Matrix(RF, [[a/RF(K) for a in row] for row in list(B)])
    return T.transpose().change_ring(ZZ), B1.transpose()


class MyLatticeSampler:
    """
    Sampling from discrete Gaussian.
    """

    def __init__(self,A,sigma = 1,dps = 60, method = 'LLL', block = None, already_orthogonal = False, gram_schmidt_norms = None):
        self.A = A # we are using column span instead of rowspan
        self.sigma = sigma

        print 'reducing the lattice...'
        t = cputime()
        self._degree = A.nrows()
        if method == 'LLL':
            self.T =  self._lll_reduce()
        elif method == 'BKZ':
            self.T = self._bkz_reduce(block = block)
        else:
            print 'no reduction is done.'
            self.T = identity_matrix(self._degree)
        self.B  = self.A*self.T
        print 'reduction done. Time: %s'%cputime(t)


        print 'Gram Schmidting...'
        t = cputime()

        if already_orthogonal: # The columns of A are already gram-schmidt.
            self._G = self.A
            if gram_schmidt_norms is None:
                self._gs_norms = [self._G.column(i).norm() for i in range(self._degree)]
            else:
                self._gs_norms = gram_schmidt_norms
        else:
            # Compute the gram-schmidt ourselves. Can be slow.
            self._gs_norms, self._G = self.compute_G(dps =  dps)
        print 'Gram Schmidt done. Time: %s'%cputime(t)

        self.final_sigma = sigma*(prod(self._gs_norms))**(1/self._degree)


    def _bkz_reduce(self,block = None):
        print 'bkz being performed...'
        if block is None:
            block = min(50, ZZ(self._degree // 2))
        return _fpbkz(self.A, block = block)[0]

    def _lll_reduce(self):
        print 'lll being performed...'
        A = self.A
        return gp(A).qflll().sage()

    @cached_method
    def col_sum(self):
        """
        related to the evaluation attack, return the list a where
                    a[i] = colsum(A^-1,i)
        """
```

```
    return vector([1 for _ in range(self._degree)])*(self.A**(-1))

def babai_quality(self):
    """
    inspired by Kim's explanation, I think the quality of a basis
    for babai should be the ratio ||\tilde{bn}||/||\tilde{b1}||
    """
    gs_norms = self._gs_norms
    return float(min(gs_norms)/max(gs_norms))

def __repr__(self):
    return 'Discrete Gaussian sampler with dimension %s and sigma = %s'%(self._degree, self.final_sigma.n())

def compute_G(self, dps = 50):
    t = cputime()
    B = self.B
    n = self._degree
    from mpmath import *
    mp.dps = dps
    prec = dps*6
    AA = mp.matrix([list(w) for w in list(B)])
    Q,R = qr(AA) # QR decomposition

    M = mp.matrix([list(Q.column(i)*R[i,i]) for i in range(n)]);
    M_sage = Matrix([[RealField(prec)(M[i,j]) for i in range(n)] for j in range(n)])
    verbose('gram schmidt computation took %s'%cputime(t))
    return [abs(RealField(prec)(R[i,i])) for i in range(n)], M_sage

def set_sigma(self,newsigma):
    self.final_sigma = newsigma

def babai(self,c):
    """
    run babai's algorithm and find a lattice vector close to the
    input point c.
    Note this is super similar to the __call__ function

    Returns a tuple (v,z), where v is the actual vector in R^n,
    and z is its coordinate *in terms of a*. So we have
    v = Az.
    """
    n = self._degree
    try:
        c = vector(c)
    except:
        pass
    G, norms  = self._G, self._gs_norms
    B = self.B
    T = self.T
    zs = []
    v = c

    for i in range(n)[::-1]:
        b_ = G.column(i)
        v_ = v.dot_product(b_) / norms[i]**2
        z = ZZ(round(v_))
        v = v - z*B.column(i)
        zs.append(z)
    return c - v, T*(vector(zs[::-1]))

def __call__(self, c = None):
    """
    c -- an n-dimensional vector, so that we are sampling a discrete gaussian
    centered at c.
    """
    v = 0
    sigma, G = self.final_sigma, self._G
    n = self._degree
    if c is None:
        c = zero_vector(n)
    B = self.B
    T = self.T
    zs = []
    norms = self._gs_norms
    for i in range(n)[::-1]:
        b_ = G.column(i)
        c_ = c.dot_product(b_) / norms[i]**2
        sigma_ = sigma/norms[i]
        assert(sigma_ > 0)
        z = DiscreteGaussianDistributionIntegerSampler(sigma=sigma_, c=c_, algorithm="uniform+table")()
        c = c - z*B.column(i)
```

```
            v = v + z*B.column(i)
            zs.append(z)
        return v, T*vector(zs[::-1])
```

9.3 SubCycSampler.sage

This file allows generating the errors and reducing them modulo prime ideals when the field K is a subfield of some cyclotomic field with odd and squarefree m.

```
from sage.stats.distributions.discrete_gaussian_integer import DiscreteGaussianDistributionIntegerSampler
import sys

class SubCycSampler:
    """
    We write our own GPV sampler for sub-cyclotomic fields.
    It also has the functionality of simulating an attack.

    Caution: according to GPV, we need to have s >= ||tilde(B)||*log(n)
    for the sampler to approximate discrete lattice Gaussian. So if
    s is smaller than what is required, the __call__() method is not
    guaranteed to output discrete Gaussian.
    """

    def __init__(self,m,H,sigma = 1,prec = 100, method = 'BKZ',block = None):
        """
        require: m must be square free and odd.

        disc: the discriminant of K =  Q(zeta_m)^H. We pass it
        as an optional parameter, since when the order of H is
        large, the computation could be very slow.
        """

        self.m = m
        self.H = H

        self.H1 =self.H.H1
        sys.stdout.flush()


        self.cosets = H.cosets

        self.sigma = sigma
        self.prec = prec


        t = cputime()
        self._degree = euler_phi(m) // len(self.H1)

        self._is_totally_real = self.H._is_totally_real


        print 'computing embedding matrix...'
        t = cputime()
        self.TstarA, self.Acan = self.embedding_matrix(prec = self.prec)
        self.Acaninv  = None
        print 'time = %s'%cputime(t)
        sys.stdout.flush()

        self.D = MyLatticeSampler(self.TstarA, sigma = self.sigma, method = method, block = block)
        self.Ared = self.D.B

        self._T = self.D.T

        self.final_sigma  =self.D.final_sigma
        self.secret = self.__call__()


    def __repr__(self):
        return 'RLWE error sampler with m = %s,  H = %s, secret  = %s and sigma = %s'%(self.m, self.H, self.secret, self.final_sigma.n())

    def minpoly(self):
        K.<z> = CyclotomicField(self.m)
        return sum([z**h for h in self.H1]).minpoly()

    def compute_G(self, prec = 53):
        """
        computing a colum gram-schmidt basis for the embedded lattice O_K.
        return the basis and the length of each vector as a list.
```

```
    Modified on 8/2: do this after using LLL to reduce the basis.
    """
    B = self.Ared
    n = self._degree
    from mpmath import *
    mp.dps = prec // 2
    BB = mp.matrix([list(ww) for ww in list(B)])
    Q,R = qr(BB) # QR decomposition
    M = mp.matrix([list(Q.column(i)*R[i,i]) for i in range(n)]);
    M_sage = Matrix([[RealField(prec)(M[i,j]) for i in range(n)] for j in range(n)])
    v = [abs(R[i,i]) for i in range(n)]
    return M_sage,v # vectors are columns

def degree_of_prime(self,q):
    """
    return the degree of q in K
    """
    if not q.is_prime():
        raise ValueError('q must be prime')
    return (self.H).multiplicative_order(q)

def degree_n_primes(self, min_prime, max_prime, n =1):
    """
    return a bunch of primes of degree n in K. When n = 1, this
    is split primes.
    """
    result = []
    for p in primes(min_prime, max_prime):
        try:
            if self.degree_of_prime(p) == n:
                result.append(p)
        except:
            pass
    return result

def basis_lengths(self):
    return [self.Ared.column(i).norm() for i in range(self._degree)]


def galois_permutation(self, c):
    """
    c -- a coset.
    returns a dictionary d such that  d[a] = \sigma_c(a),
    representing a Galois group action.
    """
    H = self.H
    Zm = Integers(self.m)
    c = Zm(c)
    d = {}
    for a in self.cosets:
        d[a] = self.H.coset(a*c)
    return d

def _vec_modq_coset_dict(self,q):
    vec = self.vec_modq(q)
    cc = self.cosets
    return dict(zip(cc,vec))

def vec_modq_twisted_by_galois(self,q,c, reduced = False):
    _dict = self._vec_modq_coset_dict(q)
    _galois = self.galois_permutation(c)
    result  = []
    for a in self.cosets:
        result.append(_dict[_galois[a]])
    if not reduced:
        return vector(result)
    else:
        return vector(result)*self._T

def embedding_matrix(self, prec = None):
    """
    We are in a simplified situation because the field K is Galois over QQ,
    so it is either totally real or totally complex.
    to-do: can optimize this.
    """
    m = self.m
    H1 = self.H1
    if prec is None:
        prec = self.prec
    C = ComplexField(prec)
```

```
    zetam = C.zeta(m)
    cosets = self.cosets
    n = self._degree

    _dict = {}
    for l in cosets:
        _dict[l]  = sum([zetam**(ZZ(l*h)) for h in H1])

    A = Matrix([[_dict[self.H.coset(l*k)] for l in cosets] for k in cosets])

    if self._is_totally_real:
        Areal = _real_part(A)
        return Areal, A
    else:
        T = t_matrix(n,prec = prec)
        return _real_part(T.conjugate_transpose()*A),A

def coset_reps(self):
    """
    I need this for representing the basis vectors. Each coset rep c
    represents the element \alpha_c =  \sum_{h \in H} \zeta_m^{ch}.
    """
    return self.cosets

def __call__(self,c = None):
    """
    return an integer vector a = (a_c) indexed by the coset reps of self,
    which represents the vector \sum_c a_c \alpha_c
    Use the algorithm of [GPV].
    http://www.cc.gatech.edu/~cpeikert/pubs/trap_lattice.pdf

    If minkowski = True, return the lattice vector in R^n. Otherwise,
    return the coordinate of the vector in terms of the embedding matrix of self.
    """
    return self.D(c = c)[1]

def babai(self,c):
    return self.D.babai(c)[1]

def _modq_dict(self,q):
    """
    a sanity check of the generators modulo q.
    """
    cc = self.cosets
    vv = self.vec_modq(q)
    return dict(zip(cc,vv))

def subfield_quality(self):
    """
    portion of elements of our reduced basis that lie in proper subfields.
    """
    T = self._T
    count = 0
    for i in range(S._degree):
        col  = T.column(i)
        sys.stdout.flush()
        deg = S.H.extension_degree(col)
        print 'degree of Q(b_i) = %s'%deg
        sys.stdout.flush()
        if deg < S._degree:
            count += 1
    return float(count/S._degree)



def subfield_quality_modq(self,q, twist = None):
    if twist is None:
        vq = self.vec_modq(q,reduced = True)
    else:
        vq = self.vec_modq_twisted_by_galois(q,twist, reduced = True)
    F = vq[0].parent()
    deg = F.degree()
    return float(len([aa for aa in vq if aa.minpoly().degree() < deg])/self._degree)


@cached_method
def vec_modq(self,q, reduced = False):
    """
    the basis elements (normal integral basis) modulo q.

    If reduced is true, return the LLL-reduced basis mod q
```

```
        v dot Tz = (vT) dot z
        """
        m = self.m
        degree = self.degree_of_prime(q)
        v = finite_cyclo_traces(m,q,self.cosets,self.H1, deg = degree) # could be slow
        if not reduced:
            result = vector(v)
        else:
            result =  vector(v)*self._T
        return result

    def _to_ccn(self, lst):
        """
        convert an element in O_K from C^n to Z^n.
        """
        return list(self.Acan*vector(lst))


    def _to_zzn(self,lst):
        """
        the inversion of the above.
        """
        if self.Acaninv is None:
            self.Acaninv = (self.Acan)**(-1)
        return list(self.Acaninv*vector(lst))


    def _prod(self,lsta, lstb):
        """
        multiplying two field elements using the canonical embedding
        """
        lsta, lstb = list(lsta), list(lstb)
        lsta_cc, lstb_cc = self._to_ccn(lsta), self._to_ccn(lstb)
        float_result = self._to_zzn([aa*bb for aa, bb in zip(lsta_cc,lstb_cc)])
        return [ZZ(round(tt.real_part())) for tt in float_result]

    def set_sigma(self,newsigma):
        self.D.final_sigma = newsigma

    def set_secret(self, newsecret):
        self.secret = newsecret
```

9.4 Chisquare.sage

This file implements the a variant of the chi-square test over finite fields F_{q^f} for q a prime and $f > 1$.

```
def subfield_unifrom_test(samples, probThreshold = 1e-5):
    """
    Assume that the samples are from a finite field.
    we separate the ones that are from a proper subfield.
    """
    F = samples[0].parent()
    q = F.characteristic()
    degF = F.degree()
    numsamples = len(samples)
    eltsWithFullDegree = elts_of_full_degree(q,degF)
    nSmall = 0
    nLarge = 0
    for aa in samples:
        if aa.minpoly().degree() < degF:
            nSmall +=1
        else:
            nLarge +=1
    card = q**degF
    eLarge= float(eltsWithFullDegree/card*numsamples)
    eSmall= numsamples - eLarge
    verbose('eSmall, eLarge = %s,%s'%(eSmall, eLarge))
    verbose('nSmall, nLarge = %s,%s'%(nSmall, nLarge))
    if min(eSmall, eLarge) < 5:
        raise ValueError('samples size too small.')

    chisquare = (nSmall - eSmall )^2/eSmall + (nLarge - eLarge)^2/eLarge
    T = RealDistribution('chisquared', 1)
    verbose('chisquare = %s'%chisquare)
    prob = 1 - T.cum_distribution_function(chisquare)
    if prob < probThreshold:
        verbose('non-uniform')
```

```
        return False
    else:
        verbose('uniform')
        return True
```

9.5 Example of an attack

This code implements the attack on an Galois RLWE instance described in Section 5.3.

```
print 'We peform the full attack on an Galois instance.'

totaltime = cputime()
import sys

load('SubgroupModm.sage','MyLatticeSampler.sage','SubCycSampler.sage','Chisquare.sage')

def _my_dot_product(lst1,lst2):
    return sum([a*b for a,b in zip(lst1,lst2)])

m = 3003; H = SubgroupModm(m, [2276, 2729, 1123]);
S = SubCycSampler(m,H);

sigma0 = 1.0

S = SubCycSampler(m,H,prec = 300, method = 'LLL', sigma = sigma0)

print 'S = %s'%S

q = 131
degq = H.multiplicative_order(q)
print 'degree of prime q = %s is %s'%(q, degq)
sys.stdout.flush()

print 'final sigma = %s'%S.final_sigma

print 'degree of field = %s'%(euler_phi(m)//H.order)
sys.stdout.flush()

numsamples = 1000;
print 'generating %s errors...'%numsamples
sys.stdout.flush()
errors = []
for dd in range(numsamples):
    error = S()
    errors.append(error)
    if dd > 0 and Mod(dd,1000) == 0:
        print '%s/%s samples generated'%(dd, numsamples)
        print 'an example error is %s'%error
        sys.stdout.flush()
print 'error generation done.'
sys.stdout.flush()
save(errors, 'errors.sobj')

vq = S.vec_modq(q)
print 'vq = %s'%vq
sys.stdout.flush()
F = vq[0].parent()
sys.stdout.flush()

Flst = [a for a in F]
alpha = F.gen()
Fp = F.prime_subfield()
print 'defining polynomial of F = %s'%alpha.minpoly()
sys.stdout.flush()

print 'Generating uniform a...'

alst = [[ZZ.random_element(q) for _  in range(S._degree)] for jj in range(numsamples)]
print 'Generation of uniform a done.'
sys.stdout.flush()

s = [ZZ.random_element(q) for _  in range(S._degree)]
print 'secret = %s'%s
sys.stdout.flush()

# The attack.
success, SUCCESS = True, True
count = 1
for cc in S.cosets:
```

```
    t = cputime()
    success = True
    print 'coset %s/%s with representative %s'%(count, S._degree, cc)
    sys.stdout.flush()

    count += 1
    vqcc = S.vec_modq_twisted_by_galois(q,cc)
    smodq = F(_my_dot_product(s,vqcc))
    print 'smodq = %s'%smodq
    sys.stdout.flush()

    amodqlst,bmodqlst = [],[]
    for a,e in zip(alst, errors):
        emodq = F(_my_dot_product(e,vqcc))
        amodq = F(_my_dot_product(a,vqcc))
        amodqlst.append(amodq)
        bmodqlst.append(amodq*smodq+emodq)

    countsmall = 0
    for sguess in Flst:
        countsmall +=1
        sys.stdout.flush()
        if Mod(count, 1000) == 0 or sguess == smodq:
            print 'example run: %s/%s runs'%(count,len(Flst))
            print 'sguess = %s'%sguess
            if sguess == smodq:
                print 'this is the correct guess'
            set_verbose(1)
        reducedErrors = [bb - aa*sguess for aa, bb in zip(amodqlst, bmodqlst)]
        uniform = subfield_unifrom_test(reducedErrors,  probThreshold = 1e-10)
        if uniform and sguess == smodq:
            print 'failed to detect'
            success = False
            break
        elif (not uniform) and sguess != smodq:
            print 'uniform is distorted'
            success = False
            break
        set_verbose(0)
    print 'Done computing with coset [%s]. success = %s'%(cc, success)
    sys.stdout.flush()

    print 'Time taken = %s'%cputime(t)
    SUCCESS = SUCCESS and success
    sys.stdout.flush()
    print '*'*20

print '#'*40
print 'Summary:'
print 'H = %s'%H
print 'degree of field = %s'%(euler_phi(m)//H.order)
print 'q = %s, degree of q =  %s'%(q, degq)
print 'sigma_0 = %s'%sigma0
print 'number of samples = %s'%numsamples
print 'success? : %s'%SUCCESS
print 'Total Time = %s'%cputime(totaltime)
sys.stdout.flush()
```