

RUHR-UNIVERSITÄT BOCHUM

Side-Channel Attacks On Implementations Of Lattice-Based Cryptosystems

Julian Speith, Felix Haarmann

Seminarausarbeitung

June 18, 2016

Embedded Security Group - Prof. Dr.-Ing. Christof Paar

Abstract

The following text¹ describes what your abstract should be about.

The abstract should convey to the reader concisely and accurately within the space of a few sentences, the claim to knowledge that the authors are making. It should indicate the boundaries of space and time within which the enquiry has occurred. If there is a claim to generality beyond the boundaries of the enquiry the basis of that claim should be given, for example that a random sample is thought to be representative of a larger population. There should also be a hint of the method of enquiry.

The boundaries of an enquiry are important - and are unfortunately too often omitted from abstracts. This is due to the regrettable tendency for researchers to generalise their results from, for example, a few schools to all schools, and to imply that what was true at a particular time, is true for all time. Some reference to the geographical location of the children, or teachers, or schools on whom the claim to knowledge rests should be made. Because of the international nature of the research community it is worth making clear in what country the research took place. Also the period in which the data was collected should be stated.

The abstract should be a condensation of the substance of the paper, not a trailer, nor an introduction. Journals and thesis regulations usually put a limit of around 200 to 300 words to the length of an abstract. Trailer is a term borrowed from the cinema industry to describe a showing of a few highlights in order to win an audience. An Introduction tells that something is coming, but doesn't reveal its substance. These are not what is needed.

Abstracts are recycled in abstract journals and electronic networks and provide the main vehicle for other researchers to become aware of particular studies. Hence the more clearly they convey the claim to knowledge of the original paper the more useful they are in helping the reader to decide whether it is worth taking the trouble to obtain and read the original and possibly cite it in his/her own writing.

Both the abstract and the paper should make sense without the other.

¹Shamelessly ripped from <http://www.leeds.ac.uk/educol/abstract.htm>

Contents

1. Introduction	2
1.1. Related Work	2
1.2. Structure of this Paper	3
2. Theoretical Background	4
2.1. Notation	4
2.2. Ideal Lattices	4
2.3. Learning with Errors Problem	5
2.4. Discrete Gaussian Distribution	5
2.5. Cryptographic Algorithms	5
2.5.1. Ring-LWE Encryption Scheme	5
2.5.2. BLISS Signature Scheme	6
2.6. Side-Channel Attack Terminology	7
3. Masking the Ring-LWE Encryption Scheme I	8
3.1. Implementation	8
3.1.1. Overview	8
3.1.2. Masked Decoder	9
3.2. Evaluation	11
4. Masking the Ring-LWE Encryption Scheme II	12
4.1. Implementation	12
4.2. Evaluation	12
5. Flush+Reload Cache Attack on Bliss	13
5.1. Gaussian Sampling	13
5.1.1. CDT Sampling	13
5.1.2. Rejection Sampling	13
5.2. Attacking the Sampling Algorithms	13
5.3. Evaluation	13
6. Blinding Countermeasures	14
6.1. Blinding Polynomial Multiplication	14
6.2. Blinding Gaussian Sampling	15
7. Conclusion	17

A. Bibliography**20**

Acronyms

1. Introduction

Despite the rapid progress in the development of quantum computers and the hereby increasingly urgent need for post-quantum cryptographic algorithms, no such algorithms has yet been standardized [CJL⁺16]. Current public-key cryptosystems like RSA, DHKE or even elliptic curve cryptography could easily be broken by a quantum computer, due to Shor’s algorithm for prime factorization and discrete logarithms [Sho97]. As most of today’s digital infrastructure depends (at least partially) on such public-key algorithms, the need for efficient and secure cryptography that can withstand the power of quantum computation is as high as never before.

Lattice-based cryptography is the most promising of all attempts in post-quantum cryptography, as its underlying mathematics are already well understood and reasonably efficient implementations of some of the proposed cryptographic schemes are available today. Our paper will give an overview over some selected lattice-based algorithms and their implementation in respect to their resistance to various side-channel attack techniques.

1.1. Related Work

This paper summarizes the content of several other papers, that have been published in recent years. Some of them are referred to below and we strongly recommend to take a look at them.

Shortly after the ring-LWE problem was introduced in [LPR12] in 2012, the authors of [RRVV15] laid the groundwork for masked implementations of one ring-LWE encryption scheme and refined it in [RdCR⁺16] by getting rid of the need for a masked decoder. Just a year after the ring-LWE encryption scheme was introduced, the authors of [DDLL13] proposed the BLISS signature scheme, which is as well based on the ring-LWE problem. A possible side-channel attack on a slightly altered version of that signature scheme was then shown in [BHLY16] in 2016, which might be prevented by the blinding techniques used by the authors of [Saa16].

1.2. Structure of this Paper

In Section 2 we will start with an explanation of our notation and give an overview over the mathematic background needed to understand this paper. This includes introducing the reader to the concept of *(ideal) lattices*, the *learning with errors problem (LWE)*, *Discrete Gaussian Distributions*, the *ring-LWE Encryption Scheme* and the *BLISS Signature Scheme*. Additionally, we will give a short explanation of the side-channel attack terminology used throughout this paper.

Section 3 will deal with the ring-LWE encryption scheme and will be split into two parts, starting with the description of a masked implementation of the decryption function, including a masked decoder build upon a masked table lookup. The second part of this section will be an evaluation of the proposed implementation in respect to its soundness to first- and second-order side-channel attacks.

A different approach to masking of the ring-LWE encryption scheme will be presented in Section 4 of our paper, which will as well be split into a description of the proposed scheme and an evaluation. Furthermore, the second masking scheme will be compared to the first one in respect to efficiency and complexity. Section 5 will discuss the **FLUSH+RELOAD** cache attacks on the Gaussian sampler used in the BLISS signature scheme. This part will start with a description of a perfect side channel attack on two Gaussian sampling algorithms, namely the cumulative distribution function (CDT sampling) and rejection sampling. This will be followed by an evaluation of the **FLUSH+RELOAD** attacks on an actual BLISS implementation, while running on modern CPUs.

Furthermore, in Section 6 we will be presenting two measures used for blinding polynomial multiplication and Gaussian sampling, which might help against the attacks described in Section 5.

Finally, Section 7 will summarize the content of our paper shortly and some conclusions will be drawn.

2. Theoretical Background

2.1. Notation

As we will only work with ideal lattices in our paper, all operations will be done within the ring $R_q = \mathbb{Z}_q[x]/(f(x))$ with $f(x)$ being an irreducible polynomial of degree n and all coefficients being reduced modulo q .

Polynomials will be written as bold lower case letters (**f**). As we will use the *NTT* for efficient polynomial multiplication within a ring R_q , polynomials in the NTT domain will be written as $\tilde{\mathbf{f}}$. Vectors will be denoted with an arrow on top of them (\vec{x}), while matrices will be denoted by bold upper case letters (**A**). The entries of a vector \vec{x} will be called x_i , with i specifying the position within the vector starting at 0.

The notation for the l_p norm of a vector \vec{x} will be $\|\vec{x}\|_p$, only with the exception of the l_2 norm, which will be referred to as $\|\vec{x}\| = \sqrt{\sum_i x_i^2}$.

2.2. Ideal Lattices

A lattice Λ is discrete subgroup of \mathbb{R}^n that is defined as a set of $m \leq n$ linearly independent vectors $\vec{b}_1, \dots, \vec{b}_m \in \mathbb{R}^n$ and is generated by all linear combinations of those \vec{b}_i 's with integer coefficients:

$$\Lambda(\vec{b}_1, \dots, \vec{b}_m) = \left\{ \sum_{i=1}^m x_i \vec{b}_i \mid x_i \in \mathbb{Z} \right\} \quad (2.1)$$

The set $\{\vec{b}_1, \dots, \vec{b}_m\}$ of those vectors is called the basis of that lattice, which commonly is represented by a matrix $\mathbf{B} = (\vec{b}_1, \dots, \vec{b}_m)$.

Furthermore, an ideal lattice is a lattice that corresponds to ideals in a ring R_q . From this it follows that we can deal with polynomials instead of matrices, which makes arithmetics used for cryptographic applications much more efficient. In our paper we will confine ourselves to those ideal lattices, as most of the current work in that area focuses around them. For more on the topic of ideal lattices, see [LPR12].

2.3. Learning with Errors Problem

2.4. Discrete Gaussian Distribution

The discrete Gaussian distribution with mean μ and standard deviation σ is denoted as $\mathcal{N}_{\mathbb{Z}}(\mu, \sigma^2)$. In this paper we will focus on zero-centered distributions $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$ with a density function $\rho_{\sigma}(x)$ given by:

$$\rho_{\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \quad (2.2)$$

The probability function of discrete Gaussian distribution over \mathbb{Z} is then defined as $D_{\sigma}(x) = \rho_{\sigma}(x)/\rho_{\sigma}(\mathbb{Z})$ with $\rho_{\sigma}(\mathbb{Z}) = \sum_{y=-\infty}^{\infty} \rho_{\sigma}(y)$. As we will sample entire vectors most of the time, we denote the discrete Gaussian distribution over \mathbb{Z}^m as $\mathcal{N}_{\mathbb{Z}}^m(\mu, \sigma^2)$ and its probability function as $D_{\sigma}^m(\vec{x}) = \rho_{\sigma}(\vec{x})/\rho_{\sigma}(\mathbb{Z})^m$ with $\rho_{\sigma}(\vec{x})$ being defined as follows:

$$\rho_{\sigma}(\vec{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\|\mathbf{x}\|^2}{2\sigma^2}} \quad (2.3)$$

2.5. Cryptographic Algorithms

This Section will give a short overview of two cryptographic algorithms, that are based on ideal lattices and the ring-LWE problem. While the first algorithm can exclusively be used for encryption, the second one is a signature algorithm. For more information on the mathematical background of those algorithms, we would like to refer you to the cited papers.

2.5.1. Ring-LWE Encryption Scheme

In our paper we focus on the encryption scheme published in [LPR12], which will be referred to as *LPR*. This scheme consists of three main operations: *Key Generation*, *Encryption* and *Decryption*. The globally known parameters of this scheme are (n, q, σ) and a polynomial \mathbf{g} . The dimension of the polynomial ring R_q is defined by n , while q is the modulus. The standard deviation of the discrete Gaussian distribution is given by σ .

Key Generation: In this step, the coefficients of the two polynomials \mathbf{r} and \mathbf{s} are sampled according to the discrete Gaussian distribution $\mathcal{N}_{\mathbb{Z}}^n(0, \sigma^2)$. Then the public key \mathbf{p} is computed by $\mathbf{p} = \mathbf{r} - \mathbf{g} \cdot \mathbf{s}$. The resulting output is a key pair (\mathbf{p}, \mathbf{s}) with \mathbf{p} being the public key and \mathbf{s} being the secret key.

Encryption: The encryption phase takes a n -bit message \mathbf{m} and the public key \mathbf{p} as input. Initially, the message \mathbf{m} is encoded as an element of the ring

R_q by multiplying each of its bits by $q/2$ and is then denoted by \mathbf{m}_{enc} . In the following step, three error polynomials \mathbf{e}_1 , \mathbf{e}_2 and \mathbf{e}_3 are sampled according to $\mathcal{N}_{\mathbb{Z}}^n(0, \sigma^2)$ and will be used as noise. The ciphertext then consists of two parts $(\mathbf{c}_1, \mathbf{c}_2)$, with $\mathbf{c}_1 = \mathbf{g} \cdot \mathbf{e}_1 + \mathbf{e}_2$ and $\mathbf{c}_2 = \mathbf{p} \cdot \mathbf{e}_1 + \mathbf{e}_3 + \mathbf{m}_{enc}$. The encryption algorithm returns the ciphertext $(\mathbf{c}_1, \mathbf{c}_2)$.

Decryption: For decryption, we start by computing $\mathbf{m}_{enc} = \mathbf{c}_1 \cdot \mathbf{s} + \mathbf{c}_2$. To decode \mathbf{m}_{enc} , we need a function $\text{DECODE}(m_{enc,i})$ with $m_{enc,i}$ being an element of \mathbf{m}_{enc} . One possible function is given below:

$$\text{DECODE}(x) = \begin{cases} 0, & \text{if } x \in (0, q/4) \cup (3q/4, q) \\ 1, & \text{if } x \in (q/4, 3q/4) \end{cases} \quad (2.4)$$

2.5.2. BLISS Signature Scheme

Algorithm 1 BLISS KEY GENERATION

Output: BLISS key pair (\mathbf{A}, \mathbf{S}) with public key $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2) \in R_{2q}^2$ and secret key $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2) \in R_{2q}^2$, such that $\mathbf{AS} = \mathbf{a}_1 \cdot \mathbf{s}_1 + \mathbf{a}_2 \cdot \mathbf{s}_2 \equiv q \pmod{2q}$

- 1: Choose $\mathbf{f}, \mathbf{g} \in R_{2q}$ uniformly at random with exactly d_1 entries in $\{\pm 1\}$ and d_1 entries in $\{\pm 2\}$
- 2: $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2) = (\mathbf{f}, 2\mathbf{g} + 1)$
- 3: **if** \mathbf{f} violates certain conditions (see [DDLL13]) **then**
- 4: Restart
- 5: **end if**
- 6: $\mathbf{a}_q = (2\mathbf{g} + 1)/\mathbf{f} \pmod{q}$ (restart if \mathbf{f} is not invertible)
- 7: **return** (\mathbf{A}, \mathbf{S}) with $\mathbf{A} = (2\mathbf{a}_q, q - 2) \pmod{2q}$

Algorithm 2 BLISS SIGNATURE ALGORITHM

Input: Message μ , public key $\mathbf{A} = (\mathbf{a}_1, q - 2)$, secret key $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)$

Output: Signature $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c}) \in \mathbb{Z}_{2q}^n \times \mathbb{Z}_p^n \times \{0, 1\}^n$

- 1: $\mathbf{y}_1, \mathbf{y}_2 \leftarrow \mathcal{N}_{\mathbb{Z}}^n(0, \sigma^2)$
 - 2: $\mathbf{u} = \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \pmod{2q}$
 - 3: $\mathbf{c} = H(\lfloor \mathbf{u} \rfloor_d) \pmod{p, \mu}$
 - 4: Choose a random bit b
 - 5: $\mathbf{z}_1 = \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \cdot \mathbf{c} \pmod{2q}$
 - 6: $\mathbf{z}_2 = \mathbf{y}_2 + (-1)^b \mathbf{s}_2 \cdot \mathbf{c} \pmod{2q}$
 - 7: Continue with a probability based on σ , $\|\mathbf{Sc}\|$, $\langle \mathbf{z}, \mathbf{Sc} \rangle$ (see [DDLL13]), else restart
 - 8: $\mathbf{z}_2^\dagger = (\lfloor \mathbf{u} \rfloor_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rfloor_d) \pmod{p}$
 - 9: **return** $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$
-

Algorithm 3 BLISS VERIFICATION ALGORITHM

Input: Message μ , public key $\mathbf{A} = (\mathbf{a}_1, q - 2)$, signature $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$

- 1: **if** $\mathbf{z}_1, \mathbf{z}_2^\dagger$ violate certain conditions (see [DDLL13]) **then**
- 2: Reject
- 3: **end if**
- 4: **if** $\mathbf{c} = H(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rfloor_d + \mathbf{z}_2^\dagger \bmod p, \mu)$ **then**
- 5: Accept
- 6: **end if**

2.6. Side-Channel Attack Terminology

Side-channel attacks use leaked information from physical implementations of cryptographic algorithms to conclude secret information like encryption keys. Such leakage could e.g. be the power consumption during cryptographic operations. One type of attacks that makes use of leakages through power consumption is *Differential Power Analysis* (DPA).

To perform a DPA, one needs to correlate a leakage with a prediction made on a special value that depends on both, the secret key and the plaintext. Such a value is called a *sensitive variable*. A common countermeasure to DPA is masking, where sensitive variables are randomly split into d shares. A masking approach with d shares is referred to as a $(d - 1)$ -th order masking, as $d - 1$ shares are picked randomly and the last share is computed in a way, that the combination of all d shares equals the shared sensitive variable. To defeat this kind of countermeasures, the class of *Higher Order DPA* (HO-DPA) has been introduced. To overcome $(d - 1)$ -th order masking, d -th order DPA is needed, which can be done by combining the leakage of d different signals that correspond to the d shares of the sensitive variable. Theoretically, higher order masking can always be defeated by HO-DPA. However, the difficulty of higher order DPA is growing exponentially due to noise effects. In practice, first order masking is most commonly used, thus there is a big focus on second order DPA in research.

3. Masking the Ring-LWE Encryption Scheme I

Since most side-channel attacks focus on the decryption operation, this section will present an attempt to masking the decryption function of the *LPR* Ring-LWE encryption scheme. This masking approach was originally proposed in [RRVV15], for more details we would refer you to that paper.

3.1. Implementation

We will start by giving the reader an overview of the general setup, before going into more detail about the masked decoding algorithms. We will make strong use of the *Number Theoretic Transform* (NTT) in this chapter. We recall, that our notation for polynomials in the NTT domain is $\tilde{\mathbf{f}}$. The NTT operation itself will be denoted as $\text{NTT}(\cdot)$, while its inverse operation will be written as $\text{INTT}(\cdot)$. We want to stress, that $\text{NTT}(\cdot)$ and $\text{INTT}(\cdot)$ are linear operations, as we will use this characteristic for our blinding technique.

3.1.1. Overview

This Subsection will cover a concise overview of the blinding technique proposed in [RRVV15]. For the sake of simplicity, the intermediate m_{enc} will be referred to as a in the following.

We start by splitting the secret key \mathbf{s} into two shares $\mathbf{s}', \mathbf{s}'' \in R_q$ such that $\mathbf{s} = \mathbf{s}' + \mathbf{s}''$. Therefore we choose all coefficients of \mathbf{s}' uniformly at random and calculate $\mathbf{s}'' = \mathbf{s} - \mathbf{s}'$. In the NTT domain it follows that $\tilde{\mathbf{s}} = \tilde{\mathbf{s}}' + \tilde{\mathbf{s}}''$. Due to the linearity of $\text{INTT}(\cdot)$ and the multiplication, we can compute \mathbf{a} as:

$$\mathbf{a} = \text{INTT}(\tilde{\mathbf{s}} \cdot \tilde{\mathbf{c}}_1 + \tilde{\mathbf{c}}_2) = \text{INTT}(\tilde{\mathbf{s}}' \cdot \tilde{\mathbf{c}}_1 + \tilde{\mathbf{c}}_2) + \text{INTT}(\tilde{\mathbf{s}}'' \cdot \tilde{\mathbf{c}}_1) \quad (3.1)$$

This enables us to split the whole equation into two branches, calculating \mathbf{m}'_{enc} and \mathbf{m}''_{enc} in the following way:

$$\mathbf{a}' = \text{INTT}(\tilde{\mathbf{s}}' \cdot \tilde{\mathbf{c}}_1 + \tilde{\mathbf{c}}_2), \mathbf{a}'' = \text{INTT}(\tilde{\mathbf{s}}'' \cdot \tilde{\mathbf{c}}_1) \quad (3.2)$$

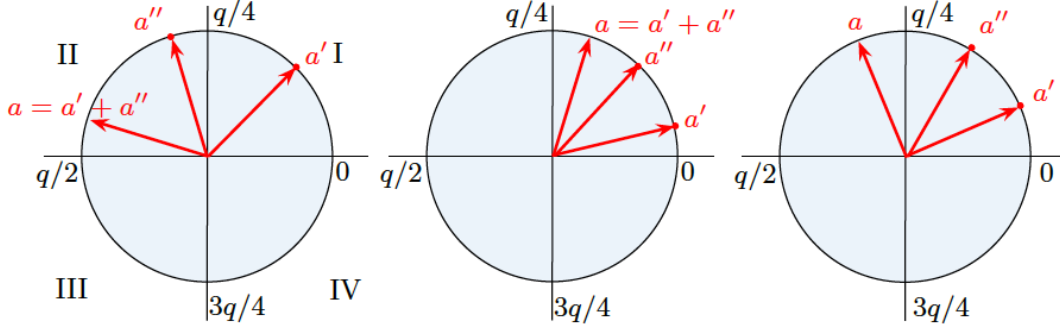


Figure 3.1.: The basic idea of our masked decoder. The circle represents elements in \mathbb{Z}_q . The first case shown allows us to conclude $\text{DECODE}(a) = 1$, while we cannot make any guesses about the last two ones. [RRVV15]

Those computations can be done on an arithmetic processor without any protection against side-channel attacks like *DPA*, as both branches are totally independent of our secret key \mathbf{s} .

However, the $\text{DECODE}(a_i)$ function in the decryption stage of the *LPR* scheme is non-linear and cannot easily be split into two parts. For this reason, we will present a masked decoder in the next Subsection, that takes \mathbf{a}' and \mathbf{a}'' as inputs to compute two shares \mathbf{m}' , \mathbf{m}'' of the decoded message \mathbf{m} in a fairly efficient way.

3.1.2. Masked Decoder

This Section briefly describes a probabilistic masked decoder. We recall, that the i -th element of a is called a_i and the shares (a'_i, a''_i) of such an element are chosen in a way, that $a'_i + a''_i = a_i \pmod{q}$. To keep it simple, we will refer to an arbitrary a_i as a , the same follows for its shares.

For our masked decoder, we do not need to know the exact values of a' and a'' to compute $\text{DECODE}(a)$. The following example will help us to lay down some rules for the decoder: Given (a', a'') with $0 < a' < q/4$ and $q/4 < a'' < q/2$. Then know that for $a = a' + a''$ it follows, that $q/4 < a < 3q/4$ and therefore $\text{DECODE}(a) = 1$. We only need to know the most significant bits of a' and a'' to determine, which values they are bounded by.

Figure 3.1 shows our example from above on the left. We can use this knowledge to state a total of four rules, the first of whom is taken from our example:

- $0 < a' < q/4, q/4 < a'' < q/2 \implies a \in (q/4, 3q/4) \implies \text{DECODE}(a) = 1$
- $q/2 < a' < 3q/4, 3q/4 < a'' < q \implies a \in (q/4, 3q/4) \implies \text{DECODE}(a) = 1$
- $q/4 < a' < q/2, q/2 < a'' < 3q/4 \implies a \in (0, q/4) \cup (3q/4, q) \implies \text{DECODE}(a) = 0$

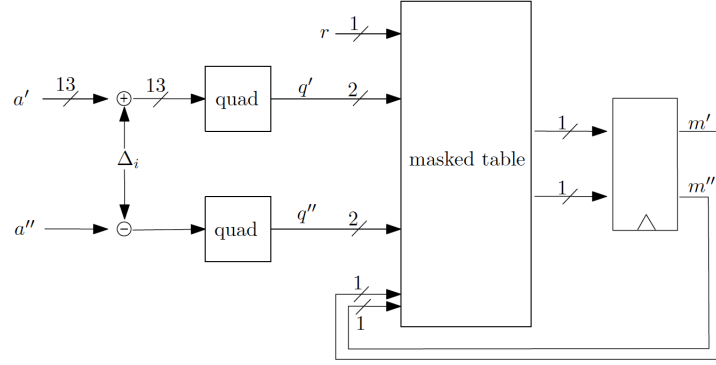


Figure 3.2.: Hardware implementation of the masked decoder. [RRVV15]

- $3q/4 < a' < q, 0 < a'' < q/4 \implies a \in (0, q/4) \cup (3q/4, q) \implies \text{DECODE}(a) = 0$

With swapping a' and a'' in the above rules, one can obtain another four rules. From the rules it follows, that we only need to know the quadrant of each a' and a'' to infer the output of $\text{DECODE}(a)$. However, this does not work for all cases, as Figure 3.1 shows. We can actually only apply those rules in half of the possible cases.

So, what happens if our (a', a'') does not match any rule? We simply need to refresh the splitting by computing $a' = a' + \Delta_1$ and $a'' = a'' - \Delta_1$ with $\Delta_i \in \mathbb{Z}_q$. From $(a' + \Delta_1) + (a'' - \Delta_1) = a' + a'' = a$ it follows, that a stays unchanged by that refresh. Now that we have a fresh pair (a', a'') , we can again try to apply our rules from above. This process can be repeated until all shares have been decoded. Note, that a new Δ_i should be chosen for each iteration. About half of the a', a'' are decoded per iteration, so that the amount of decoded shares rises exponentially with the number of iterations. The authors of [RRVV15] propose a number of $N = 16$ iterations for a satisfactory result.

A possible hardware implementation of such a masked decoder is shown in Figure 3.2. The refreshing step is depicted on the left, using a different Δ_i in each iteration. The quadrant function used in the next step simply takes a refreshed share a' or a'' as an input and outputs two bits depending on the quadrant that the share belongs to. Next, a masked table is used to check the two bits against the rules we described above. Finally, the masked table function returns two one bit shares of the decoded message m . In our implementation the masked takes additional inputs, like a random bit r and the output of the last iteration (m'_{i-1}, m''_{i-1}) . For more details on the masked table lookup, we would like to refer you to the paper of Oscar Reparaz et. al. [RRVV15].

3.2. Evaluation

4. Masking the Ring-LWE Encryption Scheme II

4.1. Implementation

4.2. Evaluation

5. Flush+Reload Cache Attack on Bliss

5.1. Gaussian Sampling

5.1.1. CDT Sampling

5.1.2. Rejection Sampling

5.2. Attacking the Sampling Algorithms

5.3. Evaluation

6. Blinding Countermeasures

Blinding is a countermeasure commonly used to prevent side-channel attacks like *Differential Power Analysis* (DPA) [KJJ99]. It is used to add additional randomness to mathematical operations in a way, that the attacker can not easily draw conclusions from his observations. This Section summarizes two blinding countermeasures presented in [Saa16], which appear to be of special interest for Ring-LWE cryptosystems. While the first countermeasure will be an approach to blinding of polynomial multiplication within a ring R_q , the second one will be a blinding countermeasure for Gaussian sampling. All those techniques are believed to help against the attacks we described in 5, yet this has not been verified.

6.1. Blinding Polynomial Multiplication

There are two pretty types of blinding for polynomial multiplications, the first of whom is the multiplication of each polynomial with a constant. For two polynomials $\mathbf{f}, \mathbf{g} \in R_q$ and constants $a, b \in \mathbb{Z}_q$ the blinding operation and the inverse operation look as follows:

$$\mathbf{h} = a\mathbf{f} \cdot b\mathbf{g} \tag{6.1}$$

$$\mathbf{f} \cdot \mathbf{g} = (ab)^{-1}\mathbf{h} \tag{6.2}$$

The second type would be circularly shifting the coefficients in each of the polynomials. As a polynomial can be written as $\mathbf{f} = \sum_{i=0}^{n-1} f_i x^i$ with f_i being the i -th coefficient of the polynomial \mathbf{f} , a shift by j positions would be equal to the following computation:

$$x^j \mathbf{f} = \sum_{i=0}^{n-1} f_i x^{i+j} = \sum_{i=0}^{n-1} f_{i-j} x^i \tag{6.3}$$

Both of those blinding operation can be combined within one function, which will be called $\text{POLYBLIND}(\mathbf{v}, s, c)$ from now on. This function works on the coefficient vectors of polynomials of degree n and is given in Algorithm 4.

The inverse operation can be denoted by $\text{POLYBLIND}(\vec{v}', -s, c^{-1})$. Due to the isometries of the ring R_q , the multiplication of two polynomials (here: their coefficient vectors) can be blinded using the POLYBLIND function in the following way:

Algorithm 4 POLYBLIND**Input:** coefficient vector \vec{v} , number of shifts s , constant c **Output:** blinded coefficient vector \vec{v}'

```

1: for  $i = 0, \dots, n - s - 1$  do
2:    $v'_i = cv_{i+s} \bmod q$ 
3: end for
4: for  $i = n - s, \dots, n - 1$  do
5:    $v'_i = q - cv_{i+s-n} \bmod q$ 
6: end for
7: return  $\vec{v}'$ 

```

$$\begin{aligned}
\vec{f}' &= \text{POLYBLIND}(\vec{f}, r, a) \text{ with } r \in_R 0, \dots, n - 1 \text{ and } a \in_R \mathbb{Z}_q \\
\vec{g}' &= \text{POLYBLIND}(\vec{g}, s, b) \text{ with } s \in_R 0, \dots, n - 1 \text{ and } b \in_R \mathbb{Z}_q \\
\vec{h}' &= \vec{f}' \cdot \vec{g}' \\
\vec{h} &= \text{POLYBLIND}(\vec{h}', -(r + s), (ab)^{-1})
\end{aligned}$$

6.2. Blinding Gaussian Sampling

As with the blinding of polynomial multiplication in the last subsection, there are two pretty easy ways to blind the coefficient vectors during the process of Gaussian sampling. We will again give a short description of both of them and present a function, that combines both methods.

We define a function $\text{VECTORSAMPLE}(n, \sigma)$, that samples and returns a vector according to the discrete Gaussian distribution $\mathcal{N}_{\mathbb{Z}}^n(0, \sigma^2)$. A naive implementation of this function could lead to leakage of information to an attacker using e.g. DPA. This has been done in the cache attack from [BHL16] we described in Section 5.

The first approach to blinding would be to randomly shuffle the elements in the coefficient vector. The function $\text{VECTORSHUFFLE}(\vec{x})$ is doing exactly that, so that $\text{VECTORSHUFFLE}(\text{VECTORSAMPLE}(n, \sigma))$ would increase security to a certain extend.

For the second approach we need to take a short detour through probability theory. For two Gaussian distributions $X = \mathcal{N}_{\mathbb{Z}}^n(\mu_X, \sigma_X^2)$ and $Y = \mathcal{N}_{\mathbb{Z}}^n(\mu_Y, \sigma_Y^2)$ it holds that their sum is equal to $X + Y = \mathcal{N}_{\mathbb{Z}}^n(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)$. As we focus on zero-centered distributions, the center does not change for us. For the standard deviation it follows, that $\sigma_{X+Y} = \sqrt{\sigma_X^2 + \sigma_Y^2}$. Algorithm 5 is one possible way to make use of those characteristics of Gaussian distributions. It also makes use of the $\text{VECTORSHUFFLE}(\vec{x})$ function to increase overall security. Another, very similar approach, has been described in [PDG14].

Algorithm 5 VECTORBLINDSAMPLE

Input: length of the vector n , number of iterations m , standard deviation σ

Output: sampled vector \vec{x}

- 1: $\mathbf{x} = \mathbf{0}$
 - 2: **for** $i = 1, \dots, m$ **do**
 - 3: $\vec{x} = \vec{x} + \mathcal{N}_{\mathbb{Z}}^n(0, (\frac{1}{\sqrt{m}}\sigma)^2)$
 - 4: $\vec{x} = \text{VECTORSHUFFLE}(\vec{x})$
 - 5: **end for**
 - 6: **return** \vec{x}
-

7. Conclusion

Conclude your thesis and discuss your results...

List of Figures

3.1.	The basic idea of our masked decoder. The circle represents elements in \mathbb{Z}_q . The first case shown allows us to conclude $\text{DECODE}(a) = 1$, while we cannot make any guesses about the last two ones.	
	[RRVV15]	9
3.2.	Hardware implementation of the masked decoder. [RRVV15] . . .	10

List of Tables

A. Bibliography

- [BHLY16] Leon Groot Bruinderink, Andreas Hlsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload – a cache attack on the bliss lattice-based signature scheme. Cryptology ePrint Archive, Report 2016/300, 2016.
- [CJL⁺16] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on post-quantum cryptography. Technical Report NIST IR 8105, National Institute of Standards and Technology (NIST), February 2016.
- [DDLL13] Lo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. Cryptology ePrint Archive, Report 2013/383, 2013.
- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.
- [LPR12] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230, 2012.
- [PDG14] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. Cryptology ePrint Archive, Report 2014/254, 2014.
- [RdCR⁺16] Oscar Reparaz, Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. *Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, chapter Additively Homomorphic Ring-LWE Masking, pages 233–244. Springer International Publishing, 2016.
- [RRVV15] Oscar Reparaz, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. A masked ring-lwe implementation. Cryptology ePrint Archive, Report 2015/724, 2015.
- [Saa16] Markku-Juhani O. Saarinen. Arithmetic coding and blinding countermeasures for ring-lwe. Cryptology ePrint Archive, Report 2016/276, 2016.

-
- [Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.