
**RUHR-UNIVERSITÄT** BOCHUM

# Side-Channel Attacks On Implementations Of Lattice-Based Cryptosystems

Julian Speith, Felix Haarmann

Seminarausarbeitung
June 23, 2016
Embedded Security Group - Prof. Dr.-Ing. Christof Paar

# Abstract

The following text[1] 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 doesnt 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.

[1] Shamelessly ripped from `http://www.leeds.ac.uk/educol/abstract.htm`

# Contents

# Acronyms

**DPA** Differential Power Analysis

**HO-DPA** Higher Order Differential Power Analysis

**NTT** Number Theoretic Transform

**ring-LWE** Learning with Errors Problem over Rings

**LPR** Lyubashevsky-Peikert-Regev

**PRNG** Pseudo Random Number Generator

# 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 logarothms [Sho97]. As most of todays 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] layed 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 blidning techniques used by the authors of [Saa16].

## 1.2. Structure of this Paper

In Section 2 we will start with an explaination 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 over Rings (ring-LWE)*, *Discrete Gaussian Distributions*, a *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 ($\mathbf{f}$). As we will use the *Number Theoretic Transform (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 ($\mathbf{A}$). 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 $\Lambda$ is discrete subgroup of $\mathbb{R}^n$ that is defined as a set of $m \leq n$ linearly independent vectors $\vec{b}_1, ..., \vec{b}_m \in \mathbb{R}^n$ and is generated by all linear combinations of those $\vec{b}_i$'s with integer coefficients:

$$\Lambda(\vec{b}_1, ..., \vec{b}_m) = \left\{ \sum_{i=1}^{m} x_i \vec{b}_i \;\middle|\; x_i \in \mathbb{Z} \right\} \tag{2.1}$$

The set $\{\vec{b}_1, ..., \vec{b}_m\}$ of those vectors is called the basis of that lattice, which commonly is represented by a matrix $\mathbf{B} = (\vec{b}_1, ..., \vec{b}_m)$.
Furthermore, an ideal lattice is is 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 $\mu$ and standard deviation $\sigma$ 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:

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

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 $\mathcal{N}_{\mathbb{Z}}^n(\mu, \sigma^2)$ 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:

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

## 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 refered to as *Lyubashevsky-Peikert-Regev (LPR)*. This scheme consists of three main operations: *Key Generation*, *Encryption* and *Decryption*. The globally known paramterers of this scheme are $(n, q, \sigma)$ 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 $\sigma$.

**Key Generation:** In this step, the coeficients 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 $\textsc{Decode}(m_{enc,i})$ with $m_{enc,i}$ being an element of $\mathbf{m}_{enc}$. One possible function is given below:

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

### 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 $\mathbf{AS} = \mathbf{a}_1 \cdot \mathbf{s}_1 + \mathbf{a}_2 \cdot \mathbf{s}_2 \equiv q \bmod 2q$

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: $\mathbf{a}_q = (2\mathbf{g} + 1)/\mathbf{f} \bmod q$ (restart if $\mathbf{f}$ is not invertible)
7: **return** $(\mathbf{A}, \mathbf{S})$ with $\mathbf{A} = (2\mathbf{a}_q, q - 2) \bmod 2q$

**Algorithm 2** BLISS Signature Algorithm

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

**Algorithm 3** BLISS Verification Algorithm

**Input:** Message $\mu$, 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** $\mathbf{c} = H(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rceil_d + \mathbf{z}_2^\dagger \bmod p, \mu)$ **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 sensitives variables are randomly split into $d$ shares. A masking approach with $d$ shares is reffered 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 Differential Power Analysis (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 *HO-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 Using a Masked Decoder

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

W 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 *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 NTT(.), while its inverse operation will be written as INTT(.). We want to stress, that NTT(.) and INTT(.) 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 $\mathbf{m}_{enc}$ will be referred to as $\mathbf{a}$ 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 INTT(.) and the multiplication, we can compute $\mathbf{a}$ as:

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

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

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

Those computations can be done on a 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 $\textsc{Decode}(a_i)$ 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 choosen 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 it's shares.
For our masked decoder, we do not need to know the exact values of $a'$ and $a''$ to compute $\textsc{Decode}(a)$. 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 $\textsc{Decode}(a) = 1$. We only need to know the most significant bits of $a'$ and $a''$ to determine, which values they are bounded by.

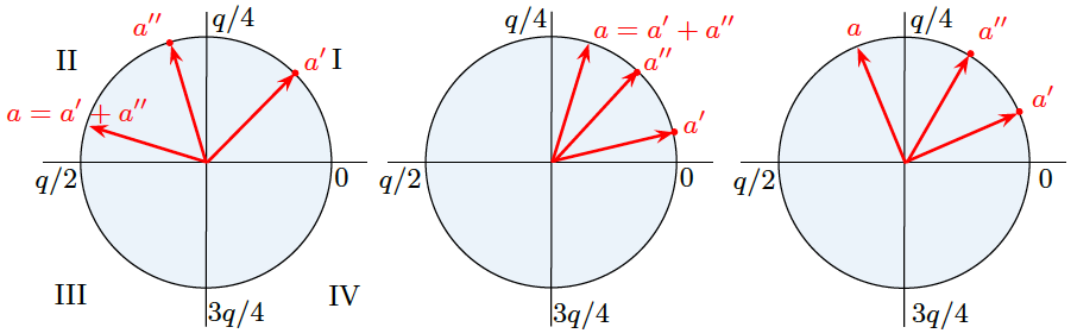


Figure 3.1.: The basic idea of our masked decoder. The circle represents elements in $\mathbb{Z}_q$. The first case shown allows us to conclude $\textsc{Decode}(a) = 1$, while we cannot make any guesses about the last two ones. [RRVV15]

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

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

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

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 $a'$ an $a''$ to infer the output of $\textsc{Decode}(a)$. 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 follow, 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 $\Delta_i$ should be choosen 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.

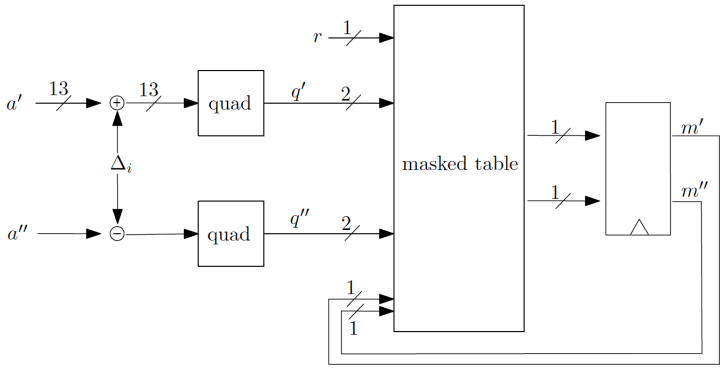


Figure 3.2.: Hardware implementation of the masked decoder. [RRVV15]

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 $\Delta_i$ 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

Starting with efficiency, Reparaz et. al. showed that this implementation is at least 1.9x times better on a Virtex-2 FPGA than an unprotected high-speed elliptic curve scalar multiplier architecture introduced in [RRM12].
Furthermore, as both, the *LPR ring-LWE* encryption scheme and our masked decoder, are probabilistic, there will always be a chance for errors occuring during decoding. The global error rate of decoding rises significantly when using our masked decoder instead of a deterministic decoder. To offset this effect, we can adapt the number of iterations for the masked decoder. While for $N = 3$ iterations the global error rate is about 49 times larger than when using a deterministic decoder, $N = 16$ iterations yield a global error rate that is almost identical with the one of a deterministic decoder. Further improvement could be achieved by increasing the number of iterations again, but this would lead to a significantly higher cycly count and thus to a much more inefficient implementation.
For our evaluation in terms of side-channel attack soundness, we assume the attacker knows the details of our implementation and is aware of the rest of the key while guessing a subkey. Our evaluation will follow three steps: First, we perform a first-order key recovery attack with our source of randomness (*PRNG*) turned off. This attack will be successful, showing that our setting is correct. In the second step, we will turn on the PRNG and repeat the same attack, but it should not be successful in this case. Finally, we perform a second-order attack to confirm the correctness of the first two steps.

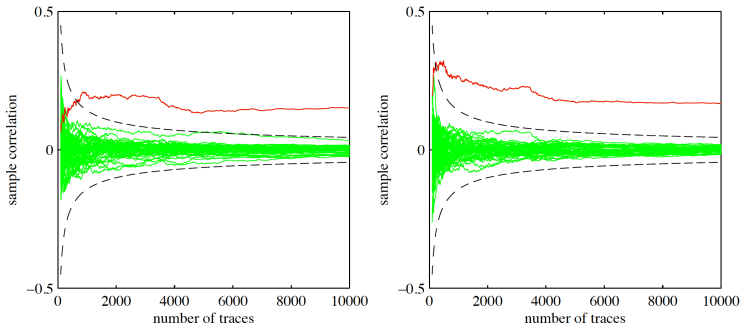


Figure 3.3.: *PRNG* is turned off. Graph shows the correlation coefficient increasing with the number of traces for the intermediates $a'_0$ (left) and $a''_0$ (right). The correct subkey is shown in red, all other guesses in green. [RRVV15]

For each of those steps, four different points that cover all relevant steps of the algorithm have been tested. The targets are $a'_0$, $a''_0$, the first input to the masked

decoder and the first output bit. Pearson's correlation coefficient has been used to compare our guesses with real measurements [BCO04].

***PRNG* off:** When the *Pseudo Random Number Generator (PRNG)* is turned off, sharing of $\mathbf{s}$ in $\mathbf{s}'$ and $\mathbf{s}''$ is deterministic. This translates to the masking being turned off. Figure 3.3 shows the correlation coefficient evolving with the number of traces. The attack seems to be successful starting at about a hundred traces.

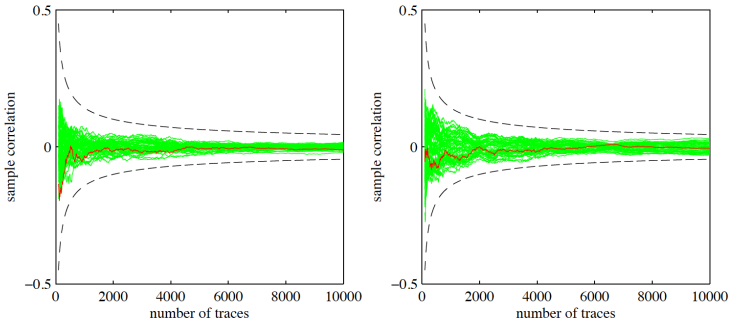


Figure 3.4.: Same as Figure 3.3, but with the *PRNG* turned on. It is no longer possible to identify the correct subkey within all guesses, meaning that the masking is successful. [RRVV15]

**PRNG on:** When the *PRNG* is turned on, the masking is effective. As we can see in Figure 3.4, the correct subkey can no longer be distinguished from all the other guesses, not even with an enormous amount of traces. This is what an attacker would see when conducting an first-order *DPA* on our masking scheme.

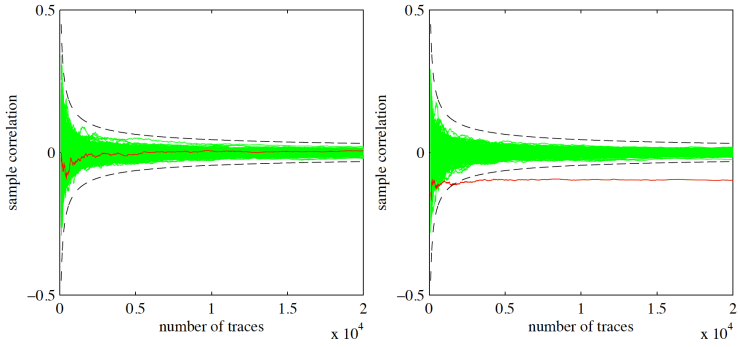


Figure 3.5.: On the left is the correlation for an increasing number of traces of a first-order attack with masking turned on. On the right we can see a successful second-order attack on our decoding scheme with masking turned on. [RRVV15]

**Second-Order Attack:** To confirm that we have used a sufficient number of

traces in the first two steps, we perform a second-order attack on our masking scheme. In Figure 3.5 we can see, that the second-order attack starts to be successful at around 2000 traces. From this we conclude, that we carried out the first-order attack on the activated masking scheme correctly, as we are already successful with 2000 traces. We stress that an attacker would need a significantly higher number of traces and computations in reality, as we used a pretty friendly setting for our scenario.

# 4. Additively Homomorphic ring-LWE Masking

Less than a year after [RRVV15] (which has been described in the last Section) was published, Reparaz et. al. published a follow-up paper [RdCR+16] introducing a much easier approach for the masking of the ring-LWE encryption scheme. In the following, the approach of said paper will be introduced and evaluated in terms of efficiency and side-channel attack resistance. In our description we again focus on the LPR scheme, though the techniques are also applicable for other ring-LWE encryption schemes.

## 4.1. Implementation

For ring-LWE masking, we make use of the fact that the LPR encryption scheme is additively homomorphic. Thus for given ciphertexts $(\mathbf{c}_1, \mathbf{c}_2)$ and $(\mathbf{c}'_1, \mathbf{c}'_2)$, which are the encryption of two messages $\mathbf{m}$ and $\mathbf{m}'$ with $m_i, m'_i \in \{0, 1\}$ using the same public key $\mathbf{p}$, it follows that $(\mathbf{c}_1 + \mathbf{c}'_1, \mathbf{c}_2 + \mathbf{c}'_2)$ is the encryption of $\mathbf{m} \oplus \mathbf{m}'$. As a consequence we can write down the following equation:

$$\mathit{decryption}(\mathbf{c}_1, \mathbf{c}_2) \oplus \mathit{decryption}(\mathbf{c}'_1, \mathbf{c}'_2) = \mathit{decryption}(\mathbf{c}_1 + \mathbf{c}'_1, \mathbf{c}_2 + \mathbf{c}'_2) \quad (4.1)$$

Now, we want to make use of the prperty of additive homomorphism for our masking scheme. This, again, focuses on the decryption function, as this is the part of the encryption scheme, where the secret key is used, which makes it a prime target for attackers.
To randomize the decryption of $(\mathbf{c}_1, \mathbf{c}_2)$, we need to follow three simple steps:

1. Generate a random message $\mathbf{m}'$ unknown to the adversary
2. Encrypt $\mathbf{m}'$ to $(\mathbf{c}'_1, \mathbf{c}'_2)$
3. Decrypt $(\mathbf{c}_1 + \mathbf{c}'_1, \mathbf{c}_2 + \mathbf{c}'_2)$ to receive $\mathbf{m} \oplus \mathbf{m}'$

The masked message returned by this approach is $(\mathbf{m}', \mathbf{m} \oplus \mathbf{m}')$, such that $\mathbf{m} = (\mathbf{m} \oplus \mathbf{m}') \oplus \mathbf{m}'$.
The advantage of this approach is, that no masked decoder is needed. For decoding, an unprotected decoder might be used without leaking any useful information for an attacker.

## 4.2. Evaluation

# 5. Flush+Reload Cache Attack on Bliss

## 5.1. Gaussian Sampling

### 5.1.1. CDT Sampling

Using the cumulative distribution function in the sampler, we build a large table, in which we approximate the probabilities $p_y = \mathbb{P}[x \leq y | x \leftarrow D_\sigma]$ with $\lambda$ Bits of precision. At sampling time, we generate a uniformly random $r \in [0, 1)$ and perform a binary search in the table to locate $y \in [-r\sigma, r\sigma]$, so $r \in [p_{y-1}, p_y)$. If restricted to the non-negative part $[o, r\sigma]$, the probabilities are $p_y^* = \mathbb{P}[|x| \leq y | x \leftarrow D_\sigma]$, sampling is still $r \in [0, 1)$, but $y \in [0, r\sigma]$ is located.

The binary search in this sampling method can take some time, so one can speed it up by using an additional *guide table I*. This table stores for example 256 entries consisting of intervalls $I[u] = (a_u, b_u), u \in \{0, ...255\}$ such that $p_{a_u}^* \leq u/256$ and $p_{b_u}^* \geq (u+1)/256$. At sampling time, the first byte of $r$ is then used to select the corresponding $I[u]$, which leads to a smaller interval to binary search. $r$ is effectively picked byte-by-byte using the guide table approach. Algorithm 4 summarizes the guide table approach.

### 5.1.2. Rejection Sampling

## 5.2. Attacking the Sampling Algorithms

## 5.3. Evaluation

**Algorithm 4** CDT Sampling With Guide Table

**Input:** Big table $T[y]$ containing values $p_y^*$ of the cumulative distribution function of the discrete Gaussian distribution (using only non-negative values), omitting the first byte. Samll table $I$ consisting of the 256 intervals.

**Output:** Value $y \in [-r\sigma, r\sigma]$, sampled with probability according to $D_\sigma$

pick a random byte $r$
Let $(I_{min}, I_{max}) = (a_r, b_r)$ be the left and right bounds of interval $I[r]$
**if** $I_{max} - I_{min} = 1$ **then**
    generate a random sign bit $b \in \{0, 1\}$
    **return** $y = (-1)^b I_{min}$
**end if**
Let $i = 1$ denote the index of the byte to look at
Pick a new random byte r
**while** 1 **do**
    $I_z = \lfloor \frac{I_{min}+I_{max}}{2} \rfloor$
    **if** $r >$ ($i$th byte of $T[I_z]$) **then**
        $I_{min} = I_z$
    **else if** $r <$ ($i$th byte of $T[I_z]$) **then**
        $I_{max} = I_z$
    **else if** $I_{max} - I_{min} = 1$ **then**
        generate a random sign bit $b \in \{0, 1\}$
        **return** $y = (-1)^b I_{min}$
    **else**
        increase $i$ by 1
        pick new random byte r
    **end if**
**end while**

# 6. Blinding Countermeasures

Blinding is a countermeasure commonly used to prevent side-channel attacks like *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 polinomial 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 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 5.

The inverse operation can be denoted by 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 5** PolyBlind

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

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

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

## 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 VectorSample$(n, \sigma)$, that samples and returns a vector according to the discrete Gaussian distribution $\mathcal{N}^n_{\mathbb{Z}}(0, \sigma^2)$. A naiv 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 [BHLY16] we described in Section 5.
The first approach to blinding would be to randomly shuffle the elements in the coeffcient vector. The function VectorShuffle$(\vec{x})$ is doing exactly that, so that VectorShuffle(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 $X = \mathcal{N}^n_{\mathbb{Z}}(\mu_X, \sigma^2_X)$ and $Y = \mathcal{N}^n_{\mathbb{Z}}(\mu_Y, \sigma^2_Y)$ it holds that their sum is equal to $X + Y = \mathcal{N}^n_{\mathbb{Z}}(\mu_X + \mu_Y, \sigma^2_X + \sigma^2_Y)$. As we focus on zero-centered distributions, the center does not change for us. For the standard deviation it follows, that $\sigma_{X+Y} = \sqrt{\sigma^2_x + \sigma^2_Y}$. Algorithm 6 is one possible way to make use of those characteristics of Gaussian distributions. It also makes use of the VectorShuffle$(\vec{x})$ function to increase overall security. Another, very similar approach, has been described in [PDG14].

**Algorithm 6** VectorBlindSample

**Input:** length of the vector $n$, number of iterations $m$, standard deviation $\sigma$
**Output:** sampled vector $\vec{x}$

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

# 7. Conclusion

Conclude your thesis and discuss your results...

# List of Figures

# List of Tables

# List of Algorithms

# A. Bibliography


[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. *Correlation Power Analysis with a Leakage Model*, pages 16–29. Springer Berlin Heidelberg, 2004.

[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, Tancrde 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.

[RRM12] Chester Rebeiro, Sujoy Sinha Roy, and Debdeep Mukhopadhyay. *Pushing the Limits of High-Speed $GF(2^m)$ Elliptic Curve Scalar Mul-*

*tiplication on FPGAs*, pages 494–511. Springer Berlin Heidelberg, 2012.

[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.